Abstract

ACHARYA, MITHUN PUTHIGE. NAMO NAMAHA: **N**etwork **A**ssisted **M**ulticast **O**verlay ConstructioN **A**lgorithms for **M**obile **A**d **H**oc **A**pplications (Under the direction of Douglas S. Reeves)[1]. Group communication is the most important mode of communication in ad hoc networks, because of the collaborative nature of mobile ad hoc applications. In this light, an efficient and light weight multicast routing protocol is necessary. Presently the multicast routing is either done entirely at the network layer, or at the application layer as stateless overlay mulitcasting. Owing to the dynamic nature of ad hoc networks, the first method incurs a large signaling overhead due to frequent modification of routing tables and exchanging of session state information. The latter approach uses the underlying unicast routing to build multicast data distribution trees without maintaining session state information thereby trading efficiency for minimal messaging overhead. For small groups with constant bound on the number of multicast group members, the overlay schemes, apart from having a trivial signaling overhead, are also known to be far more efficient than the network layer schemes. But the existing overlay schemes do not completely exploit the 'knowledge' possessed by the network layer; they just use the unicast routing at the network layer. We believe that, even for larger groups, the overlay schemes can function

[1]NAMO NAMAHA in Sanskrit means $salutations$ usually offered to the Almighty

with reasonable efficiency along with trivial signaling overhead if they intelligently use the network layer information.

In this thesis, we propose a network assisted multicast routing scheme, NAMO NAMAHA, which primarily operates as an overlay while getting assistance from the network layer unicast routing protocol, CEDAR. The overlay algorithms dynamically build an approximate Steiner data distribution tree, adopting the $(CHINS)_T$ (Cheapest Insertion Heuristic with Table) algorithm for the distributed implementation of the well known Takahashi-Matsuyama heuristic. The Steiner trees are incrementally built over a subgraph of $core\ nodes$, which form the approximate Minimum Dominating Set (MDS) over the network nodes. The $core\ nodes$ get computed by a network layer heuristic using local data at that layer, and they provide useful information for the NAMO NAMAHA tree computation algorithms at the overlay layer. The main idea in this thesis is that if the construction of overlays is aided by some minimal but useful information from the network layer that is almost invariant, local and that which would incur constant memory overhead, efficient overlays can be constructed.

This thesis presents the algorithms for the protocol NAMO NAMAHA, offers proof of correctness for the protocol and shows that the time and memory complexity of the algorithms in the protocol are either constant, or linear with the number of graph edges or nodes. We compare our work with the MCEDAR protocol in terms of the cost of the multicast data distribution trees, the number of messages exchanged in building them and the time and memory complexity of the algorithms involved. We choose MCEDAR since other multicast protocols for ad hoc networks are either network based which does not scale for large nubmer of nodes, or function as overlays designed only for small groups. When compared to MCEDAR, NAMO NAMAHA has a simpler $join$ protocol implemented by

our unique $Unicast\ Trap$ algorithm that does not make use of acknowledgements. Unlike MCEDAR, the sender discovery messages are not propagated all over the network; they are restricted to regions where it is absolutely necessary. In NAMO NAMAHA, at any given time, a path exists between any multicast subscriber and the sender (which is approximately the best path possible) with very high probability. Such a guarantee cannot be given in MCEDAR. Furthermore because of the incremental Steiner tree construction, the resulting multicast data distribution tree has nearly the least total cost. Cost is not minimized in MCEDAR. These advantages are obtained in NAMO NAMAHA just by using extra messages during tree construction, whose number is well below the actual number of nodes in the multicast group. The time and memory complexity of the NAMO NAMAHA algorithms are in the same order as that in MCEDAR.

NAMO NAMAHA: NETWORK ASSISTED MULTICAST OVERLAY CONSTRUCTION ALGORITHMS FOR MOBILE AD HOC APPLICATIONS

BY

MITHUN PUTHIGE ACHARYA

A THESIS SUBMITTED TO THE GRADUATE FACULTY OF

NORTH CAROLINA STATE UNIVERSITY

IN PARTIAL FULFILLMENT OF THE

REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE

RALEIGH

DECEMBER 2003

APPROVED BY:

DAVID J. THUENTE

MLADEN A. VOUK

DOUGLAS S. REEVES

CHAIR OF ADVISORY COMMITTEE

To my parents.

Biography

Mithun Acharya was born on July 23, 1979 in Udupi, Karnataka, India. He received his Bachelors degree in Computer Science from the Karnataka Regional Engineering College, Surathkal, India – now known as the National Institute of Technology, Surathkal – in the summer of 2001. He has been a Masters student at the North Carolina State University, in the Department of Computer Science, since August 2001. He will be pursuing his doctoral degree in the same department starting January 2004.

Acknowledgment

My foremost thanks to Dr. Douglas Reeves, my advisor, for all his innumerable tips about conducting productive research, constant guidance and inspiration. I thank Dr. Mladen Vouk and Dr. David Thuente for agreeing to be on my thesis committee. I thank Thomas Lofgren, Laura Feeney and Bengt Ahlgren of SICS, Sweden for all their help. I thank Prashant, Ranjana and everyone in Prof. Reeves' research group at Cyber Defense Lab for all the lively discussions and suggestions.

I thank my roomies, Kishan for debates on practical philosophy and spirituality, Mitul for all those long Lake Wheeler drives, racquet ball matches and rainy day vegetable delites, Ravi for his useful straightforward criticisms and for his amazing company during morning walks from home to my lab, and my very old pal from Udupi, Yathiraj (Bhattru), who can keep me happy just by his culinary abilities, not to mention many others. All my roomies have given me a home away from home. My gratitude is due to Pooja, for being a great friend all along, "Bhagwaan" Abhijit and everyone in the *"veryclose"* group, my seniors and my juniors for giving me the best time of my life, here in Raleigh.

I thank my labmates Ashok for his help with latex, Naveen for all those discussions on arbitrary algorithms and Shalu along with Susmita and Tanu for their good company.

Thank you NC State University and United States for providing me with all the facilities and environment to conduct research. A special mention to the Johnson Lake and Lake Wheeler staff, the rocking chairs, boats and the ducks over there, the place which was my second research office !

I thank Murthy mama for his advices and blessings, right from my childhood days, which have inspired and shaped my career. Finally I reserve my highest gratitude to my motherland India and to my parents. This thesis is dedicated to my parents, to whom I am forever indebted.

PS: To make my little brother, Madan, happy, I would like to thank him too for not disturbing me in my research work !

Table of Contents

List of Figures

Chapter 1

Introduction and Related Work

1.1 Mobile Ad Hoc Network (MANET)

Simply put, Mobile ad hoc networks, in short termed as MANETs, are dynamic multi hop wireless network that is established by a group of mobile nodes on a shared wireless channel by virtue of proximity to each other [SSB99a]. Currently, MANETs are one of the most widely researched areas in the field of Wireless Communication. This is because MANETs are really attractive for communication within a short range in situations where infrastructure support is an impossibility; like in military battlefields and disaster relief operations, the two most classic uses of MANETs. But with a wide range of compact wireless devices pouring into the market, the concept of MANET is now percolating more and more to the civilian world too. Wireless laptops and PDAs can collaborate in the MANET mode for classroom conferencing and the vehicles in a city can form an ad hoc network which can efficiently address the problem of traffic congestion. Sensor networks are another form of ad hoc networks, in which the nodes are really small, in terms of size, computing power and resources and they find immense use in active environmental monitoring and security surveillance.

These benefits however do not come for free. Due to the mobile nature of the nodes participating in the ad hoc networks, all the problems addressed and solved in wired networks will have to be revisited. Especially the problem of routing becomes very complicated. One side of the problem is that with the nodes moving randomly, the links can break anytime; the network may get partitioned anytime. The other side of the problem arises due to the low computational capabilities of the participating devices and scarce network bandwidth. Due to the mobility, a healthy percentage of traffic will be control overhead; in propagating the link and routing table information to the network members and these consume a good chunk of bandwidth. Scalability is another major problem. Routing schemes that perform very well with fewer nodes cripple the network if the number of nodes increase. These aspects make MANETs a very challenging area for research. Nevertheless the benefits far outweigh the challenges which make the idea of MANETs worth pursuing.

1.2 Multicasting in MANET and Related Work

One look at the type of applications for which MANETs are used, like classroom conferencing, military battlefields, disaster relief operations, city vehicular networks, convinces anyone that the applications are inherently collaborative in nature. More often than not, the form of communication in MANETs is group communication. Groups will be formed among the participating members that may communicate with each other to accomplish a particular task. Group communication easily surpasses communication between individual nodes because of this collaborative nature. Clearly the participating nodes need what is called as the $multicasting$ ability. Multicasting enables one to many communication or in some cases many to many communication between a group of nodes, usually identified by a single multicasting address. Well established routing protocols do exist to offer efficient

multicasting service in conventional wired networks. There exists a large amount of literature on multicast in wired and infrastructure wireless networks. [GCA02] gives a detailed insight into them. These protocols having been designed for fixed networks, may fail to keep up with node movements and frequent topology changes in MANET. As nodes become increasingly mobile, these protocols need to evolve to provide efficient service in the new environment. Therefore adopting the existing wired multicast protocols to MANETS that completely lack infrastructure, appears less promising. [dMCGA03] reviews the existing multicast routing protocols for MANETs which can be easily classified into four categories based on how routes are created to the members of the group:

- Tree-based approaches
- Mesh-based approaches
- Stateless multicast
- Hybrid approaches

We briefly inspect the protocols in each of this classification.

1.2.1 Tree Based Approaches

Tree based multicast is a very well established concept in wired networks. Most schemes for providing multicast in wired networks are either source- or shared-tree based.

AMRIS [WT99] is an on-demand protocol that constructs a shared multicast delivery tree to support multiple senders and receivers in a multicast session. AMRIS dynamically assigns an ID number to each node in each multicast session. Based on the ID number, a multicast delivery tree - rooted at a special node with Smallest-ID (Sid) - is created, and the ID number increases as the tree expands from the Sid. Generally, Sid is the source or the node that initiates a multicast session.

MAODV routing protocol [RP99] follows directly from unicast AODV, and discovers multicast routes on demand using a broadcast route discovery mechanism employing the same route request (RREQ) and route reply (RREP) messages that exist in the unicast AODV protocol. A mobile node originates an RREQ message when it wishes to join a multicast group, or has data to send to a multicast group but does not have a route to that group. Only a member of the desired multicast group may respond to a join RREQ. If the RREQ is not a join request, any node with a fresh enough route (based on group sequence number) to the multicast group may respond. If an intermediate node receives a join RREQ for a multicast group of which it is not a member, or it receives a RREQ and does not have a route to that group, it rebroadcasts the RREQ to its neighbors.

The LAM [JC98] protocol draws on the Core-Based Tree (CBT) algorithm [GCA02] and Temporal Ordering Routing Algorithm (TORA) in order to provide multicast services. Similar to CBT, it builds a group-shared multicast routing tree for each multicast group centered at the CORE. Nodes in LAM maintain two variables, POTENTIAL-PARENT and PARENT, and two lists, POTENTIAL-CHILD-LIST and CHILD LIST. The PARENT variable is used to remember the parent node in the multicast tree. The CHILD-LIST stores identities of one-hop children in the multicasting tree. The potential data objects are used when the nodes are in a “join” or “rejoin” waiting state.

1.2.2 Mesh Based Approaches

In contrast to a tree based approach, mesh based multicast protocols may have multiple paths between any source and receiver pair. Existing studies show that tree-based protocols are not necessarily best suited for multicast in a MANET where network topology changes frequently. In such an environment, mesh based protocols seem to outperform tree based proposals due to the availability of alternative paths, which allow multicast datagrams to

be delivered to the receivers even if links fail.

ODMRP [LSG02] is a mesh based protocol that uses a forwarding group concept (only a subset of nodes forwards the multicast packets). A soft state approach is taken in ODMRP to maintain multicast group members. No explicit control message is required to leave the group. In ODMRP group membership and multicast routes are established and updated by the source on demand. When a multicast source has packets to send, but no route to the multicast group, it broadcasts a Join-Query control packet to the entire network. This Join-Query control packet is periodically broadcast to refresh the membership information and updates routes. When an intermediate node receives the Join-Query packet, it stores the source ID and sequence number in its message cache to detect any potential duplicate. The routing table is updated with an appropriate node ID (i.e., backward learning) from which the message has been received. If the message is not a duplicate and the TTL is greater than zero, it is rebroadcast.

CAMP [MGLA01] supports multicasting by creating a shared mesh for each multicast group. Meshes thus created help maintain connectivity to multicast users, even with node mobility. It borrows concept from CBT, but unlike CBT where all traffic flows through the core node, the core nodes in CAMP are used to limit the control traffic needed for receivers to join multicast groups. The basic operation of CAMP includes building and maintaining the multicast mesh for a multicast group. It assumes a mapping service, which provides routers with the addresses of groups identified by their names. It also implies availability of routing information from a unicast routing protocol. Each router maintains a routing table (RT) built with the unicast routing protocol. This table is modified by CAMP when a multicast group needs to be inserted or removed. Based on RT, a multicast routing table (MRT) is built, which consists of a set of groups known to the router. A router may update its MRT based on topological changes or messages received from its neighbors.

FGMP [CGZ98] can be viewed as flooding with "limited scope", wherein the flooding is contained within selected forwarding group (FG) nodes. FGMP makes innovative use of flags and an associated timer to forward multicast packets. When the forwarding flag is set, each node in FG forwards data packets belonging to a group G until the timer expires. When a packet is forwarded, only the nodes with an enabled forwarding flag can accept the packet. This soft state approach of using a timer works well in dynamically changing environments. FGMP uses two approaches to elect and maintain FG of forwarding nodes: FGMP-RA (receiver advertising) and FGMP-SA (sender advertising)

1.2.3 Hybrid Approach

The tree based approaches provide high data forwarding efficiency at the expense of low robustness, whereas mesh based approaches provide better robustness (link failure may not trigger a reconfiguration) at the expense of higher forwarding overhead and increased network load. Thus, there is a possibility that a hybrid multicasting solution may achieve better performance by combining the advantages of both approaches.

AMRoute [LTM99] creates a bidirectional, shared tree using only group senders and receivers as tree nodes for data distribution. The protocol has two main components: mesh creation and tree setup. The mesh creation identifies and designates certain nodes as logical cores, and these are responsible for initiating the signaling operation and maintaining the multicast tree to the rest of the group members. A non-core node only responds to messages from the core nodes and serves as a passive agent. The selection of a logical core in AMRoute is dynamic and can migrate to any other member node depending on network dynamics and group membership. AMRoute does not address network dynamics and assumes that the underlying unicast protocol takes care of it. To create a mesh, each member begins by identifying itself as a core and broadcasts JOIN_REQ with increasing

time to live (TTL) to discover other members. When a core receives a JOIN_REQ from a core in a different mesh for the same group, it replies with JOIN_ACK. A new bidirectional tunnel is created between the two cores, and one of them is selected as a core after the mesh merger. Once the mesh has been established, the core initiates the tree creation process. The core sends out periodic TREE_CREATE messages along all links incident on its mesh. Using unicast tunnels, the TREE_CREATE messages are sent only to group members. Group members receiving nonduplicate TREE_CREATE message forwards it to all mesh links except the incoming one, and marks the incoming and the outgoing links as tree links. If a link is not going to be used as part of the tree, the TREE_CREATE is discarded and TREE_CREATE_NAK is sent back to incoming links. A member node that wants to leave a group can do so by sending a JOIN_NAK message to its neighbor nodes.

MCEDAR [SSB99b] is a multicast extension to CEDAR [SSB99a] architecture. The main idea of MCEDAR is to incorporate the efficiency of tree-based forwarding protocols and robustness of mesh based protocols by combining the two approaches. It decouples the control infrastructure from the actual data forwarding. It uses the mesh as the underlying infrastructure, so it can tolerate a few link breakages without reconfiguration of the infrastructure. The efficiency is achieved by using a forwarding mechanism on the mesh that creates an implicit route-based forwarding tree. This ensures that the packets need to travel only the minimum distance in the tree. More about this will be detailed in Chapter 3 and Chapter 5

1.2.4 Stateless Multicast

Tree and mesh based approaches have an overhead of creating and maintaining the delivery tree/mesh with time. In a MANET environment, frequent movement of mobile nodes considerably increases the overhead in maintaining the delivery tree/mesh. To minimize

the effect of such a problem, stateless multicasting is proposed wherein a source explicitly mentions the list of destinations in the packet header. Stateless multicast assumes small group multicasting.

DDM [JC01] protocol is meant for small multicasting groups operating in dynamic networks of any size. Unlike other MANET routing protocols, DDM lets the source control multicast group membership. The source encodes multicast receiver addresses in multicast data packets using a special DDM data header. This variable length destination list is placed in the packet headers, resulting in packets being self routed towards their destinations using the underlying unicast routing protocol. It eliminates maintaining per-session multicast forwarding states at intermediate nodes and thus is easily scalable with respect to number of sessions.

Location Guided Tree Construction [CN02] is another small group multicast which uses the approximation: "Smaller geometric distance implies fewer network hops" and uses location information in forming overlay trees on demand. Like DDM, here too addresses of the receivers are inserted in the data packet and the packets are self routed. [CN02] uses geometric distance as an approximation to the number of network hops and builds multicast data distribution trees. Packets are source routed from the sender to the receivers. This work, [CN02], is actually the basis of our protocol NAMO NAMAHA. Therefore an entire chapter is reserved for this protocol. Chapter 2 discusses this protocol in more detail and closely inspects the assumptions made by this protocol.

