CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm

Prasun Sinha Raghupathy Sivakumar Vaduvur Bharghavan
Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Email:{prasun, sivakumr, bharghav}@timely.crhc.uiuc.edu

Abstract

In this paper, we present CEDAR, a *Core-Extraction Distributed Ad hoc Routing* algorithm for QoS routing in ad hoc network environments. CEDAR has three key components: (a) the establishment and maintenance of a self-organizing routing infrastructure called the *core* for performing route computations, (b) the propagation of the link-state of stable high-bandwidth links in the core through *increase/decrease waves*, and (c) a QoS route computation algorithm that is executed at the core nodes using only locally available state.

Our performance evaluations show that CEDAR is a robust and adaptive QoS routing algorithm that reacts quickly and effectively to the dynamics of the network while still approximating link-state performance for stable networks.

1 Introduction

An ad hoc network is a dynamic multi-hop wireless network that is established by a group of mobile hosts on a shared wireless channel by virtue of their proximity to each other. Since wireless transmissions are locally broadcast in the region of the transmitting host, hosts that are in close proximity can hear each other and are said to be neighbors. The transitive closure of the neighborhood of all the hosts in the set of mobile hosts under consideration forms an ad hoc network. Thus, each host is potentially a router and it is possible to dynamically establish routes by chaining together a sequence of neighboring hosts from a source to a destination in the ad hoc network. Such networks find applicability in military environments, wherein a platoon of soldiers or a fleet of ships may establish an ad hoc network in the region of their deployment. Military network environments typically require quality of service for their mission-critical applications. Hence, the focus of this paper is to provide *quality of service routing in ad hoc networks.*

In particular, we seek to compute unicast routes that satisfy a minimum bandwidth requirement from the source to the destination. Of course, since the network is highly dynamic, and transmissions are susceptible to fades, interference, and collisions from hidden/exposed stations, we cannot provide bandwidth guarantees for the computed routes. Rather, our goal is to provide routes that are highly likely to satisfy the bandwidth requirement of a route [1].

Given the nature of the network and the requirements of the applications, the following are the key goals of our routing algorithm. First, the algorithm must be highly robust, and should degrade gracefully upon sudden link or host failures, or changes in the network topology. Second, the routing algorithm must generate an admissible route (i.e. a route that satisfies the bandwidth requirement) with high probability so long as such a route is available. Third, only some of the hosts in the network should be involved in route computation because this involves monitoring and reacting to changes in the network topology. However, every host should have quick access to routes when it seeks to establish connections with other hosts. Fourth, the routing algorithm at the selected subset of hosts which perform route computation must require only local computation and involve mostly local state.

Finally, each host must only maintain routes for those hosts that it cares about, i.e. those hosts with whom it communicates, and must not have to update state often even if the network topology is highly dynamic so long as the routes it cares about are stable.

In order to achieve the above goals, we propose a *Core-Extraction Distributed Ad hoc Routing* (CEDAR) algorithm for QoS routing in ad hoc networks. CEDAR has three key components: (a) the establishment and maintenance of the *core* of the network for performing the route computations, (b) propagation and use of bandwidth and stability information of links in the ad hoc network, and (c) the QoS route computation algorithm. Briefly, CEDAR dynamically establishes a *core* of the network, and then incrementally propagates link state of stable high bandwidth links to the nodes of the *core*. Route computation is on-demand, and is performed by *core* hosts using local state only. We propose CEDAR as a QoS routing algorithm for small to medium size ad hoc networks consisting of tens to hundreds of nodes. The following is a brief description of the three key components of CEDAR.

- *Core extraction*: We dynamically extract the *core* of the network by approximating a minimum dominating set of the ad hoc network using only local computation and local state. Each host in the *core* then establishes a virtual link (via a tunnel) to nearby *core* hosts. Each *core* host maintains the local topology of the hosts in its domain, and also performs route computation on behalf of these hosts. The initial *core* computation and the *core* management upon change in the network topology are purely local computations, thus the *core* adapts efficiently to the dynamics of the network.

- *Link state propagation*: QoS routing in CEDAR is achieved by propagating the bandwidth availability information of stable links in the *core* graph. The basic idea is that the information about stable high-bandwidth links can be made known to nodes far away in the network, while information about dynamic links or low bandwidth links should remain local. The propagation of link-state is achieved through slow-moving 'increase' waves (which denote increase of bandwidth) and fast-moving 'decrease' waves (which denote decrease of bandwidth). The key questions to answer in link state propagation are: when to initiate increase/decrease waves, how far can a wave propagate, and how fast can a wave propagate.

- *Route computation*: Route computation first establishes a *core* path from the domain of the source to the domain of the destination. This initial phase involves probing on the *core* graph, and the resultant *core* path is cached for future use. 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 (which then becomes the source for the next iteration) which can satisfy the requested bandwidth, using only local information. Effectively, the computed route is a shortest-widest[1] furthest path using the *core* path as the guideline.

Robustness, rather than optimality, is the primary concern of CEDAR. Core host computations are purely local, and the algorithm to re-establish the *core* upon change in the network topology is also local. Each *core* host knows only about nearby *core* hosts, and has no idea about the entire *core* graph. Unlike the spine architecture [2, 3], there is no notion of a virtual backbone tree in the *core* graph. *Core* paths are dynamically established for connection requests, and route computation is only performed when a route is requested (unlike distance vector or link state algorithms). The only information that is kept in a *core* host is the local topology, nearby *core* hosts, and information obtained about remote links from increase/decrease waves. The nature of propagation of the increase/decrease waves ensures that unstable links and low bandwidth links do not cause topology updates further away from their

[1] A shortest widest path is the maximum bandwidth path. If there are several such paths, it is the one with the least number of hops.

locations. Thus, CEDAR adapts quickly to dynamics in the topology. At the same time, it computes good routes and satisfies bandwidth requirements of connection requests with high probability so long as such routes exist. Of course, CEDAR does not compute optimal routes because of the minimalist approach to state management, but we believe that our trade-off of robustness and adaptation for optimality is well justified in ad hoc networks.

The rest of this paper is organized as follows. Section 2 describes the network model, terminology, and the goals of CEDAR. Section 3 describes the computation and dynamic management of the *core* of the network. Section 4 describes the link state propagation through the *core* using increase and decrease waves. Section 5 describes the route computation algorithm of CEDAR, and puts together the algorithms described in the previous sections. Section 6 analyzes the performance of CEDAR through simulations. Section 7 compares CEDAR to related work, and Section 8 concludes the paper.

2 Network Model and Goals

In this section, we first describe the network model, then the terminology used in this paper, and finally the goals of CEDAR.

2.1 Network Model

We assume that all the hosts communicate on the same shared wireless channel. For FHSS, this implies that all hosts have the same frequency hopping pattern, while for DSSS, this implies that all hosts have the same pseudorandom sequence. For an ad hoc network, this is a valid assumption [4]. We assume that each transmitter has a fixed transmission range, and that neighborhood is a commutative property (i.e. if A can hear B, then B can hear A). Because of the local nature of transmissions, hidden and exposed stations[2] abound in an ad hoc network. We assume the use of a CSMA/CA like algorithm such as MACAW [5] or FAMA [6] for reliable unicast communication, and for solving the problem of hidden/exposed stations. Essentially, data transmission is preceded by a control packet handoff, and the sequence of packets exchanged in a communication is the following: RTS (from sender to receiver) - CTS (from receiver to sender) - Data (from sender to receiver) - ACK (from receiver to sender). The RTS and CTS packets avoid collisions from the exposed stations and the hidden stations respectively. *Local broadcasts are not guaranteed to be reliable* (because it is unreasonable to expect a CTS from every receiver before commencing data transmission), and are typically quite unreliable due to the presence of hidden and exposed stations.

We assume small to medium networks ranging from tens to hundreds of hosts. For larger networks, we propose a clustering algorithm in a related work [2] and apply CEDAR hierarchically within each cluster, for a cluster of clusters, etc. We assume that mobility and extended fades are the main causes of link failures and topology changes. We assume that the change in network topology is frequent, but not frequent enough to render any sort of route computation useless. Note that we only care about the relative mobility of the hosts, not the absolute mobility of the hosts. In particular, even if a platoon of soldiers is moving, the ad hoc network would be considered to be stable so long as the neighborhood of each soldier does not change.

We assume that the MAC/link layer can estimate the available link bandwidth. Because all the hosts in a region share the same channel, each host must share the link bandwidth with the hosts in its second neighborhood [5]. In related work on providing QoS in wireless channels, we have provided a mechanism for each host to fairly access a shared channel, and claim at least B/N bandwidth, where B is the effective channel bandwidth and N is the number of hosts locally contending for the bandwidth

[2] A hidden station is a host that is within the range of the receiver but not the transmitter, while an exposed station is within the range of the transmitter but not the receiver.

[7]. While details of bandwidth sharing and estimation are beyond the scope of this paper, we assume that each host can estimate the available bandwidth using some link-level mechanisms.

We assume a close coordination between the MAC layer and the routing layer. In particular, we use the reception of RTS and CTS control messages at the MAC layer in order to improve the behavior of the routing layer, as explained in Section 3.

Finally, bandwidth is the QoS parameter of interest in this paper. When an application requests a connection, it specifies the required bandwidth for the connection. The goal of CEDAR is then to find a short stable route that can satisfy the bandwidth requirement of the connection.

2.2 Graph Terminology

We represent the ad hoc network by means of an undirected graph $G = (V, E)$, where V is the set of nodes in the graph (hosts in the network), and E is the set of edges in the graph (links in the network). The i^{th} *open 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} *closed neighborhood* $N^i[x]$ of node x is $N(i) \cup \{x\}$.

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. We use the term *tunnel* interchangeably with *virtual link* in our discussions.

Given an MDS V_C of graph G, we define a *core* of the graph $C = (V_C, E_C)$, where $E_C = \{[u, v] \mid u \in V_C, v \in V_C, u \in N^3(v)\}$. 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 *nearby* nodes.

For a connected graph G, consider any dominating set S. If the diameter of G is greater than 2, then 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 the *core*, if G is connected, then a *core* C of G must also be connected (via virtual links).

2.3 Goals of CEDAR

Ad hoc routing typically has the following goals. (a) Route computation must be distributed, because centralized routing in a dynamic network is impossible even for fairly small networks. (b) Route computation should not involve the maintenance of global state, or even significant amounts of volatile non-local state. In particular, link state routing is not feasible because of the enormous state propagation overhead when the network topology changes. (c) As few nodes as possible must be involved in state propagation and route computation, since this involves monitoring and updating at least some state in the network. On the other hand, every host must have quick access to routes on-demand. (d) Each node must only care about the routes corresponding to its destination, and must not be involved in frequent topology updates for parts of the network to which it has no traffic. (e) Stale routes must be either avoided or detected and eliminated quickly. (f) Broadcasts must be avoided as far as possible because broadcasts are highly unreliable in ad hoc networks. (g) If the topology stabilizes, then routes must converge to the optimal routes, and (h) It is desirable to have a backup route when the primary route has become stale and is being recomputed.

QoS routing for ad hoc networks is relatively unchartered territory. We have the following goals for QoS routing in ad hoc networks. (a) Applications provide a minimum bandwidth requirement for a connection, and the routing algorithm must efficiently compute a route that can satisfy the bandwidth requirement with high probability. (b) If there exists a route that can satisfy the bandwidth requirement, then the routing algorithm must compute an admissible route with high probability. (c) The amount of state propagation and topology update information must be kept to a minimum. In

particular, every change in available bandwidth should not result in updated state propagation. (d) Dynamic links (either unstable or low bandwidth links) must not cause state propagation throughout the network. Only stable high bandwidth link information must be propagated throughout the network. (e) As the network becomes stable, the routing algorithm should start providing near-optimal routes, and (f) The QoS route computation algorithm should be simple and robust. Robustness, rather than optimality, is the key requirement.

In summary, our goal is to compute good routes quickly, and react to the dynamics of the network with only small amounts of state propagation.

3 CEDAR Architecture and the Core

The QoS routing architecture in CEDAR has three key components: (a) the establishment of the *core* in the ad hoc network to manage topology information and perform route computation, (b) the propagation of the link-state of stable high bandwidth links in the *core* graph through increase and decrease waves, and (c) the route computation algorithm at *core* nodes using only local state. In this section, we first describe the overall CEDAR architecture and then focus on the establishment and maintenance of the *core*.

Briefly, we extract the *core* of the ad hoc network by approximating the minimum dominating set (MDS) of the ad hoc network. The nodes in the MDS comprise the *core* nodes of the network. Each *core* node establishes a unicast virtual link (via a tunnel) with other *core* nodes a distance of 3 or less away from it in the ad hoc network. The *core* nodes then collect local topology information, and perform routing for the nodes in their domain (or immediate neighborhood). Each node that is not in the *core* chooses a *core* neighbor as its dominator, i.e. the node which performs route computations on its behalf. The *core* is merely an infrastructure for facilitating route computation, and is itself independent of the routing algorithm. In particular, it is possible to use any of the well known ad hoc routing algorithms such as DSR [8], LMR [9], TORA [10], DSDV [11], etc. in the *core* graph.