1.3 Motivation for our Work in the Background of Related Work

In the previous section we saw how multicast routing protocols fall into the four categories. We can also categorize the multicast protocols based on which layer they operate in. Multicast protocols can either operate at the networking layer or it can operate at the application layer as overlays. All of the protocols listed under the first three categories, Tree based, Mesh based and Hybrid operate in the network layer. The protocols listed under Stateless multicast category usually operates in the application layer. The application layer constructs what is known as overlay tree/mesh which forms the data distribution backbone. Network layer multicasting though efficient, incur more overhead in terms of control messages and update messages exchanged between nodes, owing to frequent topology changes. Overlay multicasting, usually at the application level do not incur much control message or update message overhead since they are stateless (that is they do not maintain state information for each multicast groups) but are not as efficient as network layer multicasting. With this and the previous section in mind, our protocol NAMO NAMAHA is motivated by the following arguments:

- AMRIS, MAODV, LAM, ODMRP, CAMP, FGMP and AMRoute, which operate at the network layer, cannot assume unbounded number of nodes either in the whole network or in any given multicast group due to the control/update message overhead for each change in the topology. Some of these protocols use flooding of messages for sender discovery and data forwarding. [SSB99a] shows by simulation that flooding probes, which causes repeated broadcasts, is highly unreliable because of the presence of hidden and exposed stations.
- Location Guided Tree Construction Algorithm and DDM, which operate at the

application layer, construct overlays. But this construction is not at all aided by the network layer which at any time is better informed about the network topology than the application layer.

- The Stateless multicast protocols discussed in the previous section are mainly for small groups and they absolutely do not scale to large groups as they make assumptions such as “all the group members should know the information of every other group members” and ”the addresses of all the group members should be included in the data packet”. Many of the network layer multicasting schemes like MAODV, ODMRP, etc. also do not scale to larger networks because of excessive control messages.
- Multicasting protocols like AMRoute assume the existence of a unicast routing protocol as an underlay. However it does not make any assumption about the particular kind of protocol. We believe that by tightly coupling the overlay multicasting protocol with a specific unicast routing protocol, better performance can be achieved. NAMO NAMAHA uses CEDAR as the underlying unicast routing protocol. As we can see in the later chapters, NAMO NAMAHA runs in tandem with, and over CEDAR, which allows us to construct optimized trees with linear time approximations. NAMO NAMAHA tries to get the best of overlay layer and network level multicasting approaches. Figure 1.1 makes this point clear.

1.4 Contributions of this Thesis

The thesis makes the following contributions:

- The idea that if the overlay construction algorithms in MANETs are supplied

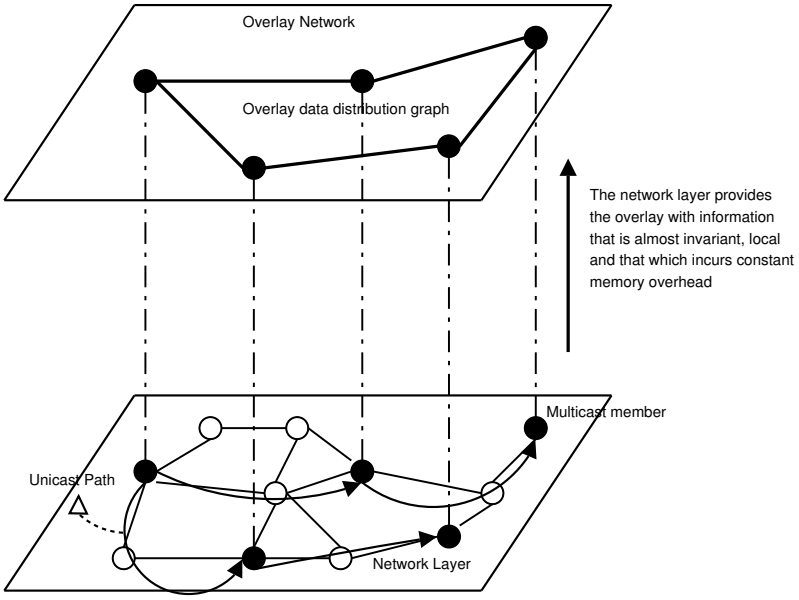

Figure 1.1: The basic idea of NAMO NAMAHA

with useful data from the network layer, that are almost invariant and local in nature, optimized data distribution graphs can be constructed. The network layer definitely has a 'better idea' of the underlying network and overlay algorithms incur lesser overhead on the network bandwidth. We are, in a way, trying to get the best of both worlds; network level multicasting and stateless overlay multicasting. The next section summarizes the features of the NAMO NAMAHA protocol.

- The protocol NAMO NAMAHA, and its algorithms that implement this idea along with proof of correctness and complexity analysis.
- Comparison with MCEDAR, in terms of cost of multicast data distribution trees, the number of messages exchanged and the time and memory complexity of the algorithms involved. We choose MCEDAR since other multicast protocols for ad hoc networks are either network based which do not scale for

large nubmer of nodes, or function as overlays designed only for small groups.

1.5 The Features of the NAMO NAMAHA protocol

In NAMO NAMAHA, we have achieved the following goals:

- NAMO NAMAHA scales to larger networks consisting of large multicast groups. NAMO NAMAHA assumes no bound on either the number of nodes in the network or the number of nodes in any given multicast group. This is made possible since the time and memory complexities of the algorithms and data structures are in the linear order of number of nodes or edges.
- NAMO NAMAHA builds close to optimal multicast data distribution trees. This is achieved by the incremental approximate Steiner tree construction algorithms.
- At any given point of time, a path exists between an arbitrary subscriber and the multicast sender with high probability. This is made possible by the redundancy offered by the $primary\ routers$ and the $secondary\ routers$ (as explained in Chapter 4)
- NAMO NAMAHA is not stateless. Maintaining and exchanging of tables exist in NAMO NAMAHA too. But in NAMO NAMAHA, these tables are exchanged only among local members. They need not propagate all over the network. Furthermore, only the delta information is exchanged. Since the information in the table is almost invariant, this delta information incurs trivial overhead in terms of size and bandwidth consumption.

1.6 Roadmap for this Thesis

The rest of the thesis is organized as follows. Chapter 2 discusses a small group multicasting scheme suitable for smaller number of nodes in the multicasting group. NAMO NAMAHA extends this scheme to large groups with no bounds on the number of nodes in the network or a given multicast group. Then we are encountered with a problem of choosing the right unicast routing protocol to function at the network layer. In Chapter 3 we discuss the CEDAR unicast routing protocol which is the chosen unicast routing protocol at the network layer. Justifications for this choice are made. This chapter also introduces MCEDAR, the multicast extension to CEDAR, since we compare this with our protocol. In Chapter 4 we present the NAMO NAMAHA protocol algorithms. In Chapter 5 we give the correctness proof, analyze the complexity and compare our protocol with MCEDAR. We conclude in Chapter 6 and present some future research directions.

Chapter 2

Small Group Multicast - The Basis for NAMO NAMAHA

2.1 Motivation for Stateless Multicast

Network layer multicasting protocols in ad hoc networks maintain multicast session state information. Because of the dynamic topology of ad hoc networks, this session state information keeps changing. The nodes will have to then exchange this volatile information and that would mean tremendous overhead on the network bandwidth. So especially in cases where the multicast groups are small, the recent shift is towards stateless overlay multicasting. Basically the overlay algorithm at the application layer queries the underlying unicast protocol to forward packets towards the members of the multicast group. Since the nodes calculate routes on the fly and do not maintain any session state information, the overlay algorithms scale to larger number of nodes. Of course there is a efficiency tradeoff involved here, but for small groups wherein the group sizes are in tens, overlay algorithms are as efficient as or more efficient than the network layer multicasting algorithms with the added benefit of lesser overhead. [CN02] proposes two tree construction heuristics for multicast

overlays using location information for small group. The heuristics rely on packet encapsulation. The algorithms target to reduce the overall bandwidth cost of the tree. Packet distribution tree is evaluated in a distributed fashion based on the list of destination nodes encapsulated in the IP packet. By distributed we mean that each node calculates only its outgoing branches to the next level subtrees and is not concerned with the construction of the whole tree.

2.2 Background

2.2.1 Problem Statement

If ad hoc network is modeled as an undirected graph, $G = (V, E)$, where V is the set of mobile nodes in the network and E is the set of wireless links between neighboring nodes, then the goal of the small group multicasting scheme is to form an undirected graph $G_G = (V_G, E_G)$ such that $V_G \subset V$ is the set of group nodes of a particular multicasting group and E_G is the set of edges which is actually a unicast route between some two nodes in V_G; the parent and the child node of the packet distribution tree. For instance an edge between node V_1 and V_2 is the unicast path computed by the underlying unicast protocol between nodes V_1 and V_2. The packet distribution tree is rooted at the sender node. Two tree construction greedy algorithms for small group multicast tree construction are proposed: Location-Guided K-ary Tree Construction algorithm (LGK) and Location-Guided Steiner Tree Construction algorithm. Before detailing the algorithms, we discuss the assumptions made by this scheme and also the approximation relating geometric distance of a node and the number of network hops between two nodes on a unicast route. The scheme also includes methods for membership information update, route caching and location update

mechanism. We do not discuss these schemes here as they are not related to our work.

2.2.2 Assumptions

The following assumptions are made by this scheme

- The number of nodes in a multicast group is in the order of tens so that including the node address of all the members of such a multicast group in the IP packet would not incur much overhead. The scheme requires that the sender node include the IP address of each node that it knows is in the multicast group to which it is sending data.
- Every member in the multicast group knows about the presence of other members in the same multicast group. Since the number of nodes involved in each multicast group is small, this assumption is justifiable. The members of the multicasting group are aware of the IP addresses of every other member in that group.
- The underlying unicast routing protocol is able to forward packets from source to destination along or close to the shortest path. The overlay algorithms assume the existence of a unicast routing protocol at the network layer.
- Each node in any given multicast group knows almost up to date location information about other nodes in its multicast group. Each node has some mechanism, like GPS, to know its own position with respect to some reference point.
- This work assumes that longer geometric distance requires more number of network hops. In reality this is generally true though there could be some situations on the contrary. Through simulation, [CN02] shows that, on an average, the number of network hops increases monotonically as the geometric distance increases. Therefore geometric distance is a fair enough approximation to the

number of network hops.

2.3 Location-Guided K-ary (LGK) Tree Construction Algorithm

This is a greedy heuristic which tries to minimize the overall cost of the tree at each step in the algorithm. The sender first includes the IP addresses of all the receivers of its multicast group in the header of the data packet. The sender then forwards a copy of the packet to the nearest k nodes. Since each of the node knows the location information of every other node in its multicast group, the sender can determine which k nodes are nearest to the sender. Once this is done the sender distributes the remaining nodes in the multicast group apart from its k children, to each of its k children based on geometric proximity. After this point, the algorithm is repeated recursively at each of its k children. Each child will forward the data packet to k more nodes that are closest to it. At each node the first step is 'children selection' and the second step is 'subtree clustering'. The recursion stops when an incoming packet at some node has an empty destination list. This heuristic uses the approximation relating geometric proximity and number of network hops

In the following example taken from [CN02], binary tree (k=2) is constructed. Figure 2.1 shows the working of the LGK algorithm. Only the participating nodes n_0, n_1, . . . , n_9 are shown in the figure. The tree is rooted at n_0 since that is the sender. Node n_0 has $<$ n_1, n_2, n_3, . . . , $n_9 >$ as its destination list. Based on the list, the LGK tree algorithm runs in two consecutive steps: 1) children selection and 2) subtree clustering. In the first step, the algorithm selects two geometrically nearest nodes as the source node's children. In the example, node n_3 and n_5 are selected because they are the nearest two nodes to n_0. In case of equal distance, the tie is broken by a random selection of the nodes. In the second step,

the algorithm goes through the rest of the destinations: if a destination is geometrically closer to n_3, it is put into a sub-list designated as the destination list for n_3; otherwise, it is put into the destination list for n_5. This ensures that later on, node n_3 and n_5 will find their destinations close to themselves. In the example, the list for n_3 is $< n_1, n_7, n_8 >$, and the list for n_5 is $< n_2, n_4, n_6, n_9 >$. When a destination has equal distances to both n_3 and n_5, it is taken by the node with a shorter destination list to achieve better tree balancing. The algorithm is repeated recursively at n_3 and n_5.

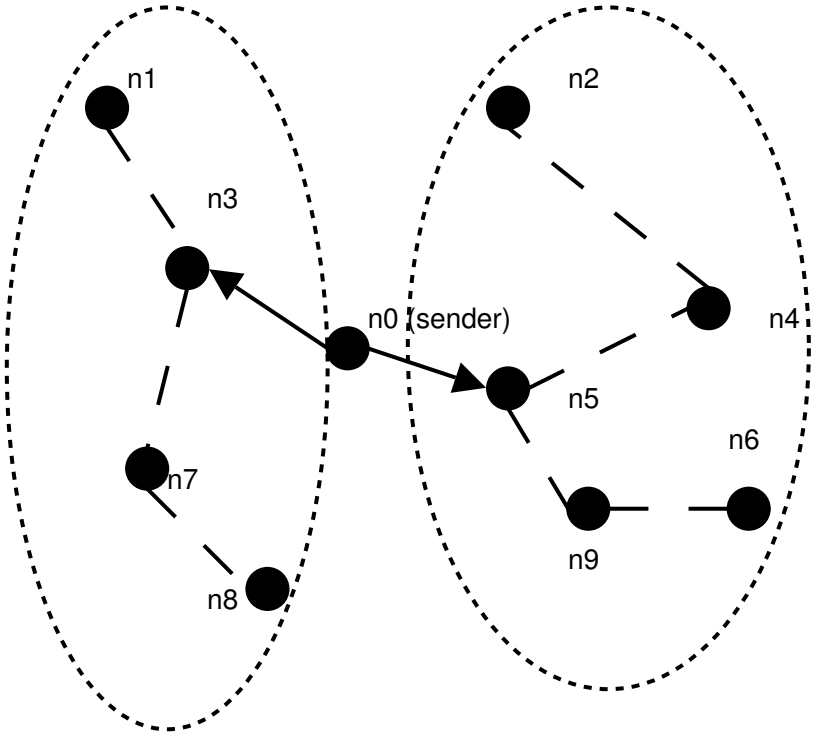

Figure 2.1: The *children selection* and *subtree clustering* in Location Guided K-ary (LGK) tree construction algorithm

In the case when a packet cannot be forwarded to a children node (no route), the packet is lost and the destination nodes in that subtree will not receive the packet. This situation may happen in MANET because mobile nodes can shut itself down abruptly or move out of

reach. Therefore, the sender should include only the active nodes as the packet's destination nodes. This requires that each node periodically refreshes the membership of itself to the rest of the group. A location update message not only updates a node's new location, but also refreshes its group membership. If an update message has not been received over a timeout period from a node, the node is purged from the destination list of other nodes. Because of the periodic membership refreshment, the destination nodes are very likely to be reachable from the source node.

To summarize, LGK tree construction algorithm produces a k-ary tree rooted at the sender with the group nodes as tree nodes. Packets are forwarded node-by-node along the k-ary tree from the source to the rest of the multicast group via unicast routing. This packet forwarding process is guaranteed loop-less because a destination address will be taken out of the list whenever the packet has reached the destination, therefore, it cannot go back to that node again.

2.4 Location Guided Steiner (LGS) Tree Construction Algorithm

The Steiner tree is commonly used as a multicast packet distribution tree for efficient delivery of multicast packets in a fixed network. It spans over all nodes in a multicast group and minimizes the overall cost of the tree. Finding a Steiner tree in network is a NP-hard optimization problem [HRW92]. Under the well-known Takahashi-Matsuyama heuristic [TM80], the multicast routing protocol generates a Steiner tree by an incremental approach. Initially the tree contains only the source node. At each iteration, the nearest unconnected destination to the partially constructed tree is found and the least-hop path between them is added to the tree. The distance is usually measured by the number of network-level

hops. This tree construction process is repeated until all destinations are included in the tree. In a router-assisted multicasting approach, every node in a network can become a tree node to forward packets, in which case the constructed Steiner tree is near optimal. The location-guided Steiner (LGS) tree in [CN02] is constructed using a modified version of the Takahashi-Matsuyama heuristic. The differences are: 1) geometric distance is used as a measurement of closeness; 2) only the group nodes can be used as tree nodes. Between the group nodes, data packets are encapsulated in unicast packets and forwarded via the underlying unicast routing protocol. Below, the same set of nodes used in the earlier example to illustrate LGK algorithm are used to illustrate the construction of a LGS tree, as shown in Figure 2.2. Again this example is taken from [CN02]. Initially, the tree only contains the sender node n_0. Within the remaining set of nodes $\{n_1, n_2, \ldots, n_9\}$, node n_3 is geometrically closest to n_0. Therefore, n_3 is added into the tree with edge n_0n_3. In the second step, the remaining set of unconnected nodes are examined and the node closest to the partially constructed tree is selected. In the example, we compare the distance from n_0 to each of the nodes in the un-connected set $\{n_1, n_2, n_4, \ldots, n_9\}$, as well as the distance from n_3 to that set, and select the shortest distance which is between n_0 and n_5. Therefore, n_5 is added to the tree with edge n_0n_5. This process repeats until all the nodes have been included in the tree as shown in the figure. Subsequently, the sender node n_0 forwards a copy of the data packet to each of its children nodes, i.e. n_2, n_3, and n_5, with their corresponding subtrees as destinations. Similar to the LGK tree construction process, the children nodes should be reachable most of the times as result of the periodic membership refreshments. At each of the children nodes, a LGS tree is computed again to further forward the packet. This forwarding process repeats until the packet has reached all members of the group.

Although the entire LGS tree can be computed at the source and included in the header of the data packet, the tree is constructed hop-by-hop because: 1) it allows the intermediate

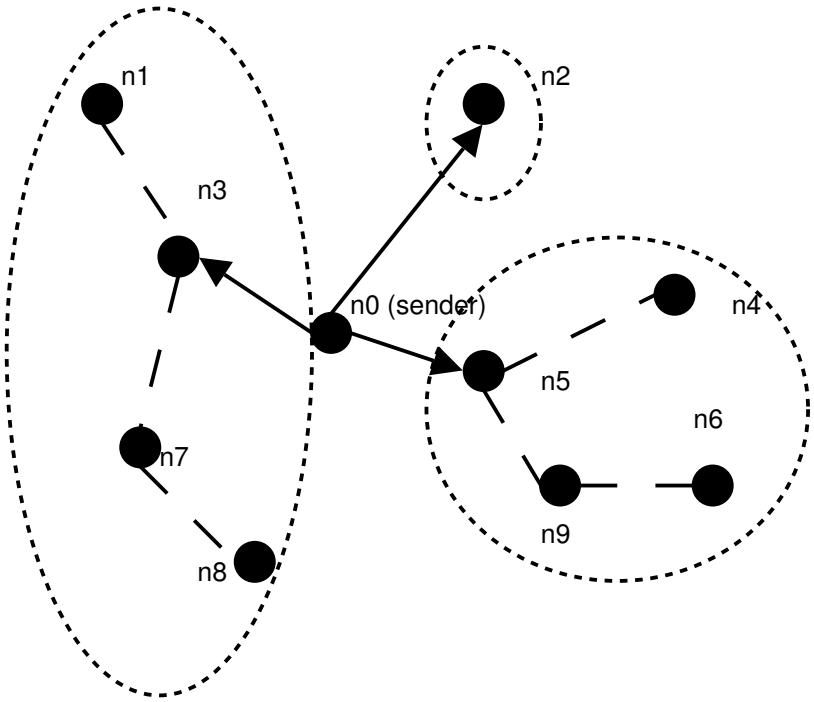

Figure 2.2: The Location Guided Steiner (LGS) tree construction algorithm

nodes to utilize the latest location information of the destination nodes in computing the tree; 2) by caching a previously computed tree, the computation will not be duplicated when the locations of the nodes have not changed between data packets. The major difference between LGK and LGS is that the outgoing degree of a tree node in LGS is not fixed. It depends on the outcome of the constructed tree. In LGK, the out-going degree of a tree node is fixed at k.

2.5 Extending the Small Group Multicast Scheme to Larger Groups: The Basic Idea for NAMO NAMAHA

The Small group multicast scheme performs very well for multicast groups having nodes in order of tens. But when it comes to large groups with number of nodes in the order of hundreds or thousands, the same scheme cannot be used; it calls for a whole new approach. There are some assumptions that [CN02] makes which are not feasible for large group multicasting. In this thesis, we attempted to extend this scheme to large group multicasting and came up with the whole new concept in NAMO NAMAHA.

- The small group multicast scheme assumes that the sender knows the group members and its identification in advance. This allows the sender to encapsulate the IP address of all the senders in the packet it sends out to the multicast group. This does not scale if there are large number of subscribers to a multicast group. The sender cannot afford to include the IP addresses of all the receivers when it sends out the data. This assumption has to be removed.
- There is no explicit join and leave protocol in [CN02]. The group membership is static and known to the sender in advance. When the group membership is dynamic, we need to have some join and leave protocol. In this thesis, we outline the join and leave protocol that is scalable with the number of network nodes. The join protocol in NAMO NAMAHA will have non-propagating sender discovery unlike that of MCEDAR.
- The scheme assumes that each node knows the existence and information of other nodes in the multicast group. For large number of nodes, this does not scale. If the information about all nodes are stored at every single node then any change in information of any node should be propagated to all the nodes.

For groups in order or tens this is trivial. But this means tremendous overhead for larger groups. This assumption has to be removed.

- [CN02] assumes that the location information about each node is known to all other nodes. The scheme adopts a hybrid location update scheme which attempts to keep the location information of all the participating members at any node up to date. The hybrid location update scheme performs poorly for large groups. Once again, this assumption has to be removed. In NAMO NAMAHA we require that the nodes know its location and the sender's location. But we do not need a location update mechanism since there is no requirement for the nodes to know each other's location.

This work [CN02] shows that if the location information is up to date, LGS algorithm outperforms the LGK algorithm. NAMO NAMAHA uses the number of network hops, delay or any other link cost information of the paths to the nodes in the region up to the third neighborhood directly. Since this information is up to date among the nodes that participate in the multicast tree construction in our protocol, NAMO NAMAHA constructs approximate network hops or delay based Steiner trees using local information. We use this fact from [CN02]. However the main contribution of this thesis comes from the realization that the overlay construction algorithms benefit a lot from the network layer assistance. This particular aspect is missing from other works like [CN02], [JC01], etc. The core nodes as evaluated by a specific unicast routing protocol at the network layer are the ones that provide this useful information. The details will be given in the next two chapters.

Chapter 3

CEDAR: QoS Routing Protocol for Ad Hoc Networks - The Unicast Underlay for NAMO NAMAHA

3.1 Logic for having a Specific Unicast Routing Protocol (CEDAR) as an underlay for NAMO NAMAHA

Overlay algorithms assume the existence of unicast routing protocol functioning at the network layer. The issue we have now is the selection of a unicast protocol at the network layer that could provide useful information to the overlay construction algorithms at the application layer. We argue that having a specific protocol for this purpose is better than having any routing protocol at the network layer. There are many overlay schemes that make no assumptions about the unicast routing protocol at the network layer like [CN02], [JC01], [LTM99], etc. The overlay algorithms and the functioning of the unicast protocol should be closely knit and the unicast routing protocol should be able to give some useful information to the application layer in constructing efficient overlays. Furthermore, the

useful information should be gleaned from local topology knowledge and should be almost invariant. The performance of multicasting overlays, on a large degree, is dependent on the performance of the underlying unicast routing protocol. The search for such a unicast routing protocol led to CEDAR [SSB99a]: Core Extracted Distributed Ad Hoc Routing Algorithm, which is detailed in this chapter along with the justification for the choice.

3.2 Introduction to CEDAR

CEDAR, which stands for Core-Extraction Distributed Ad hoc Routing algorithm, is mainly a Quality of Service routing protocol for ad hoc networks. Here route computation is on demand using only the local state kept track by a set of nodes called the core nodes. There are three components to CEDAR [SSB99a]

- Core Extraction: A set of nodes is dynamically elected in a distributed fashion to form the core of the network by approximating a minimum dominating set (MDS) of the ad hoc network using only local computation and local state. Each core node maintains the local topology of the nodes in its domain and also performs the route computation on behalf of these nodes. The members of the core (approximate MDS) are known as 'core nodes'. The region up to the third neighborhood of a given core node is known as its 'domain'. By the property of MDS, each node is either in the MDS or is a first neighbor to a core node in the MDS. For a node that is not in the core, some core node that is its first neighbor will act as its 'dominator'. This choice is made by the Core Computation algorithm which is detailed later.
- Link State Propagation: QoS routing in CEDAR is achieved by propagating the bandwidth availability information of stable links in the core known to nodes

far away in the network, while information about dynamic links or low bandwidth links is kept local. Slow moving 'increase waves' and fast moving 'decrease waves', which denote corresponding changes in available bandwidths on links, are used to propagate non-local information over core nodes. In our thesis, we do not focus on this module of the CEDAR protocol.

- Route Computation: Route computation first establishes a core-path from the dominator of the source to the dominator of the destination. The core path provides the directionality of the route from the source to the destination. Using this directional information, CEDAR iteratively tries to find a partial route from the source to the domain of the furthest possible node in the core path satisfying the requested bandwidth, using only local information. This furthest possible core node becomes then becomes the source of next iteration, that is, the route using the core path as a directional aid now has to be calculated from this 'intermediate' destination to the actual destination.

In this thesis we are interested in the Core Extraction part of the CEDAR protocol. After briefing the network model and terminologies used in CEDAR, we detail the Core Extraction procedure of this protocol.

3.3 Network Model and Graph Terminology for CEDAR and NAMO NAMAHA

CEDAR and our work assumes that all the nodes use the same shared wireless channel and that the neighborhood is a commutative property. That is if node A is a neighbor of node B then B is a neighbor of A. The existence of CSMA/CA type algorithm like MACAW [BDSZ94] is assumed for reliable unicast communication and for solving the hidden and

exposed terminal problem. The data transmission is preceded by control packet handoff and the sequence of packets exchanged in a communication is the following: RTS (Request to Send from the sender to receiver) - CTS (Clear to Send from receiver to Sender) - Data (from sender to receiver) - Ack (from receiver to sender)

Ad hoc network is represented by means of an undirected graph $G = (V, E)$ where V is the set of nodes in the graph (host in the network) and E is the set of edges in the graph (links in the network). The i^{th} $deleted\ neighborhood$, $N_i'(x)$ of node x is the set of nodes whose distance from x is not greater than i except node x itself. The i^{th} neighborhood $N_i(x)$ of node x is $N_i'(x) \cup \{x\}$

A dominating set $S \subset V$ is a set such that every node in V is either in S or is a neighbor of a node in S. A dominating set with minimum cardinality is called a minimum dominating set (MDS). A $virtual\ link$ $[u, v]$ between two nodes in the dominating set S is a path in G from u to v.

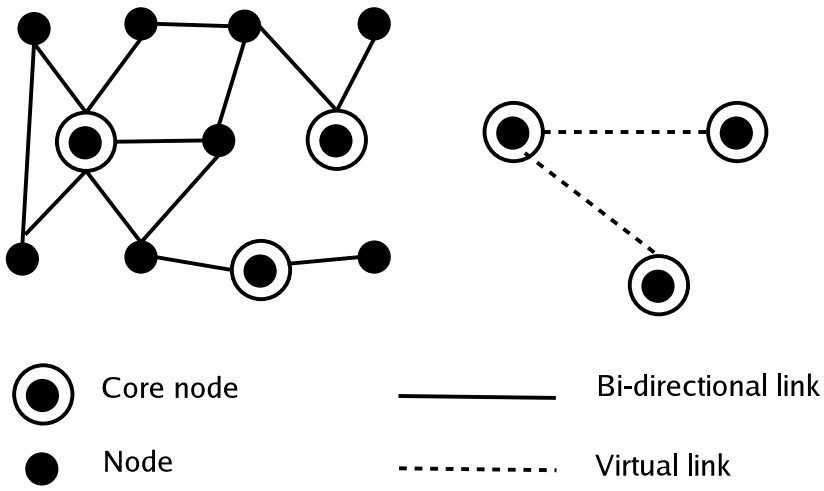

Figure 3.1: Core nodes and the Approximate Minimum Dominating Set (MDS)

Given an MDS V_C of a graph G, we define a core of the graph $C = (V_C, E_C)$, where $E_C = \{[u, v] | u \in V_C, v \in V_C, u \in N_3'(v)\}$. Thus the core graph consists of the MDS nodes V_C

and a set of virtual links between every two nodes in V_C that are within a distance 3 of each other in G. Two nodes u and v which have a virtual link $[u, v]$ in the core are said to be the $nearby$ nodes

In the CEDAR algorithm, each node picks up a node in $N_1(u)$ as its dominator (based on the criteria discussed in the next section), denoted as $dom(u)$. $dom(u)$ is the node which is then called a core node.

3.4 Generation and Maintenance of the Core in CEDAR

The generation and maintenance of core is of primary interest to us and its computation as detailed in [SSB99a] is given here. Consider a node u with first deleted neighborhood $N_1'(u)$, degree $d(u) = |N_1'(u)|$, dominator $dom(u)$, and effective degree $d^*(u)$, where $d^*(u)$ is the number of its neighbors who have chosen u as their dominator. The core computation algorithm works as follows at node u.

1. Periodically, u broadcasts a beacon which contains the following information pertaining to the core computation: $(u, d^*(u), d(u), dom(u))$

2. If u does not have a dominator, then it sets $dom(u) \leftarrow v$, where v is the node in $N_1(u)$ with the largest value for $(d^*(v), d(v))$, in lexicographic order. Note that u may choose itself as the dominator.

3. u then sends v a unicast message including the following information: $(u, \{(w, dom(w)) | \forall w \in N_1'(u)\})$. v then increments $d^*(v)$

4. If $d^*(u) > 0$, then u joins the core.

Essentially, each node that needs to find a dominator selects the highest degree node with the maximum effective degree in its first neighborhood. Ties are broken by node

id. The above algorithm for core computation results in a core which has the following properties.

- Since the core computation algorithm approximates the minimum dominating set for the nodes, the size of the core is minimal. As the route computation is done by the core nodes, minimizing the number of core nodes is desirable.
- Core computation is local. This property makes core computation in CEDAR scalable as the core can be computed in a constant amount of time.
- When a node is electing a dominator, it gives preference to core nodes already present in its neighborhood (including itself). This provides stability to the core computation algorithm, though it might have implications on the optimality of the number of core nodes.

When a node u joins the core, it issues a piggybacked broadcast in $N_3'(u)$. A piggybacked broadcast is accomplished as follows. In its beacon, u transmits a message: $(u, DOM, 3, path\_traversed \leftarrow null)$. DOM denotes the id of $u's$ dominator. When node w hears a beacon that contains a message $(u, DOM, i, path\_traversed)$, it piggybacks the message $(u, DOM, i-1, path\_traversed + w)$ in its own beacon if $i - 1 > 0$. Thus, the piggybacked broadcast of a core node advertises its presence in its third neighborhood. This guarantees that each core node identifies its nearby core nodes, and can set up virtual links to these nodes using the $path\_traversed$ field in the broadcast messages. The state that is contained in a core node u is the following: its nearby core nodes (i.e. the core nodes in $N_3'(u)$); $N_*'(u)$, the nodes that it dominates; for each node $v \in N_*'(u)$, $(\forall w \in N_1'(v), (w, dom(w)))$. Thus each core node has enough local topology information to reach the domain of its nearby nodes and set up virtual links. However, no core node has knowledge of the core graph. In particular, no non-local state needs to be maintained by core nodes for the construction or maintenance of the core. Maintaining the core in

the presence of network dynamics is simple. Consider that due to mobility, a node loses connectivity with its dominator. After listening to beacons from its neighbors, the node either finds a core neighbor which it now nominates as its dominator, or nominates one of its neighbors to join the core, or itself joins the core. If a node loses connectivity with all its dominated nodes, or discovers (by monitoring the beacons of its dominated nodes) that its effective degree has become 0, it leaves the core by tearing down virtual links with its neighbors, and finds a dominator in the core.

3.5 Core Broadcasting Mechanism

As with most existing ad hoc networking protocols, CEDAR requires the broadcast of route probes to discover the location of a destination node, and the broadcast of some topology information (in the form of increase/decrease waves). While most current algorithms assume that flooding in ad hoc networks works reasonably well, the experience of the authors of CEDAR has shown otherwise. In particular, they have observed that flooding probes, which causes repeated local broadcasts, is highly unreliable because of the presence of hidden and exposed stations. Thus, they provide a mechanism for 'core broadcast' based on reliable unicast (using RTS-CTS etc.). Note that it is reasonable to assume a unicast based mechanism to achieve broadcast in the core, because each core node is expected to have few nearby core nodes. Besides, the core broadcast mechanism ensures that each core node does not transmit a broadcast packet to every nearby core node. CEDAR uses a close coordination between the medium access layer and the routing layer in order to achieve efficient core broadcast. The goal is to use the MAC state in order to achieve efficient core broadcast using $O(|V|)$ messages, where $|V|$ is the number of nodes in the network.

In order to achieve efficient core broadcast, the scheme assumes that each node temporarily caches every RTS and CTS packet that it hears on the channel for core broadcast packets only. The purpose of caching RTS/CTS is to use them for the elimination of duplicate packet reception for broadcasts. Since RTS/CTS packets are much smaller compared to the data packets and the core broadcasts would typically arrive from the neighbors in a small period of time, the caching of RTS/CTS packets (only for core broadcasts) for a few seconds is justified. Each core broadcast message M that is transmitted to a core node has the unique tag (M, i). This tag is put in the RTS and CTS packets of the core broadcast packet, and is cached for a short period of time by any node that receives (or overhears) these packets on the channel. Consider that a core node u has heard a $CTS(M, v)$ on the channel. Then, it estimates that its nearby node v has received M, and does not forward M to node v. Essentially, the idea is to monitor the RTS and CTS packets in the channel in order to discover when the intended receiver of a core broadcast packet has already received the packet from another node, and suppress the duplicate transmission of this packet.

3.6 Advantages of Choosing CEDAR as an Underlay for NAMO NAMAHA

As it becomes evident in the next chapter, when NAMO NAMAHA uses CEDAR, the following advantages are identified for choosing CEDAR as an underlay unicast routing protocol:

- CEDAR uses only local computations and the time complexity of all the algorithms involved in CEDAR are linear with number of nodes in the network or number of edges in the network.
- Since CEDAR only maintains the state of the local nodes, precisely of all the

core nodes in the third neighborhood, the memory complexity is also constant. We assume a constant bound on the number of neighbors a node can have. See Chapter 5 for complexity analysis.

- Core broadcasting has $O(|V|)$ message complexity.
- The core nodes can provide useful information about the local topology for the application layer to construct efficient overlays. The useful information includes the local link characteristics like delay information, number of network hops local and local topology graph which would aid NAMO NAMAHA to incrementally build the Steiner tree over a subgraph of the core graph which is actually an approximate minimum dominating set.
- NAMO NAMAHA uses the CEDAR unicast routing algorithm in its $Unicast\ Trap$ algorithm. In NAMO NAMAHA, we use the fact that the unicast path between the subscriber and the sender is approximately the best path for the sender to route its multicast data packets to the subscriber.

3.7 MCEDAR: Multicasting Extension to CEDAR

MCEDAR [SSB99b] is a multicast routing protocol built on top of the CEDAR protocol. In Chapter 5, we compare MCEDAR with NAMO NAMAHA. Therefore this section briefly introduces the MCEDAR protocol. MCEDAR uses two of CEDAR's components, mamely: the core and the core broadcast. The infrastructure for a multicast group resides entirely within the core and the core broadcast mechanism is used to perform data forwarding on the infrastructure. Briefly, for each multicast group MCEDAR extracts a sub-graph of the core graph to function as the routing infrastructure. The subgraph is a mesh structure and

is called the $mgraph$ for the multicast group. Once the $mgraph$ is extracted for a multicast group, data forwarding is done on the $mgraph$ using the core broadcast mechanism. MCEDAR thus has four key components: (i) the $mgraph$ which is the multicast routing infrastructure, (ii) the join protocol, (iii) the core broadcast based on forwarding protocol and (iv) the leaving, pruning and reconstruction protocols. The rest of this section describes in more detail each of these components.