While it is possible to execute ad hoc routing algorithms using only local topology information at the *core* nodes, CEDAR propagates the link-state corresponding to stable high-bandwidth links among the *core* nodes. The motivation for this approach is that the link-state of dynamic and low-bandwidth links should not be made known to *core* nodes far away (since the link-state is likely to change frequently), but the link-state of stable high-bandwidth links can be used by remote *core* nodes to compute better routes for connections passing through them. As we describe in Section 4, we propagate link-state via the use of two 'waves': a slow moving increase wave that denotes an increase in the available bandwidth of a link, and a fast moving decrease wave that denotes a decrease in the available bandwidth of a link. For unstable links (which come up and go down frequently), the decrease wave upon link failure will quickly overtake and kill the increase wave that corresponds to a previous link establishment, thereby ensuring that the link state for unstable links remains within the locality of the link. On the other hand, the increase wave of a stable link slowly propagates throughout the *core* graph, thereby enabling remote *core* nodes to take advantage of this link. We constrain the maximum distance to which the increase wave of a link can propagate depending on its available bandwidth. Thus, only stable high-bandwidth links are made known to *core* nodes far away. While the details of the link state propagation algorithm are deferred to Section 4, it is important to note that the state available at a *core* node comprises the up-to-date local topology, and possibly outdated information about stable high-bandwidth links far away. This is the state that is used by a *core* node for route computation.

The route computation in CEDAR follows two steps. First, we establish a *core* path from the dominator of the source node to the dominator of the destination node if such a path is not already cached. The basic idea of establishing a *core* path is to provide the directionality from the source

to the destination. Once the *core* path is established, the dominator of the source node tries to find the shortest admissible path to the domain of the furthest possible node in the *core* path using its local state. The route computation is then iteratively carried out at the furthest reachable node in the *core* path. Each *core* node only uses local state, and an admissible route is established in a single pass (and carried in the route request packet). The link-state information that is obtained about remote links using the increase/decrease waves is very useful in terms of both setting up shortest admissible paths, as well as reducing the number of connection rejections. Note that for a stable ad hoc network, CEDAR converges to an optimal link-state routing algorithm, where only the core nodes are involved in route computation. While it is well known that link-state algorithms do not scale well for dynamic networks, CEDAR provides an approximation of a link-state algorithm for stable networks and provides a robust low overhead algorithm to adapt to the dynamics of the network while still computing good routes with low overhead for small to medium size networks. For large networks, our ongoing work proposes a hierarchical clustering of the network and the application of CEDAR at each level of the hierarchy. Finally, CEDAR also provides for effective route recomputation mechanisms for route changes in ongoing connections, as described in Section 5.

In the following subsections, we first describe the motivation for choosing a *core*-based routing architecture, then describe a low overhead mechanism to generate and maintain the *core* of the network, and finally describe an efficient mechanism to accomplish a '*core* broadcast' using unicast transmissions. The *core* broadcast is used both for propagation of increase/decrease waves, and for the establishment of the *core* path in the route computation phase.

3.1 Rationale for a Core-based Architecture in CEDAR

Many contemporary proposals for ad hoc networking require every node in the ad hoc network to perform route computations and topology management [8, 9, 11, 12]. However, CEDAR uses a *core*-based infrastructure for QoS routing due to two compelling reasons.

1. QoS route computation involves maintaining local and some non-local link-state, and monitoring and reacting to some topology changes. Clearly, it is beneficial to have as few nodes in the network performing state management and route computation as possible.

2. Local broadcasts are highly unreliable in ad hoc networks due to the abundance of hidden and exposed stations. Topology information propagation [11, 12] and route probes [8, 9] are inevitable in order to establish routes and will, of necessity, need to be broadcast if every node performs route computation. While the adverse effects of unreliable broadcasts are typically not considered in most of the related work on ad hoc routing, we have observed that flooding in ad hoc networks is highly lossy [13]. On the other hand, if only a *core* subset of nodes in the ad hoc network perform route computations, it is possible to set up reliable unicast channels between nearby *core* nodes and accomplish both the topology updates and route probes much more effectively.

The issues with having only a *core* subset of nodes performing route computations are threefold. First, nodes in the ad hoc network that do not perform route computation must have easy access to a nearby *core* node so that they can quickly request routes to be setup. Second, the establishment of the *core* must be a purely local computation. In particular, no *core* node must need to know the topology of the entire *core* graph. Third, a change in the network topology may cause a recomputation of the *core* graph. Recomputation of the *core* graph must only occur in the locality of the topology change, and must not involve a global recomputation of the *core* graph. On the other hand, the locally recomputed *core* graph must still only comprise of a small number of *core* nodes - otherwise the benefit of restricting route computation to a small *core* graph is lost.

We have previously developed the *spine* architecture for ad hoc routing, in which we establish a tree among the nodes of an approximate minimum connected dominating set of the ad hoc network [2]. While the spine architecture provides some of the benefits of the *core* architecture, it does not handle the last two issues mentioned above. In particular, the spine computation is a global algorithm, each spine node is required to know the entire spine topology, and disconnection of the spine due to topology changes can result in a global recomputation in the worst case. Unlike the spine architecture, the *core* architecture requires purely local computations and local state to generate and maintain the *core*, each *core* node only has local information about the *core*, and there is no explicit tree structure that is established in the *core*. We believe that the *core* provides the benefits of the spine architecture without incurring the high maintenance overhead.

3.2 Generation and Maintenance of the Core in CEDAR

Ideally, the *core* comprises of the nodes in a minimum dominating set V_C of the ad hoc network $G = (V, E)$. However, finding the MDS is a NP-hard problem that is also hard to approximate [14]. The best known distributed algorithm for MDS approximation 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). While we can use the greedy algorithm to generate the best known approximation for the MDS, we have 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 open 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 which is performed periodically, works as follows at node u.

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

2. 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: $< u, \{(w, dom(w)) \mid \forall w \in N^1(u)\} >$. 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 closed neighborhood. Ties are broken by node id.

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: $< u, DOM, 3, path\_traversed = null >$. 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. As shown in Section 2, 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*. Note from

steps 2 and 4 that over a period of time, the *core* graph prunes itself because nodes have a propensity to choose their *core* neighbor with the highest effective degree as their dominator.

Maintaining the *core* in the presence of network dynamics is very 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*.

3.3 Core Broadcast and its Application to CEDAR

As with most existing ad hoc networks, 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, our experience has shown otherwise [13]. In particular, we have observed that flooding probes, which causes repeated local broadcasts, is highly unreliable because of the abundance of hidden and exposed stations. Thus, we provide a unicast based mechanism for achieving a '*core* broadcast'. 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, our *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.

Recall that a virtual link is a unicast path of length 1, 2, or 3. Recall also, that CSMA/CA protocols use a RTS-CTS-Data-ACK handshake sequence to achieve reliable unicast packet transmission. Our 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, we assume that each node temporarily caches every RTS and CTS packet that it hears on the channel for *core* broadcast packets only. Each *core* broadcast message M that is transmitted to a *core* node i 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. Now suppose that u and v are a distance 2 apart, and the virtual channel $[u, v]$ passes 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. If u and v are a distance 3 apart, using the same argument we will have atmost one extra message. 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.