3.7.1 The $mgraph$ Infrastructure

MCEDAR uses a mesh structure called the $mgraph$ as its multicast routing infrastructure. The inherent redundancy present in meshes increase the robustness of the $mgraph$. Hence recomputation of the $mgraph$ may not be necessary for every link breakage. This is a property that is critical to ad hoc environments where link breakages occur often due to node mobility. However unlike other mesh based approaches [], MCEDAR minimizes the number of nodes in the $mgraph$ by requiring only the core nodes to become the member of $mgraph$. Specifically, an $mgraph$ is a subgraph of the core graph and not a subgraph of the underlying network. Thus, only core nodes can become members of an $mgraph$. When a node wants to become a member of multicast group, it requests its dominating core node to join the appropriate $mgraph$ and the dominator then performs the join operation. MCEDAR does not differentiate between sending and receiving members of a multicast group. Hence all aspects of MCEDAR algorithm are same both for senders and receivers of a multicast group.

3.7.2 The Join Protocol

Since the $mgraph$ consists of only core nodes, only a core node is allowed to perform a join operation in MCEDAR. When a non-core node wants to become a member of a multicast group it requests its dominating core node to perform the join operation. A core node performs the join operation by core broadcasting a $join\ request\ JOIN(MA, joinID)$. The join request consists of the address of the group the node wishes to join and the current joinID of the node, corresponding to the multicast group. The joinID of a freshly joining node is set to infinity. When a node that is not a member of MA receives the join request, it forwards the message to its nearby core nodes in accordance with the core broadcast mechanism. On the other hand, when an existing member of MA receives the join request, it sends a $JOIN\_ACK(MA, joinID)$ only if its joinID is lesser than the joinID that arrives in the request. It then forwards the JOIN request further down. However if its joinID is greater than the incoming joinID, it forwards the request like a non-member. Ties in joinIDs are broken based on nodeIDs.

The joinID in the JOIN_ACK message back to the node requesting the join is that of the replying node. When an intermediate node on the reverse path (from the replier to the requester) receives the JOIN_ACK message, it decides on whether to accept the JOIN_ACK or reject it based on the number of JOIN_ACKs it has already accepted for the particular multicast group. The member, on accepting a JOIN_ACK, sets its joinID to the maximum of its current joinID and the arriving joinID incremented by one. It then stamps the joinID of the JOIN_ACK with its new joinID. Each $mgraph$ member maintains two other data structures, the $parent$ set and the $child$ set. When a node accepts a JOIN_ACK, it adds the upstream $mgraph$ member to its parent set. Further, if the downstream node (as specified in the JOIN_ACK) is not already in its child set, it forwards the JOIN_ACK to the downstream node and adds the downstream node to its child set. On other hand, when the intermediate

node decides to reject a JOIN_ACK, it suppresses the JOIN_ACK and does an explicit leave from the upstream node so that its ID is removed from the upstream node's child set.

3.7.3 The Forwarding Protocol

Although the $mgraph$ for a multicast group is a mesh infrastructure, the forwarding of data on the infrastructure is done only on a source based tree, thus saving on redundant transmissions leading to the efficient usage of the scarce bandwidth. The forwarding protocol of MCEDAR uses the core broadcast mechanism. Specifically, when a data packet arrives at an $mgraph$ member, the member attempts to forward the data packet only to those nearby core nodes that it knows are also member of the same $mgraph$. Further, some of these attempts are suppressed by core broadcast mechanism if it is found that a downstream member has already received the same data through a different path. Such a forwarding protocol has two key advantages: (i) it eliminates redundant transmissions and hence saves bandwidth usage and (ii) it implicitly creates a source based tree that represents the fastest delivery structure, for each data packet of the multicast group.

3.7.4 The Leaving and Pruning Protocol

A member of the $mgraph$ issues a $leave$ message to each of its parents when it does not have any local members in its domain and its child list is empty. Since all member of the $mgraph$ perform the leave operation if the two conditions are satisfied, the mesh is automatically pruned. A parent that receives the leave message from one of its children deletes the corresponding child's ID from its child set. When a node loses connectivity with all of its parents, then there is a potential partition of the $mgraph$. In such an event, the node issues a join request with the joinID set to its current joinID (as opposed to infinity for a fresh

join). Since only members that have lesser joinIDs respond to this request, the formation of partitions in the $mgraph$ when the underlying graph is connected is eliminated.

3.7.5 Shortcomings of MCEDAR Answered in NAMO NAMAHA

We leave the detailed comparison between NAMO NAMAHA and MCEDAR to Chapter 5. However before introducing NAMO NAMAHA, we briefly list the issues in MCEDAR addressed by NAMO NAMAHA. We start with the join protocol. The join protocol is costly and complicated in MCEDAR and the sender discovery messages propagate all over the network. NAMO NAMAHA has a very simple join protocol without acknowledgements; the sender discovery messages are restricted to a very small portion of the network. The forwarding protocol in MCEDAR relies completely on core broadcasting. Because of this, many of the messages that are core broadcast are just discarded thus wasting some precious bandwidth. NAMO NAMAHA ensures that the multicast messages are propagated only to the regions, where it is absolutely necessary to be sent. MCEDAR uses the core graph which is a subgraph of the underlying network graph. NAMO NAMAHA uses the graph comprising of sender, subscriber, primary routers and secondary routers which altogether form a subgraph of the core graph. Furthermore, NAMO NAMAHA minimizes the delay or number of network hops by constructing approximate Steiner trees in the third neighborhood region of the core nodes. MCEDAR does not minimize the total cost of the multicast data distribution tree.

Chapter 4

Algorithms for the NAMO NAMAHA Protocol

4.1 Introduction

We start with a brief overview of the NAMO NAMAHA protocol. The NAMO NAMAHA protocol has three main components: the $join$ protocol, the $forwarding$ protocol and the $leave$ protocol. When a node wants to subscribe to a multicast group, the dominator of that node would send a unicast message to the multicast group sender using the CEDAR unicasting capability. Approximately this path would be used to route the multicast data packets from the sender to the subscriber. The core broadcast messages of this unicast is restricted to a small zone called the 'request zone', which is calculated based on the subscriber's location and the sender's location. Periodic unicast messages need to be sent from the subscriber to the sender because the unicast path soon becomes outdated owing to the node mobility. The forwarding protocol is basically the distributed implementation of Takahashi Matsuyama heuristic for constructing Steiner trees. To begin with, Steiner tree construction starts at the sender node and is built in the region up to the third neighborhood of the sender node. New Steiner tree construction starts at the periphery of the third neighborhood region of the sender node, and the construction is always limited up to the third

neighborhood region of the core node that initiated the new Steiner tree construction. The process continues until all the subscribers are included in the multicast data distribution tree. We detail each of the three components of the NAMO NAMAHA protocol, along with the tables and timers used, after looking at the assumptions made by this protocol. See Figure 4.1

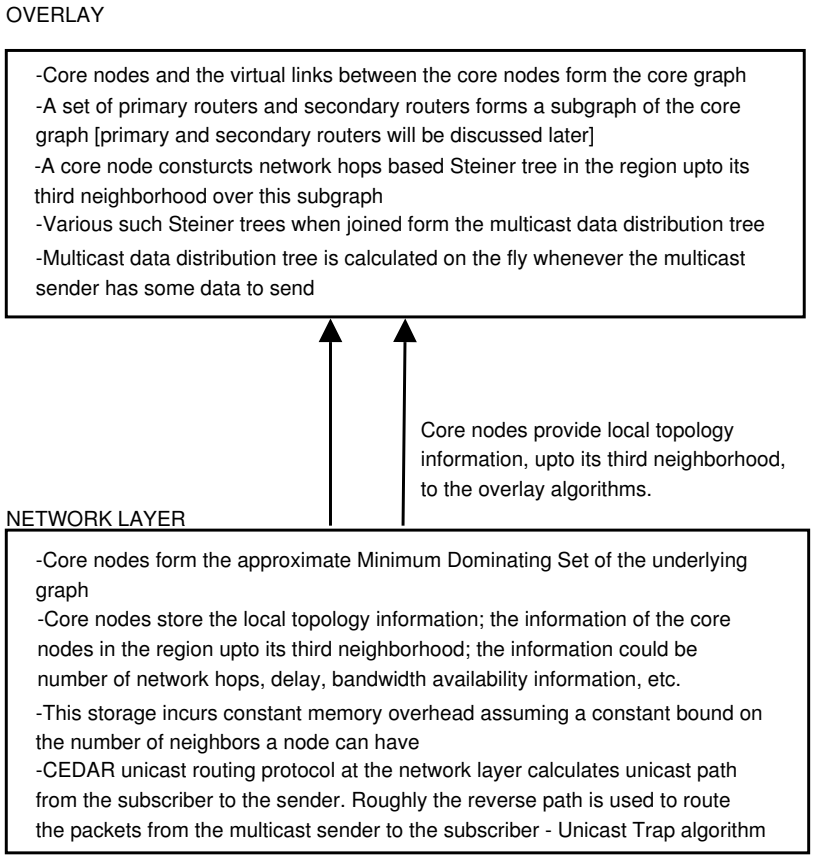

Figure 4.1: Activities at the Network and the Overlay layers and the information flow between them

4.2 Assumptions

NAMO NAMAHA makes the following assumptions:

- The link characteristics are symmetric; it is same when measured from node A to node B, if node A and node B are neighbors; and neighborhood is a commutative property; that is if node A is the neighbor of node B, then node B is a neighbor of node A.
- Each node knows its location with respect to some coordinate using mechanism such as GPS.
- Each multicast subscriber knows the address of the sender for the group which it is subscribing to. This assumption is needed for our unique $Unicast\ Trap$ algorithm. We may assume that the sender addresses are published along with the multicast addresses.

4.3 Join Protocol Using the Unicast Trap Algorithm

The join protocol is implemented using our unique 'Unicast Trap' algorithm. To be precise, if a node wants to join a multicast group, it just informs its dominator that it needs to join that group and the dominator sends a unicast message to the sender of that multicast group. Approximately the same path is used to route the data packets from the multicast group sender to the subscriber. NAMO NAMAHA also requires that the dominator send a unicast message to the sender of the multicast group periodically as the unicast path may soon become outdated due to node mobility. The core broadcast associated with this unicast is restricted to a small zone between the subscriber and the sender called the 'request zone'. The join protocol is unlike the join of many other MANET multicasting protocol which rely on some kind of flooding or broadcasting. We detail the join protocol that uses the Unicast Trap algorithm in the following sub sections. First we detail on the CEDAR QoS routing, as it is used as the underlying unicast protocol. After this, we introduce the concept

of primary and secondary routers. We then deal with the request zone calcuation and the 'trapping' process. All these components collectively form the Unicast Trap algorithm which implement the join protocol in NAMO NAMAHA.

4.3.1 CEDAR QoS Route Computation

Briefly, QoS route computation in CEDAR is on-demand routing algorithm which proceeds as follows: when a source node s seeks to establish a connection to a destination node d, s provides its dominator node $dom(s)$ with a (s, d, b) tuple, where b is the required bandwidth for the connection. If $dom(s)$ can compute an admissible available route to d using its local state, it responds to s immediately. Otherwise if $dom(s)$ already has the dominator of d cached and has a core path established to $dom(d)$, it proceeds with the QoS route establishment phase. If $dom(s)$ does not know the location of d, it first discovers $dom(d)$, simultaneously establishes a core path to d, and then initiates the route computation phase. A core path from s to d results in a path in the core graph from $dom(s)$ to $dom(d)$. $dom(s)$ then tries to find the shortest widest furthest admissible path along the core path. Based on its local information $dom(s)$ picks up the farthest reachable domain up to which it knows an admissible path. It then computes the shortest-widest path to that domain, ending at a node say t, once again based on local information. Once the path from s to t is established, $dom(t)$ then uses its local state to find the shortest-widest furthest admissible path to d along the core path, and so on. Eventually, either an admissible route to d is established, or the algorithm reports failure to find an admissible path. For our purpose we may 'turn off' the QoS route computation part of the CEDAR unicast protocol.

4.3.2 The Primary and the Secondary Routers

The dominator of the node that joins the multicast group is marked as a primary router for that multicast group. The unicast path between the subscriber's dominator and the sender's dominator is a core path and each core node in this path flags itself as a primary router for the multicast group to which the subscriber just subscribed. All the core nodes in the third neighborhood region of a primary router flag themselves as a secondary router for that multicasting group. A core node becomes a secondary router on receiving the periodic beacon message from some primary router in its third neighborhood.

4.3.3 The Request Zone Selection and Multicast Sender Discovery

Every time a node joins a multicast group, it informs its dominator and the dominator sends a unicast message to the sender informing the sender that the node wants to join the multicast group. The CEDAR Unicast QoS route computation however employs core broadcast and the multicast sender discovery message may propagate all over the network. We borrow the concept of Request Zone calculation from LAR [KV98]. We assume that the sender sends its location, direction and velocity information in all the multicast data packets. So the subscriber can predict the location of the sender approximately using this information. Request zone is a subset of the core graph, in which the route discovery messages are broadcasted. The probability of finding the unicast path in the request zone is very high. If the unicast route is not found, we may adopt the incremental Request Zone calculation as used in [RAMM03], the modified version of LAR. Each time the sender sends out the data, it also includes its location, direction and velocity information. NAMO NAMAHA requires that each node know its location using some mechanism such as GPS. Whenever a unicast message is sent, the core broadcasting of the core path finding message

is restricted to the request zone. The core broadcast messages used in unicasting will have the location of the subscriber and the sender in it. The core nodes receiving these core broadcasting messages will further core broadcast the message only if it finds that it is in the request zone based on the location of the sender and the subscriber. Else it will discard the messages and will not further core broadcast them. Figure 4.2 depicts the idea of the request zone calculation. There is a 'bootstrapping' problem that needs to be addressed in the request zone calculation protocol. How will the dominator know the location information of the sender when it sends the unicast for the first time ? If the dominator is also dominating a node which subscribes to the same multicast group, it would have the location information of the sender (location information is included in all the multicast data packets). It will then include this information in the join core broadcast unicast packet. Else the dominator will leave the sender location fields empty hoping that some core nodes in the path that knows the correct location of the sender will enter in the values there. However this approach is still better than the join protocol in MCEDAR.

4.3.4 The Process of Unicast Trapping

When the dominator sends the unicast message it need not necessarily travel all the way from the subscriber to the sender. When a primary router on the unicast path from the subscriber to sender, which is also a dominator for a node subscribing to the same multicast group, gets this unicast message, it traps and terminates the unicast. That node is however subscribing to the same multicast group from the same sender and would have sent a unicast message to the sender. We term this process as $trapping$. Figure 4.3 depicts the overall idea of the Unicast Trap algorithm.

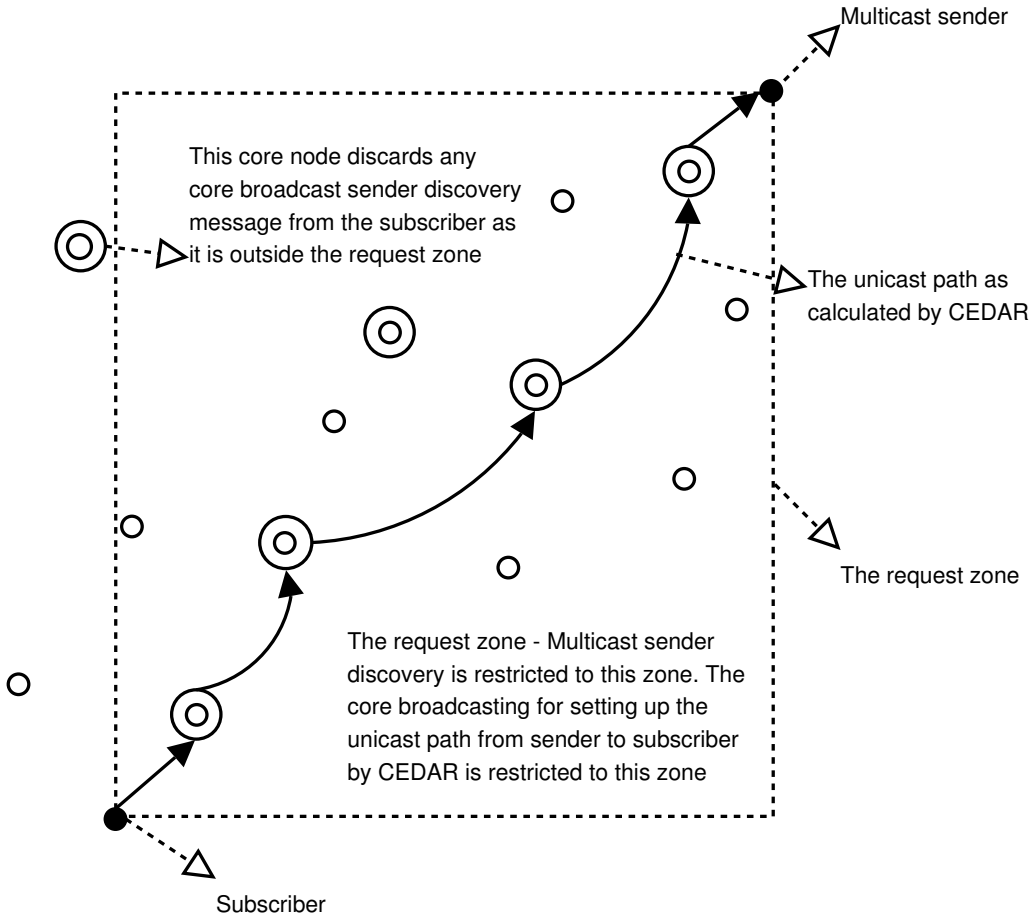

Figure 4.2: Request zone calculation restricts the propagation of join discovery messages

4.4 Tables involved in NAMO NAMAHA and the Modifications Required for the Beacon Messages in CEDAR

The dominator core node will have to maintain the following:

- Multicast membership information for every node it dominates.
- Multicast groups for which it acts as a primary router.
- Multicast groups for which it acts as a secondary router.

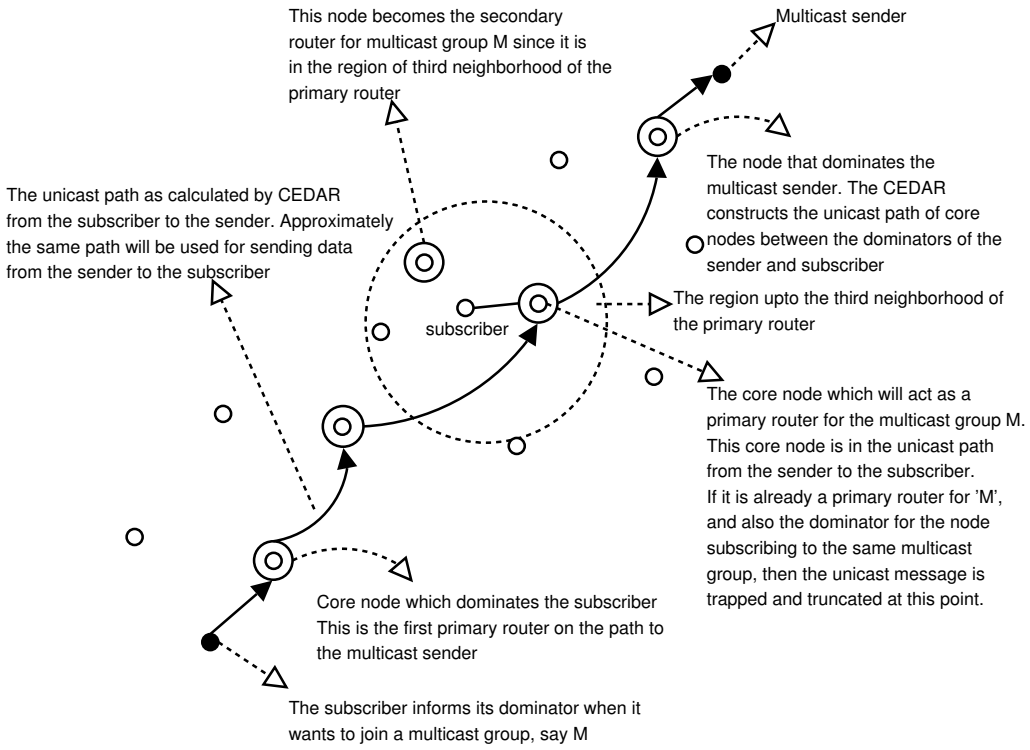

Figure 4.3: The Unicast Trap algorithm

A core node exchanges this information with the core nodes in its third neighborhood. This would involve the modification of the beacon messages used in the CEDAR protocol. In CEDAR, When a node u joins the core , it periodically issues a piggybacked broadcast in $N_3'(u)$. A piggybacked broadcast is accomplished as follows. In its beacon, u transmits a message: $(u, DOM, 3, path\_traversed \leftarrow null)$. DOM denotes the id of $u's$ dominator. When node w hears a beacon that contains a message $(u, DOM, i, path\_traversed)$, it piggybacks the message $(u, DOM, i-1, path\_traversed + w)$ in its own beacon if $i - 1 > 0$. Thus, the piggybacked broadcast of a core node advertises its presence in its third neighborhood. This guarantees that each core node identifies its nearby core nodes, and cet up virtual links to these nodes using the $path\_traversed$ field in the broadcast messages. Thus each core node has enough local topology information to reach the domain of its

nearby nodes and set up virtual links without having the knowledge of the whole core graph.

NAMO NAMAHA requires following changes to the beacon message. Each beacon message should also contain a list of multicast groups for which the beacon originating core node is a primary router and a list of multicast groups for which it is a secondary router. So the new beacon message would be $(u, DOM, < primary - router - list >, < secondary - router - list >, 3, path\_traversed \leftarrow null)$. The $path\_traversed$ should be of the form $n_1 - c_{12} - n_2 - c_{23} - n_3 ...$, where n_i, is node i and c_{ij} is the cost for taking the link between nodes i and j. The receipt of $path_traversed$ in this format from all the core nodes in the third neighborhood in effect gives the local topology information for the core node that is receiving the beacon message.

When a beacon message from a core node reaches another core node, if the beacon originating node is a primary router for a multicasting group, then the beacon receiving node flags itself as a secondary router for that multicasting group. Each core node stores the primary router list and the secondary router list for every other core nodes in its third neighborhood. Figure 4.4 shows the table maintained by each of the core node.

4.5 The Timers in NAMO NAMAHA

Following timers are used in NAMO NAMAHA:

- $membershipRefreshTimer$: A subscriber should periodically send membership refresh information to its dominator. Else the dominator will assume that the node is no longer subscribing
- $primaryRouterLifeTimer$: A node will act as a primary router only for the time length specified by this timer. If a primary router does not receive a fresh

unicast message from the subscriber to the sender, it ceases to act as primary router

- $secondaryRouterLifeTimer$: A node ceases to act as a secondary router for a multicast group if it does not hear a refresh message from some primary router for that multicast group within the time length specified by this timer
- $unicastMessageTimer$: The dominator of a subscriber should send periodic unicast messages to the sender based on this timer. The reason for this periodic unicast message is to refresh the trail of primary routers that would be used to route the multicast data packets from the sender to the subscriber. Due to node mobility, primary routers may move away, or the unicast route from the subscriber to the sender might become outdated. However this period should be large enough to justify the advantages of such a join protocol. Each time the unicast message is sent it is not necessary that the message be relayed up to the sender. The unicasting may be truncated midway if it encounters a primary router, which is also a dominator for some subscribing node, on its path to the sender. In the next subsection we outline the guidelines to select the values for $unicastMessageTimer$.

4.5.1 Guidelines for Selecting Optimum Unicast Interval

The period for unicasting from the subscriber to sender is so set that the number of unicast messages is much less compared to the number of multicasting messages sent. And at the same time not too few to disconnect the node from the multicasting tree for a long time. The interval is set based on the following guidelines:

- The value of $unicastMessageTimer$ should be inversely proportional to the mobility of the ad hoc networks. Super mobile ad hoc networks will have

smaller unicast interval and relatively less mobile ad hoc networks will have larger unicast intervals.

- The value of $unicastMessageTimer$ should be inversely proportional to the distance of the subscriber from the sender. If the distance of the subscriber from the sender is more, the core path would involve more core nodes and the chance that the path breaks due to some primary router moving out increases. So fresh routes will have to be established more often.
- $secondaryRouterLifeTimer$ should be large enough so that even if the primary router ceases to exist, the secondary routers can still route packets from the sender to the subscriber in the absence of the primary routers. A new set of primary routers might get created when the subscriber's dominator sends a unicast message to the sender after $unicastMessageTimer$.

4.6 Forwarding Protocol

4.6.1 Introduction

From [CN02], we have seen that if the location information is up to date, LGS algorithm outperforms LGK algorithm. The fundamental idea of NAMO NAMAHA (see Figure 4.1) is to use the Steiner tree heuristics over a subgraph of the core graph computed as in [SSB99a]. This subgraph comprises of the primary routers and the secondary routers. Unlike [CN02], the forwarding protocol builds Steiner trees based on network hops, delay or any other link cost directly, rather than using the approximation relating geometric distance to the number of network hops. Core nodes of the approximate minimum dominating set would have the information about all other core nodes in its third neighborhood

and also about the cost and the bandwidth availability information of the network links in this region. So the core nodes can immensely help the forwarding protocol in the creation of overlay data distribution tree. Since this locally gleaned information is up to date, we construct Steiner trees in the regions where this information is available, that is the third neighborhood region of the core nodes; and then connect them.

4.6.2 The Big Picture

As long as we have some primary router to be added to the Steiner tree, we do not consider the secondary routers. The Steiner tree algorithm continues to span the whole network. In case there are no primary routers to be added to the Steiner tree, the secondary routers are considered. This happens when the primary router in the core path has moved elsewhere due to its mobility. In case there are no secondary routers too, then that would mean with high probability that there are no subscribers in that region. RTS/CTS caching method as used in the core broadcast mechanism of CEDAR can be used for avoiding two cases:

- A core node sending the data back to the core node already in the partially constructed Steiner tree, that is a core node sending data to its ancestor, which in fact has already got the data.
- A core node getting added twice to the Steiner tree as it might be in the third neighborhood of two or more core nodes which are already in the Steiner tree - In this case the node will be added by only one node in the Steiner tree and the other nodes will refrain from sending data to it since they will learn from the RTS/CTS mechanism that the node has already received the data. This also prevents the formation of loops (cycles in the Steiner tree). Figure 4.5 gives a graphical representation of the forwarding protocol.

The Steiner tree construction is always restricted to the third neighborhood region. The sender initially constructs the Steiner tree in its third neighborhood. We can achieve this by setting $TTL = 3$ at the sender. The TTL is decremented at each node. The core nodes which get the multicast data packet with $TTL = 2$ or $TTL = 1$ do not start the calculation of new Steiner trees. It actually participates in the Steiner tree that is already being calculated. Once the TTL becomes zero, new Steiner tree calculation is initiated - at the periphery of the third neighborhood region; the TTL is set to 3 again at these nodes. How to build Steiner trees in a distributed fashion? This question is answered in the next three subsections.

4.6.3 Graph Theoretical Formulation for the NAMO NAMAHA Forwarding Protocol

Steiner tree calculation is a NP-Hard problem. Starting with Takahashi Matsuyama heuristic [TM80], demonstrated in 1980, many approximation heuristics have been proposed to solve the Steiner tree problem. Only a subset of Steiner tree heuristics has the properties that make them suitable for distributed implementation in networks where nodes have minimal routing information. For a network of nodes, especially mobile nodes in a wireless environment, the centralized algorithms perform very badly since each node needs to have the knowledge of the global topology. The problem of building multicast data distribution tree in a network of mobile wireless nodes with bidirectional links is exactly equivalent to the construction of Steiner trees in an undirected graph. [NRK01] suggests a distributed heuristic, by name $(CHINS)_T$ (Cheapest Insertion Heuristic with table), as an approximation to the Steiner tree construction problem. In this section, we formulate the multicast

data distribution tree construction problem as a Steiner tree problem. In the next subsection, we detail the $(CHINS)_T$ heuristic [NRK01], and in the subsection that follows, we adopt this heuristic for the NAMO NAMAHA forwarding protocol.

Definition: Let $G = (V, E)$ be an undirected connected graph of the communication network with the nodes V, the connections set E, and non-negative weights associated with the connections. In this graph we have a set $Q \in V$ of destination nodes, called the multicast group. A Steiner tree problem is to find a minimum cost subgraph of G, such that there exists a path in the subgraph between every pair of destination nodes. In order to realize this subgraph, additional nodes from $V \backslash Q$ may be included.

4.6.4 $(CHINS)_T$ (Cheap Insertion Heuristic) - A Distributed Steiner Tree Construction Algorithm

The $(CHINS)_T$ algorithm can be explained non-formally as follows [NRK01]. The algorithm starts with the sender node. The group member node that was added last decides, according to the information available in the table, which member node should be added next. A table is a list of triplets of the form (x, y, k) where x is a vertex which is not yet in the tree, and the nearest node that is already in the tree is y which is at a distance k from the vertex x. It then passes the table to the tree node from which the just selected minimum cost path proposition originates. The selected node establishes a connection with the nearest group member node and passes the table through the intermediate nodes to the newly connected node. The intermediate nodes on the newly created path, including the destination node, update the table according to their local information. If the node has a cheaper proposal for a specific unconnected group member node, it replaces the table record for that node. The distributed algorithm terminates its execution when all the member nodes are

connected. The algorithm is repeated at the nodes that receive the multicast data packets with $TTL = 0$. Figure 4.6 shows the graphical representation of the algorithm. A is the sender node and B,C,D,E and F are the multicast members. The tables are shown next to the nodes. Messages of type $Connect$ establish a new path from the partially built tree, while messages of the type $Pass$ only transfer the table to the next selected node. The $Connect$ messages can be piggybacked with the copy of the multicast data packet which we have to however send to the core nodes that are primary routers.

4.6.5 $(CHINS)_T$ adopted to the NAMO NAMAHA Forwarding Protocol

In NAMO NAMAHA, the core nodes, which are primary routers (secondary routers if primary routers are not present) form the multicast destinations.

The algorithm starts with the sender node. In the $(CHINS)_T$ heuristic, each node that is involved in the Steiner tree computation should know the minimum cost to every other node. But in the NAMO NAMAHA forwarding protocol, this requirement is compromised in the following two ways:

- There are some node pairs, the nodes that are involved in the Steiner tree computation, that do not know the minimum cost between them. For example, in Figure 4.7, node L does not know the minimum cost to node R and vice versa. This is because node L and node R are not in the third neighborhood region of each other. However node L and node R are involved in the calculation of the same Steiner tree since both of the nodes are in the third neighborhood of the sender node S, from where the Steiner tree computation starts.

- The nodes that are not the core nodes do not have the minimum cost information to the other nodes that are involved in the Steiner tree computation.

The first compromise does not affect the Steiner tree computation with $(CHINS)_T$ algorithm much, since it is very unlikely that the closest core node not in the partially constructed Steiner tree is beyond the third neighborhood of some node in the tree.

NAMO NAMAHA forwarding protocol deals with the second compromise by having the core nodes do the 'minimum cost to other core nodes' computation for the non core nodes. The core nodes will have the topology information for the subgraph comprising of itself and all the core nodes in its third neighborhood (owing to the periodic CEDAR beacon messages). The non-core nodes do not perform any computation since they are not aware of the local topology.

For example, in the figure 4.6, core node F does the computation on behalf of the non-core node G, and then sends the $connect$ message to E, the newly added core node. G does not perform any computation; but G records the path to the nearest core node from it, in this case to the core node D. When the core node E sends the $pass$ message back to G, G knows how to get to D. In case of the node K, the computation is done by D and the table is updated in the $pass$ message. On receiving the $pass$ message which includes a route to the core node B, K knows how to get to B.

4.7 Leave Protocol

The Leave protocol is really trivial. If a node wants to remove itself from a multicast group it simply sends a leave message to its dominator and the dominator updates its group membership table. There is no need to inform the primary routers on the way from the node to the sender because those nodes could be the primary routers for the same multicasting

group for some other node. The dominator will continue getting the multicast packet from the multicasting group if it is still dominating some of the nodes that are subscribing to the same multicasting group; it just does not send the packet to the node that just left the multicasting group. If the last subscriber quits the multicast group membership, pruning happens automatically. The dominator ceases to be the primary router for that multicast group after sometime and hence will stop receiving the multicast data packets.

Each core node maintains

- list of nodes for which it is the dominator
- For each node it dominates, the list of multicast groups to which the node subscribes
- List of multicast groups for which it is a primary router
- List of multicast groups for which it is a secondary router
- List of core nodes in the region upto its third neighborhood
- For each core node, its primary router list
- For each core node, its secondary router list
- For each core node, the least cost path and its cost

A sample table at a core node would look like this

<dominee list>: <N1, N2>
<multicast membership information for each dominee>:
    <N1: M1, M2>,<N2 : M3, M5>
<primary router list>:<M1, M2, M3, M5, M7>
<secondary router list>:<M6, M1>
<core nodes list upto third neighborhood>:<C1, C2>
< primary router list for each core node>:
    <C1: M3, M5>,<C2: M6, M8>
<secondary router list for each core node>:
    <C1: M1>,<C2:M9>
<path information for each core node>
    <C1: least_cost_path, cost>, <C2: least_cost_path, cost>

N: Node
M: Multicast group
C: Core node

Figure 4.4: Information maintained at the core nodes

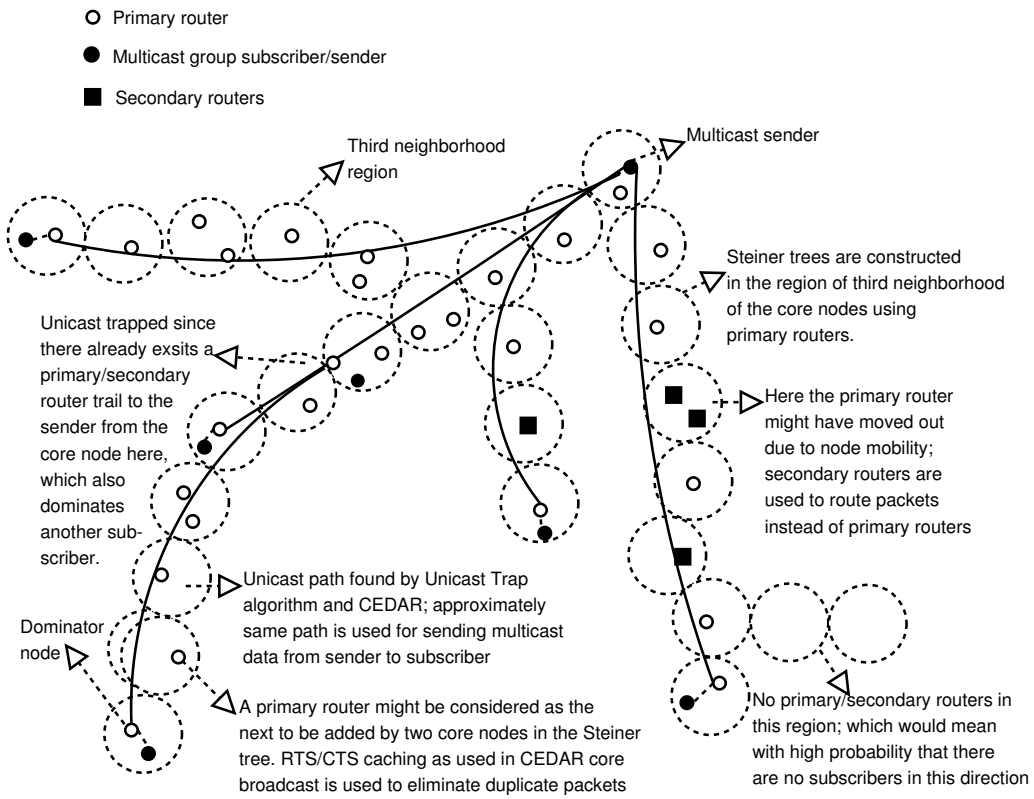

Figure 4.5: Incremental Steiner tree construction by the NAMO NAMAHA forwarding protocol using primary routers and secondary routers