In the ad hoc network shown in Figure 1, when node 1 is the source of the core broadcast, 10 would not be sending a message to 11 as it would have heard a CTS from 11, when 11 was receiving the message from 3. Similarly, 8 would not be sending on the tunnel to 10, as 9 would have heard the CTS from 10, and hence, would send a NACK when 8 sends an RTS to 9. Also, on the tunnel from 6 to 3, the message would be sent to 5, but 5 would not be able to forward it any further because of 4 having heard CTS from 3, and hence, 5 receiving NACK from 4. Thus, the example illustrates that a duplicate message can be avoided on tunnels of length 1 and 2, but a duplicate message will travel one extra hop for tunnels of length 3.

Note that our *core* broadcast has the following features:

1. The *core* nodes do not explicitly maintain a source-based tree. However, the *core* broadcast dynamically (and implicitly) establishes a source-based tree, which is typically a breadth-first search tree for the source of the *core* broadcast.

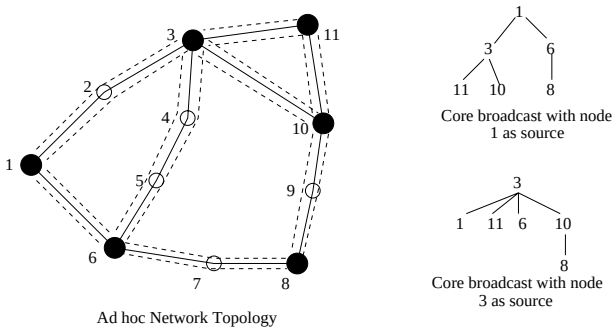

Figure 1: Example of a *core* broadcast. Nodes in black are *core* nodes. Solid lines denote links in the ad hoc network. Dotted pipes denote virtual links in the *core* graph.

2. The number of messages is $O(|V|)$ in the worst case, and $O(|V_C|)$ in the average case. In particular, the only case we transmit extra messages is when two nearby *core* nodes are a distance 3 apart.

3. Since the trees are not explicitly maintained, different messages may establish different trees. Likewise, changes in the network topology do not require any recomputation. However, the coordination of the MAC layer and the routing layer ensures that the *core* broadcast establishes a tree, and that a *core* node typically does not receive duplicates for a *core* broadcast.

While our approach for the *core* broadcast is very low overhead and adapts easily to topology changes, the RTS and CTS packets corresponding to a *core* broadcast need to be cached for some time after their reception.

Core broadcast finds applicability in two key aspects of CEDAR: discovery of the *core* path, and propagation of increase/decrease waves. The discovery of the *core* path is broadcast because the sender may not know the location of the receiver. It initiates a *core* broadcast to find the location of the receiver, and simultaneously, discover the *core* path.

4 QoS State Propagation in CEDAR

Section 3 described the *core* routing infrastructure of CEDAR. Since each *core* node uses only the locally cached state to compute the shortest-widest furthest path along the *core* path in the route computation phase, we now turn our attention to the nature of state that is stored in each *core* node. At one extreme is the minimalist approach of only storing local topology information at each *core* node. This approach results in a poor routing algorithm (i.e. the routing algorithm may fail to compute an admissible route even if such routes exist in the ad hoc network) but has a very low overhead for dynamic networks. At the other extreme is the maximalist approach of storing the entire link state of the ad hoc network at each *core* node. This approach computes optimal routes but incurs a high state management overhead for dynamic networks, and potentially computes stale routes based on out-of-date cached state when the network dynamics is high.

The problem with having only local state is that *core* nodes are unable to compute good routes in the absence of link-state information about stable high-bandwidth remote links, while the problem of having global state is that it is useless to maintain the link state corresponding to low-bandwidth and highly dynamic links that are far away because the cached state is likely to be stale anyway. Fundamentally, each *core* node needs to have the up-to-date state about its local topology, and also the link-state corresponding to relatively stable high-bandwidth links further away. Providing for such a link-state propagation mechanism ensures that CEDAR approaches the minimalist local state

algorithm in highly dynamic networks, and approaches the maximalist link-state algorithm in highly stable networks. We achieve the goal of having stability and bandwidth based link-state propagation using increase and decrease waves, as described in this section.

The basic idea of having our increase/decrease wave approach for updating link-state is the following. There are two types of waves: a slow-moving *increase* wave that denotes an increase of bandwidth on a link, and a fast-moving *decrease* wave that denotes a decrease of bandwidth on a link. For unstable links that come up and go down frequently, the fast moving decrease wave quickly overtakes and kills the slower moving increase wave, thus ensuring that the link-state corresponding to dynamic links is local. For stable links, the increase wave gradually propagates through the *core*. Each increase wave also has a maximum distance it is allowed to propagate. Low bandwidth increase waves are allowed only to travel a short distance, while high bandwidth increase waves are allowed to travel far into the network. Essentially, the goal is to propagate only stable high-bandwidth link-state throughout the *core*, and keep the low-bandwidth and unstable link-state local.

We first describe the mechanics of the increase and decrease waves, and then answer the three key questions pertaining to these waves: *when* should a wave be generated, *how fast* should a wave propagate, and *how far* should a wave propagate.

4.1 Increase and Decrease Waves

For every link $l = (a, b)$, the nodes a and b are responsible for monitoring the available bandwidth on l, and for notifying the respective dominators for initiating the increase and decrease waves, when the bandwidth changes by some threshold value. These waves are then propagated by the dominators (*core* nodes) to all other *core* nodes via *core* broadcasts. Each *core* node has two queues: the *ito-queue* that contains the pending *core* broadcast messages for increase waves, and the *dto-queue* that contains the pending *core* broadcast messages for decrease waves. For each link l about which a *core* node caches link-state, the *core* node contains the cached available bandwidth $b_{av}(l)$.

The following is the sequence of actions for an *increase wave*.

1. When a new link $l = (a, b)$ comes up, or when the available bandwidth $b(a, b)$ increases beyond a threshold value, then the two end-points of l inform their dominators for initiating a *core* broadcast for an increase wave: $ito(< a, b, dom(a), dom(b), b(a, b), ttl(b) >)$
   where *ito* (increase to) denotes the type of the wave, (a, b) identifies the link, $dom(a)$ denotes the dominator of a, $dom(b)$ denotes the dominator of b, $b(a, b)$ denotes the available bandwidth on the link, and $ttl(b)$ is a 'time-to-live' field that denotes the maximum distance to which this wave can be propagated as an increase wave. The *id*s of the dominators of the link end-points are required by the routing algorithm. $ttl(b)$ is an increasing function of the available bandwidth, as described in Section 4.3.