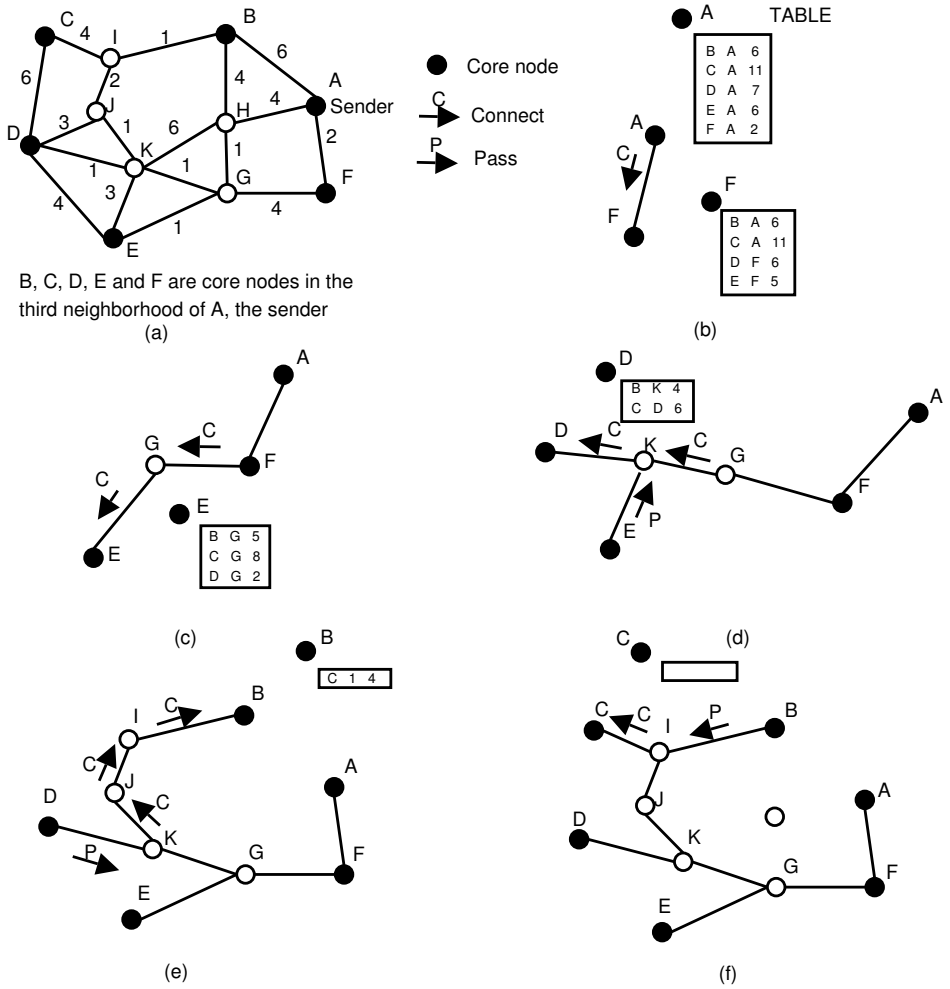

Figure 4.6: Distributed Steiner tree construction for the Forwarding Protocol with $(CHINS)_T$ Algorithm

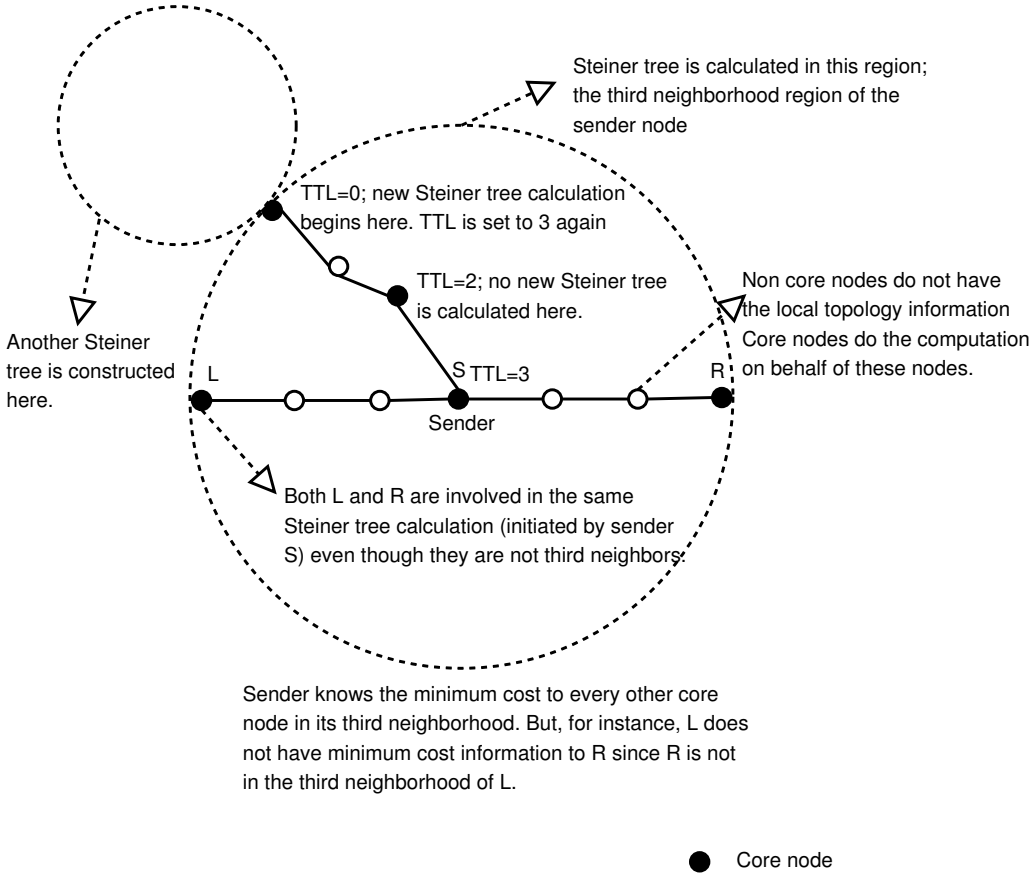

Figure 4.7: Steiner tree calculation with partial information - A need to adopt $(CHINS)_T$

Chapter 5

Complexity, Correctness and Comparison Analysis for NAMO NAMAHA Algorithms

In this chapter, we arrive at the time, memory and message complexities of all the algorithms in NAMO NAMAHA and then compare our protocol with MCEDAR.

5.1 Assumptions

While analyzing the algorithms, we assume a constant bound k, on the number of neighbors a node can have. The value of k is choosen to be in tens. Furthermore, we also assume a constant bound, again in tens, on the number of active multicast groups in the ad hoc network at any given time. These two assumptions are practical since in real world scenarios, the number of nodes surrounding a given node is limited; and also one does not see more than tens of active multicast group at any given point of time. This assumption also reflects on the number of core nodes a given core node can have in its third neighborhood. In any given graph, the number of nodes required to form a Minimum Dominating Set is

much less than the number of vertices. Consider any arbitrary spanning tree for the underlying graph. In such a tree, to form a Dominating Set, we can select the root, not select its children and grand children, select all great grand children and so on; that is we select the root, skip two levels, select all the nodes in the next two levels, and then skip two levels and so on (that way every node is either in the set or is a neighbor of a node in the set). Since we are forming the Minimum Dominating Set, the number of nodes in this set will be smaller than the Dominating Set. And since we are select the primary routers which is a subset of core nodes, it is reasonable to assume that the number of primary routers in the third neighborhood of some primary router is also bounded by a constant. We retain these assumptions for the rest of the chapter.

5.2 Analysis of the Core Computation Algorithm

Finding the MDS is a NP-hard problem that is also hard to approximate. The best known distributed algorithm for MDS approximation [GK98] is a greedy algorithm that requires $O(D)$ steps and has a competitive ratio of $log(|V|)$, where D is the diameter of the network. However, this algorithm requires global computation (i.e. the result of step i at node u can affect the computation of step $i+1$ at node v). The CEDAR scheme has chosen to use a robust and simple constant time algorithm which requires only local computations and generates good approximations for the MDS in the average case. Consider a node u with first deleted neighborhood $N_1'(u)$, degree $d(u) = |N_1'(u)|$, dominator $dom(u)$, and effective degree $d^*(u)$, where $d^*(u)$ is the number of its neighbors who have chosen u as their dominator. The core computation algorithm works as follows at node u [SSB99a]

1. Periodically, u broadcasts a beacon which contains the following information pertaining to the core computation: $(u, d^*(u), d(u), dom(u))$

2. If u does not have a dominator, then it sets $dom(u) \leftarrow v$, where v is the node in $N_1(u)$ with the largest value for $(d^*(v), d(v))$, in lexicographic order. Note that u may choose itself as the dominator.

3. u then sends v a unicast message including the following information: $(u, \{(w, dom(w)) | \forall w \in N_1'(u)\})$. v then increments $d^*(v)$

4. If $d^*(u) > 0$, then u joins the core.

The node u here sends the beacon message only to its neighbors, since u can only have itself or one of its neighbors as its dominator. This takes $O(|N_1(u)|)$ messages at the most. Node u will also have to perform a lexicographic sort on $(d^*(v), d(v))$. This would take $O(|N_1(u)|lg(|N_1(u)|))$ computations. Since we can set an upper bound (say k) on the number of nodes surrounding a given node, the lexicographic sorting can be done in $O(2(|N_1(u)| + k))$ steps using radix sort[1] algorithm over the $(d^*(v), d(v))$ pairs; using stable counting sort as an intermediate sort. Since $d(V)$, $d^*(v)$ including V are actually in the order of k, the algorithm runs in $O(k)$ time. The core computation algorithm is hence a constant time algorithm.

5.3 Analysis of the Core Broadcast Algorithm

In wireless networks, flooding probes causes repeated local broadcasts and is highly unreliable because of hidden and exposed terminals. The core broadcast as discussed in Section 3.5 of Chapter 3, which is based on reliable unicast (using RTS-CTS-etc.), is used to solve this problem [SSB99a]. The core broadcast mechanism ensures that each core node

[1]Radix sort (using counting sort as intermediate sort), of n numbers, each of which has d digits, with each digit not greater than k takes $O(nd + kd)$ time [CLRS01].

does not transmit a broadcast packet to every nearby core node. CEDAR uses a close coordination between the MAC layer and the routing layer to achieve efficient core broadcast.

Assume that some nodes u and v are first neighbors. If node u has heard a $CTS(M, v)$, it realizes that node v has already received M and does not forward M to node v. Now assume that node u and v are second neighbors with the virtual channel $[u, v]$ passing through a node w. Since w is a neighbor of v, w hears $CTS(M, v)$. Thus when u sends a $RTS(M, v)$ to w, w sends back a NACK back to u. Finally, if u and v are a distance 3 apart, using the same argument, there will be at most one extra message transmission. The RTS and CTS packets are monitored in the channel in order to discover when the intended receiver of a core broadcast packet has already received the packet from another node, and suppress the duplicate transmission of this packet. Since there is at most one extra message involved in the core broadcast when u and v are third neighbors, the core broadcast message overhead is in the order of $O(|V|)$. See Figure 5.1

5.4 Analysis of Periodic Beacon Message Transmission

The beacon messages for NAMO NAMAHA contains a list of multicast groups for which the beacon originating core node is a primary router and a list of multicast groups for which it is a secondary router, apart from the information needed by the CEDAR protocol. So the new beacon message would be: $(u, DOM, < primary - router - list >, < secondary - router - list >, 3, path\_traversed \leftarrow null)$. Furthermore, $path\_traversed$ will be a list of the form $n_1 - c_{12} - n_2 - c_{23} - n_3$, where n_i is node i and c_{ij} is the cost of the link between node i and node j, which could be, for example, the delay information for taking the path between node i and node j. NAMO NAMAHA uses the CEDAR core broadcast to transmit these periodic beacons messages. In the previous section, we noticed that the

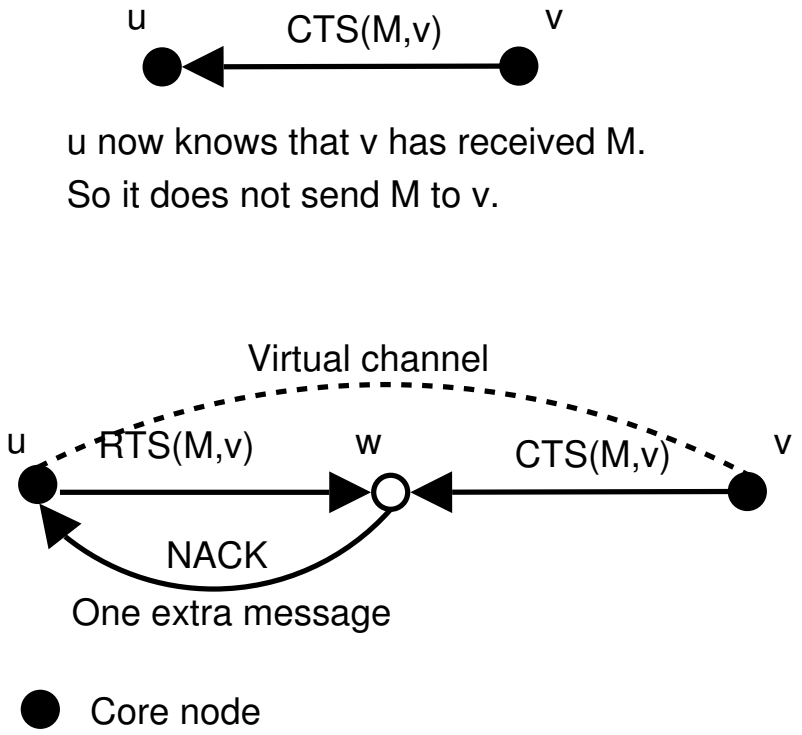

Figure 5.1: Core Broadcast has a $O(|V|)$ message overhead

complexity of the CEDAR core broadcasting is $O(|V|)$. On getting these beacon messages, the core nodes should maintain a table and also perform minimum cost computation for the core nodes in its third neighborhood. Below we analyze the time complexity for such a computation and the memory complexity for storing such tables.

5.4.1 Time complexity for Computing 'minimum cost' Values

Each core node will have to calculate the best path to the other core nodes and record it. We can use Dijkstra's single source shortest path algorithm which is known to run in $O(|V^2|)$ time. This computation is however done by the CEDAR protocol. To adopt the $(CHINS)_T$ algorithm to the NAMO NAMAHA forwarding protocol, we require that the

core nodes calculate the 'minimum cost' information for the non-core nodes too. Between any two core nodes that are within the third neighborhood of each other, the number of non-core nodes cannot exceed two, since if it does, the core nodes will not be in the third neighborhood region of each other. Therefore, each core node will have to run Dijkstra's single source shortest path algorithm on behalf of at most two non-core nodes. The time complexity does not exceed $O(|V^2|)$.

5.4.2 Memory Complexity for the Storage of Tables

For convenience, in Figure 5.2, we reproduce a part of Figure 5.1. Line 1 in Figure 5.2, takes $O(|V|)$ space since there can be atmost V neighbors. Line 2 would take $O(|mV|)$, where m is the number of active multicast group in the ad hoc network. Line 3 and line 4 would each take $O(|m|)$ space. Line 6 and 7 would take $O(|mV|)$ each, while line 5 and 8 would take $O(|V|)$ each. Since the number of neighbors surrounding a node, the number of core nodes in the third neighborhood and the number of active multicast groups in the ad hoc network are all bounded by a constant in tens, the tables would incur a constant memory overhead at the core nodes. Furthermore, the core node does not transmit this entire table in each beacon message. It transmits the delta list (the change from the previously transmitted table) instead, which is very much smaller than the actual table. Most of the information maintained in the table does not change frequently.

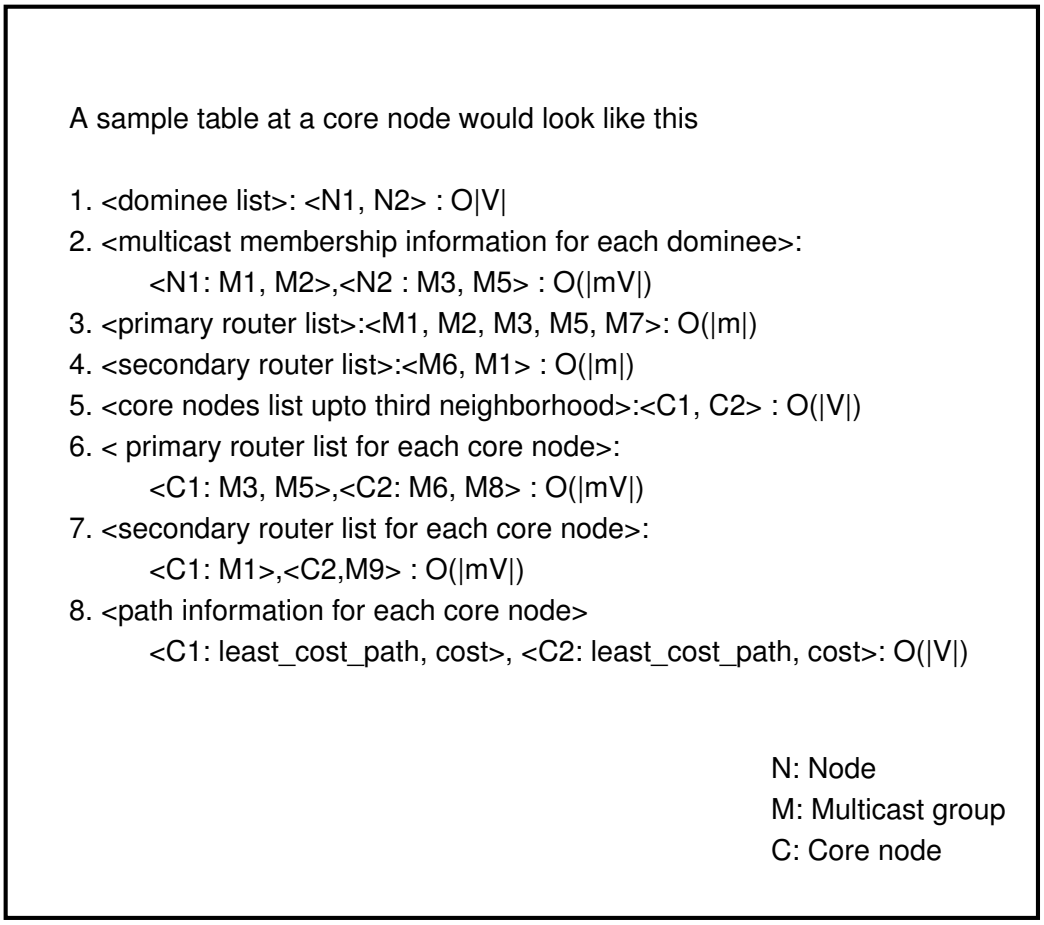

Figure 5.2: Memory complexity of core node tables

5.5 Analysis of the Unicast Trap Algorithm and the Join Protocol

When a node wants to subscribe to a multicast group, its dominator sends the unicast message to the sender of the multicast group. The unicast routing in the CEDAR protocol involves core broadcasting whose message complexity is $O(|V|)$, as shown in Section 5.3. The unicast messages sent for discovering the sender need not necessarily propagate all the way up to the sender, since the unicast message is 'trapped' if it encounters a primary

router (which is also a dominator to a node subscribing to the same group) on its way to the sender. Furthermore, since NAMO NAMAHA computes the request zone before sending the unicast message, the sender discovery core broadcast is restricted to a very small region of the entire network; only to the region where the discovery process is required.

5.6 Analysis of the Forwarding Protocol

Finding Steiner trees is a NP-Hard problem. NAMO NAMAHA adopts the $(CHINS)_T$ algorithm to implement the Takahashi Matsuyama heuristic, which is adding the nearest unconnected destination to the partially constructed Steiner tree. $(CHINS)_T$ implements the Takahashi Matsuyama heuristic in a distributed way. Below we analyze the message and time complexity of the $(CHINS)_T$ algorithm.

5.6.1 Complexity of the Forwarding Protocol

Although the execution of the $(CHINS)_T$ algorithm is distributed among network nodes no parallel activities are performed (no parallel actions are performed when a single Steiner tree is in the process of construction. But throughout the network, many such Steiner trees will be constructed in parallel in all the directions where the subscribers exist). A particular node receives a message, performs a computation and sends a new message to the chosen node. Thus, the number of sequentially exchanged messages is equal to the number of all exchanged messages. The number of exchanged messages of the type $Connect$ is $|V_T| - 1$. Each core node gets added to the approximate Steiner tree through a $Connect$ message. This means that the $Connect$ message passes through all the edges and the number of edges in a tree with $|V_T|$ vertices is $|V_T| - 1$ which is exactly the number of $Connect$ messages exchanged. Messages of type $Pass$ are sent atmost $(|Q| - 2)$ times between

pairs of tree nodes, using the minimum cost paths. $Pass$ messages are generated only by the core nodes that have just received the $Connect$ message. The core node that last received the connect message might not have a better path to the core nodes that are not in the tree. This prompts the core node to send the $Pass$ message. $Pass$ message is generated not more than once by any of the core nodes since the core nodes receive the $Connect$ message exactly once. Therefore the number of $Pass$ messages generated is definitely not more than $|Q|$. It is at most $(|Q| - 2)$ since the first and last core nodes that are added to the tree cannot send the $Pass$ message. A rarely achieved upper bound for the length of these paths is $|N_3'(u)| - 1$, where u is the node that initiates the approximate Steiner tree calculation. A better approximation would be $depth(T)$, which can be achieved by sending messages of the type $Pass$ using tree connections. As given in [NRK01], in the former case the execution time and the number of exchanged messages are bounded by $O((|Q| - 2)|N_3'(u)| + |V_T| - 1)$ and in the latter case the execution time by $O((|Q| - 2)depth(T) + |V_T| - 1)$ and the number of exchanged messages by $O((|Q| - 1)(|V_T| - 1))$. However, the need to find the single source shortest path distance by a given core node to all other core nodes (such shortest paths should be computed for the non-core nodes too by the core nodes) would take $O((|N_3'(u)|)^2)$ time, which is larger than all the above complexities. A core node calculates the single source shortest path to not more than two non-core nodes, since the core nodes in the third neighborhood are not separated by more than two network hops.

We need not send a separate $Connect$ message every time a core node gets added to the tree. We can piggyback the $Connect$ message with the copy of the multicast data which we have to anyway send. So the actual message overhead is only that of the $Pass$ messages, which is not more than $|Q| - 2$. This is tolerable as $|Q|$, which is the number of core nodes that are primary routers in a given third neighborhood, cannot be unreasonably large.

What is the additional overhead for the table that needs to be passed with the $Connect$ message? We discuss the size of the table that is being passed. Initially only the Steiner tree initiator node is in the tree and the table will contain the minimum distance of all the core nodes in the third neighborhood of the tree initiator to itself. After this point, the table only shrinks; the table cannot have more than $|Q| - 1$ entries. The size complexity of the table is $O(|Q|)$.

5.6.2 Correctness of the Forwarding Protocol

In proving the correctness of the forwarding protocol, we assume that the ad hoc network is a static topology. We prove a set of lemmas under this assumption. This assumption is required in proving the correctness because it is impossible to guarantee data delivery in the face of random node mobility. So assuming that the topology is static, we prove the correctness of the forwarding protocol. Later, during the analysis, we remove this requirement and give some qualitative comments on the correctness of the forwarding protocol. Absolute guarantees for path availability cannot be given in ad hoc networks.

Lemma 1: If the underlying graph is connected, then the core graph is also connected.

Proof: We prove that for a connected graph G with diameter greater than 2, the core graph formed by the core computation algorithm is connected. For this assume that S is any dominating set. For each node $v \in S$ there must be at least one other node of S in $N_3'(v)$. Otherwise there is at least one node in G which is neither in S nor has a neighbor in S. From the definition of core if G is connected then so is the core graph formed by the core computation algorithm.

Lemma 2: A primary router will always have at least two primary routers in its third neighborhood (except for the dominator of the sender and the subscriber, which will have at least one)

Proof: The unicast route between the subscriber and the sender is a core path comprising of core nodes. CEDAR builds this unicast path using core nodes, adding core nodes to the already constructed core path from the third neighborhood region. Consider the core node that is last in the partially constructed core path. The core node that gets added to the partially constructed core path has to be in the third neighborhood of this core node. And the core node that got added before this to the core path should also be in its third neighborhood. since the primary routers are nothing but the core nodes in the core path from the subscriber to the sender, any primary router will always have at least two more primary routers in its third neighborhood.

Lemma 3: For a given multicast group, the subgraph of the core graph comprising of sender, subscriber, primary router and secondary router is connected through virtual links.

Proof: By Lemmas 1 and 2, the sender, all the subscribers and the primary routers are connected. Since secondary routers are chosen from the third neighborhood of the primary routers, to which the primary routers will have a virtual path, the sender, subscribers, primary routers and the secondary routers are all connected through virtual links.

Lemma 4: There are no cycles in the multicast data distribution graph

Proof: In the third neighborhood regions, Steiner tree is constructed and trees cannot have cycles. The possibility that a single node participates in the calculation of two or more Steiner trees is eliminated by the RTS/CTS caching mechanism as used in CEDAR core broadcasting. There can be no cycles.

Lemma 5: A primary router will be eventually added to the Steiner tree.

Proof: By Lemma 3, the graph comprising of primary routers, secondary routers, sender and subscribers are all connected. Steiner tree builds a tree over this connected graph, connecting the core nodes that are either primary or secondary routers (using the non-core nodes as the intermediate connection points). Since the tree is connected and the

underlying graph is connected, all the primary routers should be in the Steiner tree.

Since the dominator of a subscriber is itself a primary router, it will eventually be added to the Steiner tree. Since the subscriber and its dominator are connected, the subscriber gets the data. The fact that the subscriber gets the data and the fact that there can be no cycles in the multicast data distribution graph, proves the correctness of the NAMO NAMAHA forwarding protocol.

We remove the static topology requirement now and comment on the path availability. Assume that some arbitrary primary router is no longer on the core path because of the node mobility. From Lemma 2, we know that this primary router had at least two primary routers in its third neighborhood. With high probability there exists at least one secondary router, which is within the third neighborhood of these two primary routers. In the absence of the missing primary router, such a secondary router will provide the connectivity between the sender and the subscriber and the data still reaches the subscriber. This could happen even if all the primary routers move out of their way. But the probability that all the primary routers move out of their way is extremely low. With high probability, the sender always gets the data. Furthermore since the unicast message is sent periodically, new primary router paths are set which means in the worst case, the subscriber eventually gets the data from the sender after some finite number of unicast messages from the subscriber to the sender, in the practical case.

5.6.3 On the Efficiency of Forwarding Protocol

The $(CHINS)_T$ algorithm which implements the Takahashi Matsuyama heuristic efficiently is shown to be a good approximation to the NP-Hard Steiner tree problem. In NAMO NAMAHA, approximate Steiner trees are built in the third neighborhood regions using the $(CHINS)_T$ algorithm and such Steiner trees are connected. We comment on the

efficiency of such an algorithm.

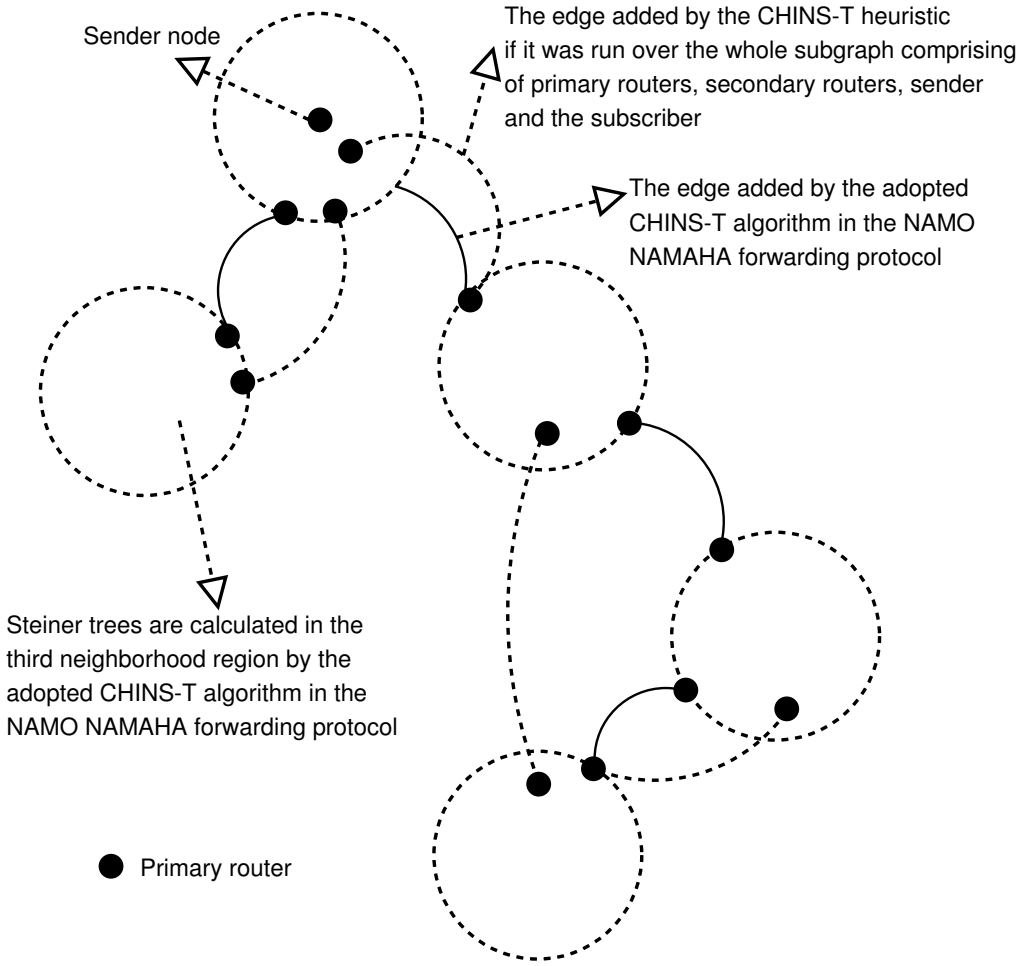

Figure 5.3: The edge between the two locally constructed Steiner trees

The Steiner tree construction starts from the sender node. As shown in Figure 5.3, compromises occur when we add an edge connecting the Steiner tree already computed in one third neighborhood and the one that is about to be constructed in a different third neighborhood. This is because, had we considered all the nodes in both the neighborhoods and then constructed a Steiner tree using the adopted $(CHINS)_T$ heuristic, a different edge would have been added. But the edges that gets added when constructing the Steiner

tree in the third neighborhood region are the right ones with high probability since it is very unlikely that a core node beyond the third neighborhood region of the nodes in the locally and partially constructed Steiner tree has a better edge to the node that is being added next (that is, it is very unlikely that a non-local edge will be added to the partially constructed Steiner tree in the third neighborhood region) . Hence, usually efficient Steiner trees are constructed by the NAMO NAMAHA forwarding protocol.

5.7 Analysis of the Leave Protocol

This analysis is really trivial. When a node wants to stop subscribing to a multicast group, it simply sends a leave message to its dominator node and the dominator removes the node from the list of nodes for that multicast group. This is a constant time and constant memory operation.

5.8 Overall Analysis of NAMO NAMAHA Protocol and Comparison with MCEDAR

The first part of this section summarizes the complexities for the NAMO NAMAHA protocol. In the second part, we compare our work with the MCEDAR protocol in terms of the cost of the multicast data distribution trees, the number of messages exchanged in building them and the time and memory complexity of the algorithms involved. Figure 5.4 summarizes the time and memory complexity for all the algorithms and data structures involved in the NAMO NAMAHA protocol. We assume a constant bound of k on the number of neighbors a node can have. k is assumed to be in tens.

Both MCEDAR and NAMO NAMAHA use CEDAR as the underlying unicast protocol. MCEDAR just uses the unicast capability of the CEDAR protocol, while NAMO NAMAHA tries to fully make use of the local network knowledge possessed by the core nodes apart from its unicast routing capability. We start the comparison with the join protocol. In MCEDAR, whenever a node wants to join a multicast group, it requests its dominating core node to perform the join operation. The dominator then performs the join operation by core broadcasting the $join\ request$ message. Since the dominator does not have any clue as to where the sender node is, this join request message is propagated all over the network. The precious bandwidth resource is wasted here. In NAMO NAMAHA, the sender discovery core broadcast is restricted only to the request zone as calculated by the node that wishes to subscribe. The join protocol is fairly complicated in MCEDAR with the nodes maintaining different data structures, calculating join IDs, maintaining the robustness factor R and issuing acknowledgements. Acknowledgements should be avoided in ad hoc networks, as far as possible, due to the mobile nature of ad hoc networks. Routing the acknowledgement back to the sender is no simpler than routing the data to the receiver. The join protocol in NAMO NAMAHA is fairly simple.

Unicast messages are sent periodically from the subscriber to the sender. In MCEDAR, the node periodically issues join requests, when the cardinality of the node's parent set is less than half the robustness factor R. Apart from this, whenever a node loses connectivity with all its parents, it issues a join request. NAMO NAMAHA has periodic unicast messages, while MCEDAR has periodic join requests. In MCEDAR, each time a join request is issued by any given node, the request propagates all over the network. This happens since MCEDAR does not make use of the location information. But in NAMO NAMAHA, the unicast message sent from the node that wishes to subscribe, to the sender may be trapped by a core node which is dominating a node that has already subscriber to that multicast

group. Because of this and assuming that the multicast subscribers are randomly scattered all over the network, the number of transmission by the nodes for the unicast is limited by the number of edges in the network graph; in fact to the set of edges in the request zone between the subscriber and the sender. The unicast interval should be set large enough to justify the use of such a protocol. The unicast path is approximately the path the multicast data packet should take from the sender to the subscriber, in reverse direction.

The trail of primary routers and the secondary routers created when the unicast message is sent provide this directional aid for the multicast packets. The existence of primary and secondary router trail in this unicast path ensure that a path would exist between the sender and the subscriber at any given point of time with high probability; and that path is approximately the best path. Such a guarantee cannot be given in MCEDAR.

We now shift our focus onto the forwarding protocol of NAMO NAMAHA and MCEDAR. MCEDAR forwarding protocol uses core broadcasting. In NAMO NAMAHA, the multicast data distribution tree is restricted to only where it is required. The primary and secondary router trails created by the unicast trap algorithm provide the directional aid for the multicast packets. In MCEDAR, if the $child\ set$ and the $parent\ set$ is not up to date, the multicast data packets are core broadcasted all over the network in a hope of finding a subscriber. In NAMO NAMAHA, approximate distributed Steiner tree algorithm, $(CHINS)_T$, computes a tree that spans all the core nodes that are primary routers. Therefore the resulting multicast data distribution tree should almost have the minimum weight. The core broadcasting used in MCEDAR does not compute this minimum weight tree; it simply core broadcasts the packets to the nearby multicast receivers.

But what price are we paying for this? Since the $Connect$ messages are piggybacked with the multicast data packets, the only extra messages are the $Pass$ messages, which are atmost $|Q| - 2$ in number in a given local third neighborhood. The number of such

messages throughout the network are hence bounded by the number of primary routers. The nodes that are primary routers is a subset of all the core nodes that form the MDS. Since the number of nodes in the MDS is much fewer than the actual number of vertices, the number of such $Pass$ messages, though in $O(|V|)$, is much fewer than $|V|$. Each multicast data packet increases in size by $O(|Q|)$, to hold the table, plus by the space required to hold the sender velocity, direction and location information. Since $|Q|$, the number of primary routers in the third neighborhood cannot be unreasonably large, this overhead is in fact trivial. Since both MCEDAR and NAMO NAMAHA have the core nodes calculate the single source shortest path using Dijkstra's algorithm for other core nodes, the time complexity is same for both the forwarding protocols, which is $O(|V|^2)$ at the most, where u is the node that initiated the Steiner tree computation.