2. When a *core* node u receives an *ito* wave $ito(< a, b, dom(a), dom(b), b(a, b), ttl >)$,

   (a) if u has no state cached for (a, b),
   $b_{av}(a, b) \leftarrow b(a, b)$
   if $(ttl > 0)$, then add $ito(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the ito-queue.

   (b) else if u has cached state for (a, b) and $(ttl > 0)$,

      i. if $(b_{av}(a, b) < b(a, b))$
      $b_{av}(a, b) \leftarrow b(a, b)$
      delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
      add $ito(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the ito-queue.

      ii. else if $(b_{av}(a, b) > b(a, b))$,
      $b_{av}(a, b) \leftarrow b(a, b)$

delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
add $dto(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the dto-queue.

(c) else if u has cached state for (a, b) and $(ttl = 0)$,
$b_{av}(a, b) \leftarrow b(a, b)$
delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
add $dto(< a, b, dom(a), dom(b), 0, \infty >)$ to the dto-queue.

3. The ito-queue is flushed periodically, depending on the speed of propagation of the increase wave.

The following is the sequence of actions for a *decrease wave*.

1. When a link $l = (a, b)$ goes down, or when the available bandwidth $b(a, b)$ decreases beyond a threshold value, then the two end-points of l inform their dominators for initiating a *core* broadcast for a decrease wave: $dto(< a, b, dom(a), dom(b), b(a, b), ttl(b) >)$, where *dto* (decrease to) denotes the type of the wave, and the other parameters are as defined before.

2. When a *core* node u receives a *dto* wave $dto(< a, b, dom(a), dom(b), b(a, b), ttl >)$,

(a) if u has no state cached for (a, b) and $(b(a, b) = 0)$,
the wave is killed.

(b) else if u has no state cached for (a, b) and $(b(a, b) > 0)$,
$b_{av}(a, b) \leftarrow b(a, b)$
if $(ttl > 0)$, then add $ito(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the ito-queue.

(c) else if u has cached state for (a, b) and $(ttl > 0)$,

i. if $(b_{av}(a, b) < b(a, b))$,
$b_{av}(a, b) \leftarrow b(a, b)$
delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
add $ito(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the ito-queue.

ii. else if $(b_{av}(a, b) > b(a, b))$,
$b_{av}(a, b) \leftarrow b(a, b)$
delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
add $dto(< a, b, dom(a), dom(b), b(a, b), ttl - 1 >)$ to the dto-queue.

(d) else if u has cached state for (a, b) and $(ttl = 0)$,
$b_{av}(a, b) \leftarrow b(a, b)$
delete any pending ito/dto message for (a, b) from the ito-queue and dto-queue.
add $dto(< a, b, dom(a), dom(b), 0, \infty >)$ to the dto-queue.

3. The dto-queue is flushed whenever there are packets in the queue.

There are several interesting points in the above algorithm. First, the way that the ito-queue and the dto-queue are flushed ensures that the decrease waves propagate much faster than the increase waves and suppress state propagation for unstable links. Second, waves are converted between *ito* and *dto* on-the-fly, depending on whether the cached value for the available bandwidth is lesser than the new update (*ito* wave generated) or not (*dto* wave generated). Third, after a distance of *ttl* (which depends on the current available bandwidth of the link), the $dto(< a, b, dom(a), dom(b), 0, \infty >)$ message ensures that all other *core* nodes which had state cached for this link now destroy that state. However, the $dto(< a, b, dom(a), dom(b), 0, \infty >)$ wave does not propagate throughout the network - it is suppressed as soon as it hits the *core* nodes which do not have link state for (a, b) cached (point 2(a) in *decrease wave* propagation). As we have noted before, the increase/decrease waves use the efficient *core* broadcast mechanism for propagation.

Essentially, the above algorithm ensures that the link-state information for stable high-bandwidth links gets propagated throughout the *core*, while the link-state information for unstable and low-bandwidth links remains local - which is the goal of the CEDAR state propagation algorithm.

We now answer the three key questions pertaining to the propagation of increase/decrease waves: *when* should a wave be generated, *how fast* should a wave propagate, and *how far* should a wave propagate.

4.2 When should a Increase/Decrease Wave be Generated?

Clearly, a wave should not be generated for every incremental change in the available bandwidth of the link. In CEDAR, we only generate a wave when the bandwidth has changed by a threshold value since the last wave was generated. The setting of the threshold parameter is the focus of the discussion in this section.

A simple approach would be to make the threshold a constant system parameter. In other words, a wave is generated when the bandwidth change since the last generation of the wave has exceeded a fixed value, say 10% of the raw channel bandwidth. While this approach works well when the available bandwidth has the same order of magnitude as a typical connection request (e.g. a channel capacity of 10Mbps and connection requests in the range of 1Mbps), it is very wasteful (in terms of the number of waves generated) if the channel capacity is orders of magnitude higher than the typical connection request (e.g. a channel capacity of 10Mbps and connection requests in the range of 20Kbps). In the latter case, we propose a logarithmic scale for the threshold, similar to the algorithm by [15]. This approach is motivated by the following argument: if the available bandwidth is 5Mbps and the typical request is 20Kbps, then an increase of the channel bandwidth to 5.02Mbps or a decrease of the channel bandwidth to 4.98Mbps is not significant. However, if the available channel bandwidth is 40Kbps, and the typical request is 20Kbps, then an increase to 60Kbps or a decrease to 20Kbps is a significant change. The advantage of the logarithmic update is that it does not wastefully generate increase/decrease waves when the change in link capacity is unlikely to alter the probability of computing admissible routes.

In the CEDAR simulations, we currently use the constant threshold approach for simplicity. Future work will migrate to the logarithmic scale for the threshold.

4.3 How Far does a Increase/Decrease Wave Propagate?

Our goal is to propagate information about stable high-bandwidth links throughout the network and localize the state of the low-bandwidth links. This is because every *core* node that caches information corresponding to a link can potentially use the bandwidth of the link, and the contention for a link is dependent on the number of *core* nodes caching the state of the link. For low-bandwidth links, it makes sense to have as few nodes as possible contending for the link, while for stable high-bandwidth links, it makes sense to have as many *core* nodes as possible know about the link in order to compute good routes. In order words, the maximum distance that the link state can travel (i.e. the time-to-live field) is an increasing function of the available bandwidth of the link.

Our current CEDAR simulation uses a linear function for computing the *ttl*. However, we have used a fluid model analysis of an ad hoc network, and determined that in general, the *ttl* should be a function of $b^{1/k}$, where k is a small number ranging from 1 to 3 depending on the density of the fluid generated from a source to its neighbors. The details of this analysis are ongoing work and not presented for lack of space. However, the summary of our results indicates that the selection of $k = 1$ (i.e. the constant function) works well for the environments under consideration.

4.4 How Fast does a Increase/Decrease Wave Propagate?

An increase wave waits for a fixed timeout period (which is a system parameter that should be approximately twice the expected interarrival time between the generation of two successive waves for any link in the network) at each node before being forwarded to its neighbors (using the *core* broadcast). Thus, increase waves propagate slowly. A decrease wave is immediately forwarded to its neighbors (using the *core* broadcast). Thus decrease waves move much faster and can kill increase waves for unstable links.

An added benefit of waiting at each node for a timeout period before forwarding the wave is that it naturally leads to the implicit establishment of a source-based breadth-first-search tree for the *core* broadcast described in Section 3.

5 QoS Routing in CEDAR

In the previous two Sections, we have described the *core* infrastructure (i.e. which nodes in the ad hoc network perform route computation and how they communicate among themselves) and the state propagation algorithm (i.e. what state does each *core* node contain). In this section, we complete the description of CEDAR by specifying how the *core* nodes use the state information to compute QoS routes.

The QoS route computation in CEDAR consists of three key components: (a) discovery of the location of the destination and establishment of the *core* path to the destination, (b) establishment of a short stable admissible QoS route from the source to the destination using the *core* path as a directional guideline, and (c) dynamic re-establishment of routes for ongoing connections upon link failures and topology changes in the ad hoc network.

Briefly, QoS route computation in CEDAR is an 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 $< s, d, b >$ triple, 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, if 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, i.e. $dom(s)$ uses its local state to find the shortest-widest admissible path to a node t in the domain of the furthest possible *core* node $dom(t)$ in the *core* path. 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 a failure to find an admissible path. As we have already discussed in previous sections, the knowledge of remote stable high-bandwidth links at each *core* node significantly improves the probability of finding an admissible path so long as such a path exists in the network.

In the following subsections, we describe the three key components of QoS routing in CEDAR.

5.1 Establishment of the Core Path

The establishment of a *core* path takes place when s requests $dom(s)$ to set up a route to d, and $dom(s)$ does not know the identity of $dom(d)$ or does not have a *core* path to $dom(d)$. Establishment of a *core* path consists of the following steps.

1. $dom(s)$ initiates a *core* broadcast to set up a *core* path with the following message: $< core\_path\_req, dom(s), d, b, P = null >$.

2. When a *core* node u receives the *core* path request message $< core\_path\_req, dom(s), d, b, P >$, it sets $P \leftarrow P \cup \{u\}$, and forwards the message to each of its nearby *core* nodes (according to the *core* broadcast algorithm) to whose domain there exists atleast one path (from u's domain) satisfying bandwidth b.

3. When $dom(t)$ receives the *core* path request message $< core\_path\_req, dom(s), d, b, P >$, it sends back a source rooted unicast *core_path_ack* message to $dom(s)$ along the inverse path recorded in P. The response message also contains P, the *core* path from $dom(s)$ to $dom(d)$.

Upon reception of the *core_path_ack* message from $dom(d)$, $dom(s)$ completes the *core* path establishment phase and enters the QoS route computation phase.

Note that by virtue of the *core* broadcast algorithm, the *core* path request traverses an implicitly (and dynamically) established source routed tree from $dom(s)$ which is typically a breadth-first search tree. Thus, the *core* path is approximately the shortest admissible path in the *core* graph from $dom(s)$ to $dom(d)$, and hence provides a good directional guideline for the QoS route computation phase.

5.2 QoS Route Computation

After the *core* path establishment, $dom(s)$ knows $dom(d)$ and the *core* path from $dom(s)$ to $dom(d)$. Recall from Section 3 that $dom(s)$ has the local topology - which includes all the nodes in its domain, and for each dominated node u, the bandwidth of each link incident on u, the adjacency list of u and the dominator of each of the neighbors of u. Recall from Section 4 that $dom(s)$ has the information gathered about remote links through increase/decrease waves, and for each such link (u, v), the bandwidth of (u, v), $dom(u)$, and $dom(v)$. $dom(s)$ thus has a partial knowledge of the ad hoc network topology, which consists of the up-to-date local topology, and some possibly out-of-date information about remote stable high-bandwidth links in the network. The following is the sequence of events in QoS route computation.

1. Using the local topology, $dom(s)$ tries to find a path from s to the domain of the furthest possible *core* node in the *core* path (say $dom(t)$) that can provide at least a bandwidth of b (bandwidth of the connection request). The bandwidth that can be provided on a path is the minimum of the individual available link bandwidths that comprise the path.

2. Among all the admissible paths (known using local state) to the domain of the furthest possible *core* node in the *core* path, $dom(s)$ picks the shortest-widest path using a two phase Dijkstra's algorithm [16].

3. Let t be the end point of the chosen path and $p(s, t)$ denote the path. $dom(s)$ sends $dom(t)$ the following message: $< s, d, b, P, p(s, t), dom(s), t >$, where s, d, and t are the source, destination, and intermediate node in the partially computed path, b is the required bandwidth, P is the *core* path, and $p(s, t)$ is the partial path.

4. $dom(t)$ then performs the QoS route computation using its local state identical to the computation described above.

5. Eventually, either there is an admissible path to d or the local route computation will fail to produce a path at some *core* node. The concatenation of the partial paths computed by the *core* nodes provides an end-to-end path that can satisfy the bandwidth requirement of the connection with high probability.

The *core* path is computed in one round trip, and the QoS route computation algorithm also takes one round trip. Thus, the route discovery and computation algorithms together take two round trips if the *core* path is not cached and one round trip otherwise.

Note that while the QoS route is being computed, packets may be sent from s to d using the *core* path. The *core* path thus provides a simple backup route while the primary route is being computed.

5.3 Dynamic QoS Route Recomputation for Ongoing Connections

Route recomputations may be required for ongoing connections under two circumstances: the end host moves, and there is some intermediate link failure (possibly caused by the mobility of an intermediate router). End host mobility can be thought of as a special case of link failure, wherein the last link fails.

CEDAR has two mechanisms to deal with link failures and reduce the impact of failures on ongoing flows: dynamic recomputation of an admissible route from the point of failure, and notification back to the source for source-initiated route recomputation. These two mechanisms work in concert and enable us to provide seamless mobility.

1. *QoS Route Recomputation at the Failure Point*: Consider that a link (u, v) fails on the path of an ongoing connection from s to t. The node nearest to the sender, u, then initiates a local route recomputation similar to the algorithm in Section 5.2. Once the route is recomputed, u updates the source route in all packets from s to t accordingly. If the link failure happens near the destination, then dynamic route recomputation at the intermediate node works very well because the route recomputation time to the destination is expected to be small, and packets in-flight are re-routed seamlessly.