Finally, the RTS/CTS mechanism in the core broadcast ensure that the number of times the multicast message copies are sent over the links is in the order of $|V|$. In NAMO NAMAHA too, the number of times, the multicast message copies are sent over the links is bounded by $|V|$. This is because the multicast data packet is sent only once over any given Steiner tree link. The number of multicast data packet copies sent is exactly equal to the number of $Connect$ messages, since with every multicast message copy, we have the $Connect$ message piggybacked. We had shown in Section 5.6.1 that the number of $Connect$ messages in a given local third neighborhood is $|V_T| - 1$. So throughout the network, NAMO NAMAHA causes $O(|V|)$ complexity on the number of transmissions made for distributing multicast messages.

The comparison is summarized in Figure 5.5. The price we pay for the computation of minimum weight tree and the reduction messages in join and forwarding protocol is a minimal increase in the multicast data packet and extra $Pass$ messages whose number is well below $|V|$. With the Leave protocol being the same for both MCEDAR and NAMO

NAMAHA, NAMO NAMAHA is clearly superior to MCEDAR.

Core computation algorithm:
- -Time Complexity
  - -O(|N(u)||lg(|N(u)|)), for the lexicographic sort. Since |N(u)| < k, we perform radix sort which runs in O(2(|N(u| + k))) = O(k); constant time.

Core Broadcast algorithm
- -Time complexity
  - -O(|V|^2), to calculate the best path to the core nodes using Dijkstra's single source shortest path algorithm
- -Message complexity
  - -O(|V|), using the RTS/CTS caching mechanism
- -Memory complexity
  - -not worse than O(|mV|) = O(|V|), if we assume m < k [See Figure 5.2]

-Join protocol
- -Message complexity
  - -O(|V|), since it uses Core Broadcast algorithm

-Forwarding protocol
- -Time complexity
  - -O(|V^2|), to calculate the best path to the core nodes using Dijkstra's single source shortest path algorithm. In addition to this, the core nodes will have to calculate the single source shortest path for the non core nodes, which bears the same complexity
- -Extra message and packet size overhead
  - -(|Q| - 2) Pass messages, where |Q| is the number of primary routers in the third neighborhood region; which translates to less than |V| 'Pass' messages all over the network
  - -Multicast packets will increase in size by O(|Q|), since they also carry 'Connect' message, the table, velocity, direction and location information.
- -Number of transmissions of a single multicast data packet
  - -O(|V|), since the number of multicast data packet sent is equal to the number of 'Connect' messages which is piggybacked with each multicast data packet

-Leave protocol
- -Time and memory complexity
  - -Constant, since it involves only one message from the subscriber that wishes to unsubscribe to its dominator

Figure 5.4: The time and memory complexities for all the algorithms and data structures in NAMO NAMAHA

Underlying Unicast protocol

- Both NAMO NAMAHA and MCEDAR use CEDAR as the underlying unicast protocol. MCEDAR only uses the unicast and core broadcast capabilities of CEDAR. NAMO NAMAHA also exploits the local knowledge possessed by the CEDAR core nodes.

Join Protocol

- MCEDAR join protocol has acknowledgements and is complicated. No acknowledgements are used in NAMO NAMAHA making the join protocol much simpler. The join protocol is also efficient since the unicast path between the subscriber and the sender is approximately the best path for the sender to send multicast data packets to the subscriber.
- The periodic join requests issued by MCEDAR are propagated all over the network. The join requests issued in NAMO NAMAHA, and the core broadcast for periodic unicast messages are restricted only to the 'request zone' which is a small subgraph of the network graph.

Path redundancy

- Owing to the redundancy offered by the secondary routers, a path between some arbitrary subscriber and the multicast sender almost always exist in NAMO NAMAHA even if some primary routers responsible for data routing moves away. Such a guarantee cannot be given in MCEDAR.

Forwarding Protocol

- In NAMO NAMAHA, the resulting multicast data distribution tree has almost the least possible cost (eg. delay). MCEDAR does not minimize the cost.
- In NAMO NAMAHA, multicast messages are sent to the regions where they are absolutely necessary for the subscribers to get the data. If the 'child' and 'parent' set is not up to date in MCEDAR, the core broadcast propagates all over the network in the hope of finding a subscriber.
- Both have a time complexity of O(|Q|^2), where |Q| is the number of core nodes in the third neighborhood of a given core node. The algorithms in both the forwarding protocols are no costler than the Dijkstra's single source shortest path algorithm, which runs in O(|Q|^2).

NAMO NAMAHA Overhead

- Inclusion of location, velocity and direction information for the multicast sender in each multicast data packet. This is a trivial overhead.
- Inclusion of primary router and secondary router list in the beacon messages. Since the delta information for the table is transmitted, the overhead is tolerable. We assume that the number of active multicast groups at any given time is in the order of tens.
- Maintaining the primary router and secondary router information for all the core nodes in the region up to the third neighborhood in the table. This is a constant overhead, since the number of core nodes in the third neighborhood of a given core node is limited in tens.
- Piggybacked 'Connect' message in each multicast data packet. This should not take more than a bit in the data packet.
- Extra 'Pass' messages for each multicast data packet sent from the multicast sender. This number is not more than the number of primary routers and hence much lesser than the number of core nodes. Thus the extra pass messages, though in O(|V|), is actually very much smaller compared to the actual number of vertices in the graph.

Leave protocol

- Same for both NAMO NAMAHA and MCEDAR.

Figure 5.5: Comparsion of NAMO NAMAHA with MCEDAR - Summary

Chapter 6

Conclusion and Future Work

This chapter summarizes the thesis and gives some research pointers for future work.

6.1 Conclusion

The thesis started by introducing various multicast protocols for ad hoc networks and then justifying the need for NAMO NAMAHA in this background. Network layer multicast protocols, which maintain session state, simply does not scale for large number of nodes and overlay multicast protocols are not as efficient. Network layer is at any time better informed about the topology than the application layer, where overlays are usually constructed. The idea that if the overlay multicast distribution graph construction algorithms are aided by the network layer, efficient overlays can be constructed, formed the cornerstone of NAMO NAMAHA.

Having detailed the NAMO NAMAHA protocol, we see that overlay multicasting in ad hoc network has similarities to query routing in peer to peer networks [RF02]. The overlay network formed by the $super\ peers$ in peer to peer network such as $Gnutella$ resembles the overlay network formed by the core nodes in the NAMO NAMAHA protocol. A new

node connects to a super peer and this super peer serves the new node. In effect, the super peers shield the new nodes from some traffic. There are protocols for node joins and leaves in peer to peer network as well. But the difference between query routing in peer to peer networks and data routing in NAMO NAMAHA is that the query routing can be simply broadcast in a hope of finding nodes that can answer the query. On the other hand, multicast data routing in NAMO NAMAHA, or for that matter any multicasting approach in ad hoc networks, involve routing the data to a specific set of receivers which may dynamically vary. [SSB99a] shows by simulation that the flooding approach in ad hoc network is highly detrimental to the network bandwidth. A simple flooding approach for multicasting does not scale for large number of nodes. Flooding approach for data multicasting is usually not taken in ad hoc networks unless the multicast group is static and very small. The Gnutella protocol uses broadcasts to query for files. Despite popular rumor that this does not necessarily limit the overall size of the network, it is clearly desirable to eliminate broadcast queries. Reducing query traffic increases scalability and leaves more bandwidth for uploads and downloads. Schemes like $Freenet$ and $Prinkey$ have been proposed to eliminate broadcasts. In summary, though some of the design issues are same in data multicasting and query routing, multicasting is generally a harder problem than query routing. In multicasting, data needs to be sent to a specific set of receivers that dynamically vary. Query routing terminates when some of the nodes satisfactorily answer the query posed by a given node. But in both the cases, the network needs to constantly evolve to increase the efficiency for the next on the fly data multicasting or query routing. This could be in the form of session state exchange, caching, etc., which are common to both.

This thesis first looks at the small group multicast presented in [CN02]. NAMO NAMAHA removed the non-scalable assumptions made by this protocol. It also added the linear time

join and leave protocols for dynamic multicast group membership which was not possible with [CN02]. The result from [CN02] that LGS algorithm performs better than the LGK algorithm for data distribution tree construction, when the minimization parameter information is up to date, was used in NAMO NAMAHA. The minimization parameter for the LGS algorithm in [CN02] was distance, which was an approximation for the number of network hops. In NAMO NAMAHA delay could also be the minimization parameter.

Next, we shifted our focus to the selection of the unicast protocol to function at the network layer. We introduced CEDAR unicast routing protocol [SSB99a] and justified its choice as an underlay for NAMO NAMAHA. The information at the network layer which is useful, local, almost invariant, and that which would incur constant memory overhead at the nodes were identified to aid the overlay data distribution graph. Core nodes as calculated by the CEDAR unicast routing protocol were selected to be the nodes that would provide such an information. At this point we introduced the algorithms used by the NAMO NAMAHA protocol. The Join protocol had two parts to it: Request Zone calculation and Unicast Trap algorithm. The Request Zone calculation algorithm ensures that the join discovery messages are not propagated all over the network. The Unicast Trap algorithm calculates the unicast path from subscriber to sender. The core nodes in such a unicast path are the primary routers and approximately the same path is used to send the data from the multicast sender to the subscriber. Furthermore the secondary routers generated in this process ensures that a path exists between the sender and subscriber almost all the time with high probability. The primary router paths are refreshed occasionally by requiring the subscriber to send unicast messages to the sender. The forwarding protocol builds Steiner trees in the third neighborhood of the core nodes over the subgraph of the core nodes comprising of the subscriber, sender, primary routers and the secondary router. The $(CHINS)_T$ algorithm which implements the well known Takahashi Matsuyama heuristic

was adopted in the forwarding protocol for NAMO NAMAHA. The connection of such Steiner trees forms the multicast data distribution tree, which is calculated on the fly.

Finally, after offering the proof of correctness for the protocol, we show that all the algorithms involved in the NAMO NAMAHA protocol are either constant time or linear in terms of number of nodes or number of edges. NAMO NAMAHA scales to very large networks. However, we have to assume that the number of nodes surrounding a give node are limited. In this we are justified since practically in a given collaborative ad hoc network, we will not have too many nodes surrounding a given node. We compared our work with the MCEDAR protocol in terms of the cost of the multicast data distribution trees, the number of messages exchanged in building them and the time and memory complexity of the algorithms involved. We chose MCEDAR since other multicast protocols for ad hoc networks are either network based which does not scale for large nubmer of nodes, or function as overlays designed only for small groups. When compared to MCEDAR, NAMO NAMAHA has a simpler $join$ protocol that does not make use of acknowledgements. Unlike MCEDAR, the sender discovery messages are not propagated all over the network; they are restricted to a region where it is absolutely necessary. In NAMO NAMAHA, at any given time, a path exists between any multicast subscriber and the sender (which is approximately the best path possible) with very high probability. Such a guarantee is not possible with MCEDAR. Furthermore, because of the incremental Steiner tree construction, the resulting multicast data distribution tree has nearly the least total cost. Cost is not minimized in MCEDAR. These advantages are obtained in NAMO NAMAHA just by using minimal extra messages during tree construction, whose number is well below the number of nodes in the multicast group and hence tolerable. Dijkstra's Single Source Shortest Path algorithm is the costliest algorithm in both the protocols. In both the cases, shortest paths are calculated by a core node to the core nodes in its local third neighborhood.

6.2 Future Work

We intend to pursue the following research directions beyond this thesis:

- The forwarding protocol of NAMO NAMAHA can be formulated as Steiner tree construction with incomplete global knowledge in a graph [KM00]. Since Steiner tree problem with complete global knowledge (which is NP Complete) is a special case, this problem is also NP Complete. The distributed algorithms for the Steiner tree implementation assume that the nodes participating in the Steiner tree computation know the best path to all other participating nodes. If we are to retain this assumption, then we have to restrict the Steiner tree construction to a very small local region. In NAMO NAMAHA we exactly did that. We restricted the Steiner tree construction to the third neighborhood region of the core nodes. In this small region too, we could not assume that all the nodes have the best path information to the other nodes since two third neighbors of a given core node need not be third neighbors of each other (see Figure 4.7). What exactly needs to be done to restrict Steiner tree computation to local regions and at the same time ensure that all the participating nodes know the best path to each other? What is the best way to connect the Steiner trees thus formed? We intend to answer these questions and give a rigorous graph theoretical model for this problem. We intend to come up with better approximation algorithms that can perform in a distributed fashion. The $(CHINS)_T$ had all sequential steps. Is it possible to come up with a parallel distributed algorithm? All these questions form a very interesting research topic to pursue.
- What if we choose fourth neighborhood instead of third neighborhood for the

core nodes? Which choice will give the most optimum performance? What is the right number? 3,4,5...? We intend to answer this question.

- There are various timers involved in NAMO NAMAHA. The most important one is the $unicastMessageTimer$ (how frequent should a subscriber send the 'dummy' unicast message to the sender?). Another timer is the $secondaryRouterLifeTimer$. These two timers are strongly related. The relation between these two timers also affects the connectivity as seen by the subscriber. If the 'dummy' unicast message is not sent for a long time, both the primary routers and the secondary routers may cease to exist and there may not be a route for the sender to send data to the subscriber. We intend to study of the interdependency between the timers involved in NAMO NAMAHA and their relation to the network mobility randomness.
- Caching systems at the core nodes will help speed up on the fly data distribution tree construction. In our work, distribution trees are constructed on the fly whenever the sender needs to send the data. A study on caching systems at the core node is another interesting area that we plan to explore.

List of References

[BDSZ94] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang. MACAW: A Media Access Protocol for Wireless LANs. In *Proceedings, 1994 SIGCOMM Conference*, pages 212–225, London, UK, August 31st - September 2nd 1994.

[CGZ98] Ching-Chuan Chiang, Mario Gerla, and Lixia Zhang. Forwarding Group Multicast Protocol (FGMP) for Multihop, Mobile Wireless Networks. *Cluster Computing*, 1(2):187–196, 1998.

[CLRS01] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[CN02] Kai Chen and Klara Nahrstedt. Effective Location-Guided Tree Construction Algorithms for Small Group Multicast in MANET. In *Proceedings of IEEE INFOCOM*, pages 1180–89, 2002.

[dMCGA03] Carlos de Morais Cordeiro, Hrishikesh Gossain, and Dharma Agrawal. Multicast over Wireless Mobile Ad Hoc Networks: Present and Future Directions. *IEEE Network*, 17(1):52–59, 2003.

[GCA02] H. Gossain, C. Cordeiro, and D. Agrawal. Multicast: Wired to Wireless. *IEEE Communications Magazine*, 20(1):99–116, June 2002.

[GK98] S. Guha and S. Khuller. Approximation Algorithms for Connected Dominating Sets. *ALGORTHMICA: Algorithmica*, 20, 1998.

[HRW92] Frank K. Hwang, Dana S. Richards, and Pawel Winter. *The Steiner Tree Problem*. Number 53 in Annals of Discrete Mathematics. Elsevier Science Publishers B. V., Amsterdam, 1992.

[JC98] L. Ji and M. Scott Corson. A Lightweight Adaptive Multicasting Algorithm. In *GLOBECOM*, pages 1036–42, 1998.

[JC01] Lusheng Ji and M. Scott Corson. Differential Destination Multicast - a MANET Multicast Routing Protocol for Small Groups. In *Proceedings of IEEE INFOCOM*, pages 1192–1202, 2001.

[KM00] D. R. Karger and M. Minkoff. Building Steiner Trees with Incomplete Global Knowledge. In IEEE, editor, *41st Annual Symposium on Foundations of Computer Science: proceedings: 12–14 November, 2000, Redondo Beach, California*, pages 613–623. IEEE Computer Society Press, 2000.

[KV98] Young-Bae Ko and Nitin H. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of Mobile Computing and Networking*, pages 66–75, 1998.

[LSG02] Sung Ju Lee, William Su, and Mario Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networking Application*, 7(6):441–453, 2002.

[LTM99] M. Liu, R. Talpade, and A. McAuley. AMRoute: Adhoc Multicast Routing Protocol, 1999.

[MGLA01] Ewerton L. Madruga and J. J. Garcia-Luna-Aceves. Scalable Multicasting: The Core-Assisted Mesh Protocol. *ACM/Baltzer Mobile Networks and Applications, Special Issue on Management of Mobility*, 6(2):151–165, April 2001.

[NRK01] Roman Novak, Joze Rugelj, and Gorazd Kandus. A note on distributed multicast routing in point-to-point networks. *Computers and Operations Research*, 28:1149–1164, 2001.

[RAMM03] F. De Rango, A.Iera, A. Molinaro, and S. Marano. A Modified Location-Aided Routing Protocol for the Reduction of Control Overhead in Ad-hoc Wireless Networks. In *10th International Conference on Telecommunications*, volume 2, pages 1033–1037, 2003.

[RF02] Matei Ripeanu and Ian Foster. Mapping the gnutella network: Macroscopic properties of large-scale peer-to-peer systems. *Lecture Notes in Computer Science*, 2429:85–92, 2002.

[RP99] Elizabeth M. Royer and Charles E. Perkins. Multicast Operation of the Ad-hoc On-Demand Distance Vector Routing Protocol. In *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking*, pages 15–19, august 1999.

[SSB99a] Prasun Sinha, Raghupathy Sivakumar, and Vaduvur Bharghavan. CEDAR: a Core-Extraction Distributed Ad Hoc Routing Algorithm. In *Proceedings of IEEE INFOCOM (1)*, pages 202–209, 1999.

[SSB99b] Prasun Sinha, Raghupathy Sivakumar, and Vaduvur Bharghavan. MCEDAR: Multicast Core Extraction Distributed Ad Hoc Routing. In *Proceedings of*

the Wireless Communications and Networking Conference, volume 3, pages 1313–1317, 1999.

[TM80] H. Takahashi and A. Matsuyama. An Approximate Solution for the Steiner Tree Problem in Graphs. *Mathmatica Japonica*, pages 573–577, 1980.

[WT99] C.W. Wu and Y.C. Tay. AMRIS: A Multicast Protocol for Ad hoc Wireless Networks. In *Proceedings of IEEE MILCOM'99*, volume 1, pages 25–29, November 1999.