2. *QoS Route Recomputation at the Source*: Consider that a link (u, v) fails on the path of an ongoing connection from s to t. The node nearest to the sender, u, then notifies s that the link (u, v) has failed. Upon receiving the notification, u stops its packet transmission, initiates a QoS route computation as in Section 5.2, and resumes transmission upon the successful re-establishment of an admissible route. If the link failure happens near the source, then source-initiated recomputation is effective, because the source can quickly receive the link-failure notification and temporarily stop transmission.

The combination of these two mechanisms is effective in supporting seamless communication inspite of mobility and dynamic topology changes. Basically, we use source-initiated recomputation as the long-term solution to handling link failure, while the short-term solution to handle packets in-flight is through the dynamic recomputation of routes from the intermediate nodes. Recomputation at the failure point is not really effective if the failure happens close to the source, but in this case, the number of packets in flight from s to u is small.

6 Performance Evaluation

We have evaluated the performance of CEDAR via both implementation and simulation. Our implementation consists of a small ad hoc network consisting of six mobile nodes that use a Photonics (Data Technology) 1Mbps Infrared network. We have customized the Linux 2.0.31 kernel to build our ad hoc network environment (written partly in user mode and partly in kernel mode). While the testbed shows a proof of concept and has exposed some of the practical difficulties in implementing CEDAR, our detailed performance evaluation has been using a simulator that faithfully implements the CEDAR algorithms.

For our simulations, we make the following assumptions about the network environment. (a) The channel capacity is 1Mbps. (b) It takes δ time for a node to successfully transmit a message over a single link, where δ is the degree of the node. (c) The dynamics of the topology are induced either by link failure or mobility. (d) Packets are source routed. (e) The transmission range for each node

is a 10 by 10 unit square region with the node at the center of this region (we generate our test graphs by randomly placing hosts in a 100 by 100 square region) and (vi) each CEDAR control packet transmission slot has a period of 2ms.

We present three sets of results from our simulations. The first set of results characterizes the performance of CEDAR in a best-effort service environment. The goal is to isolate the characterization of the basic routing algorithm from the effects of QoS routing for this set of results. The second set of results evaluates the performance of QoS routing in CEDAR. The third set of results evaluates the performance of CEDAR for ongoing connections in the presence of mobility. Essentially, the first two sets of results evaluate the performance of CEDAR in coming up with new routes in an ad hoc network, while the third set of results evaluates how CEDAR copes with link failures for ongoing connections.

In the first set of results, presented in Tables 1-2, we compare CEDAR to an optimal shortest path routing algorithm in a best-effort service environment. Our performance measures are the following: (i) average path length (APL), (ii) message complexity for route computation (MC) and (iii) time complexity for route computation (TC). In addition we present the *core usage (CU)* which is the average number of virtual links used in a route. Note that for the best effort environment, we do not have a concept of QoS for connections, and the increase/decrease waves essentially carry only link up/down information.

In the second set of results, presented in Tables 3-6, we evaluate the QoS routing algorithm of CEDAR. We use bandwidth as the quality of service parameter. Tables 3 and 4 compare the performance of CEDAR against the performance of an optimal *shortest-widest path* algorithm in terms of the the path length (*hops*) and the maximum available bandwidth (*bw*) for computed routes. Tables 5 and 6 compare the *accept/reject* ratio for CEDAR (with and without increase/decrease waves) and an optimal *shortest-widest path* algorithm.

In the third set of results, presented in Tables 7 and 8, we evaluate the performance of CEDAR for ongoing connections upon topology change (induced by link failures and host mobility). We consider the following parameters: (i) location of the link failure relative to the source (*Relative Link Position (RLP)*, (ii) number of packets sent, (iii) number of packets received, (iv) number of packets lost, (v) number of packets re-routed and (vi) minimum delay experienced by packets in the flow once the source receives notification about the link failure.

In all our simulations, the notation CEDAR_t stands for a simulation run of CEDAR at time t (increase/decrease waves would have thus been propagated up to time t).

6.1 Performance of CEDAR in a best-effort service environment

We use 3 randomly generated graphs for the results in this section. The graphs are of sizes 9, 20, and 30 respectively. The significant parameters for the graph - number of nodes(n), number of edges(m), number of *core* nodes(C), diameter of the *core*($diam_C$), average degree (δ) - are shown in the caption of the table containing the results for that particular graph. For each graph, we measure as mentioned above, the average path length (APL) in number of hops, message complexity for route computation (MC), route computation time (TC) in seconds and the *core* node usage ratio (CU) also in number of hops. These measurements are taken for both optimal shortest path routing and CEDAR. For CEDAR we measure these parameters at different points of time to study the impact of the propagation of *ito* waves. The time d used in the tables is the constant time for which *ito* waves are delayed at each hop.

As can be seen from the results, CEDAR performs reasonably well before the introduction of *ito/dto waves*, but converges to a near optimal performance once these waves are introduced. The ideal value for the CU should be zero as we seek to avoid using the virtual tunnels for data flow in order to prevent it from becoming a bottleneck. CEDAR exhibits a low CU because we preferentially avoid using the virtual tunnels; a virtual tunnel is chosen only if the local state at the *core* node performing the route computation is inadequate to forward the probe into a farther domain towards

the destination.

The counter-intuitive increase in APL, MC and TC with increase in time in these simulations are due to the fact that we are able to preferentially bypass the *core* nodes (as indicated by the decrease in CU) as more topology information becomes available. Thus the results shown in Tables 1 and 2 indicate the near optimal nature of CEDAR with increase in network stability.

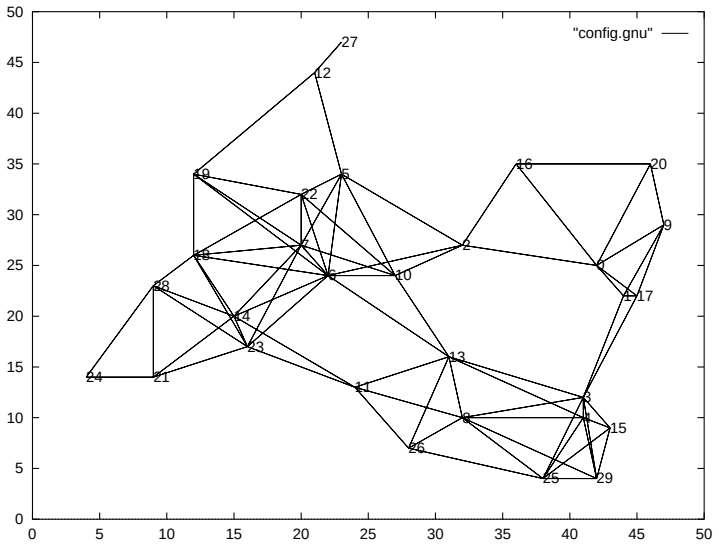

Figure 2: Graph used for Performance Evaluation Simulations

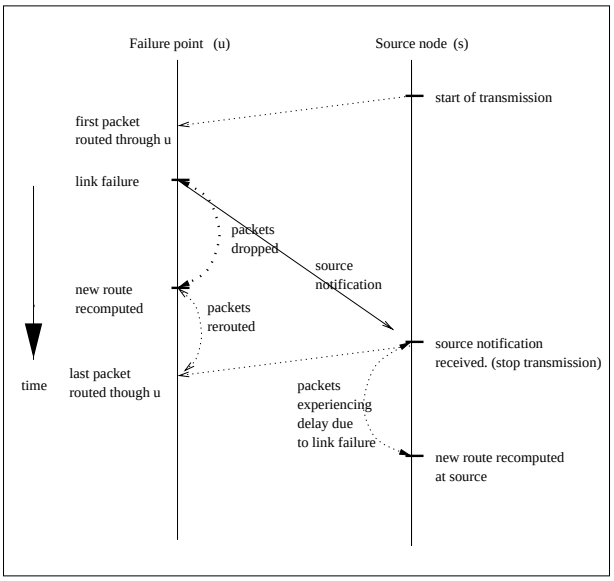

Figure 3: Effect of a link failure on an ongoing flow.

6.2 Performance of QoS Routing in CEDAR

Bandwidth is the QoS parameter of interest in CEDAR. We first compare QoS routing in CEDAR with an *optimal shortest widest path* algorithm with respect to two parameters: the available bandwidth

	APL	MC	TC	CU
optimal	2.4722	4.9444	0.028	N/A
$CEDAR_{t_0}$	2.5278	5.1667	0.029	0.6389
$CEDAR_{t_0+d}$	2.5694	5.2083	0.030	0.5972
$CEDAR_{t_0+2d}$	2.5694	5.2083	0.030	0.5972
$CEDAR_{t_0+3d}$	2.5694	5.2083	0.030	0.5972

(a)

	APL	MC	TC	CU
optimal	2.4316	4.8632	0.059	N/A
$CEDAR_{t_0}$	2.5053	5.4789	0.072	0.5473
$CEDAR_{t_0+d}$	2.6684	5.5816	0.073	0.4816
$CEDAR_{t_0+2d}$	2.4447	5.2868	0.069	0.4816
$CEDAR_{t_0+3d}$	2.4447	5.2868	0.069	0.4816
$CEDAR_{t_0+4d}$	2.4447	5.2868	0.069	0.4816
$CEDAR_{t_0+5d}$	2.4447	5.2868	0.069	0.4816

(b)

Table 1: Performance of CEDAR compared to an optimal approach. (a) (n,m,C,$diam_C$,$Avgdeg$) = (9,10,4,3,2) (b) (n,m,C,$diam_C$,$Avgdeg$) = (20,56,6,5,5)

	APL	MC	TC	CU
optimal	2.8229	5.6459	0.070	N/A
$CEDAR_{t_0}$	3.3988	6.8241	0.084	0.2689
$CEDAR_{t_0+d}$	3.3494	6.7298	0.083	0.1494
$CEDAR_{t_0+2d}$	3.3471	6.7022	0.083	0.1011
$CEDAR_{t_0+3d}$	3.4988	6.9367	0.084	0.1000
$CEDAR_{t_0+4d}$	3.2977	6.6390	0.083	0.0931
$CEDAR_{t_0+5d}$	3.1195	6.3931	0.080	0.0908
$CEDAR_{t_0+6d}$	3.1034	6.3747	0.079	0.0908
$CEDAR_{t_0+7d}$	3.1034	6.3747	0.079	0.0908

Table 2: Performance of CEDAR compared to an optimal approach. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5)

(*bw*) along the computed path, and the path length (in *hops*). The time field in Tables 3 and 4 represents the time at which the QoS route request was issued. Once the route is computed, each link locks the specified amount of resources along that route before processing the next connection request.

Next, we present the improvement in the performance of CEDAR with the advent of the *ito* and *dto* waves. We use the constant threshold approach to decide when to generate a wave. The *ttl* field in a wave is set using a linear function (of the advertised bandwidth) and while *ito* waves travel from one hop to another with a constant delay, *dto* waves travel are propagated from one hop to another with no delay. The parameter we use to evaluate the performance is the accept/reject ratio for connection requests. As can be seen, once the *ito/dto* waves are introduced, the performance of CEDAR is close to that of an optimal algorithm.

time	*source*	*destn*	bw_{req}	$hops_{CEDAR}$	bw_{CEDAR}	$hops_{OPT}$	bw_{OPT}
$t_0+\delta$	25	24	50	5	100	5	100
$t_0+2\delta$	29	18	40	5	100	5	100
$t_0+3\delta$	26	20	50	7	50	5	50
$t_0+4\delta$	0	28	30	8	50	4	50
$t_0+5\delta$	11	23	50	1	50	6	60
$t_0+6\delta$	24	19	50	4	50	4	50
$t_0+7\delta$	12	19	50	1	50	1	50
$t_0+8\delta$	16	11	25	5	50	5	60
$t_0+9\delta$	17	19	35	8	45	5	50
$t_0+10\delta$	10	25	20	3	50	3	50

Table 3: Performance of CEDAR compared to an optimal approach. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5) with connection results issued as shown

$time$	$source$	$destn$	bw_{req}	$hops_{CEDAR}$	bw_{CEDAR}	$hops_{OPT}$	bw_{OPT}
$t_0 + \delta$	23	11	50	1	100	1	100
$t_0 + 2\delta$	11	23	50	1	50	6	100
$t_0 + 3\delta$	26	18	30	4	50	4	50
$t_0 + 4\delta$	16	17	50	2	50	2	50
$t_0 + 5\delta$	21	24	50	1	50	2	100
$t_0 + 6\delta$	11	8	60	1	100	1	100
$t_0 + 7\delta$	15	17	50	2	50	2	50
$t_0 + 8\delta$	12	27	80	1	100	1	100
$t_0 + 9\delta$	17	10	30	5	50	4	50
$t_0 + 10\delta$	26	20	50	6	50	5	50

Table 4: Performance of CEDAR compared to an optimal approach. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5) with connection requests issued at times shown.

$starttime$	$endtime$	$source$	$destn$	bw_{req}	$accept_{OPT}$	$accept_{waves}$	$accept_{plain}$
0	2	11	23	5	*yes*	*yes*	*yes*
40	45	8	23	55	*yes*	*yes*	*no*
46	48	12	22	5	*yes*	*yes*	*yes*
49	70	5	12	15	*yes*	*yes*	*yes*
60	71	5	12	50	*yes*	*yes*	*no*
62	72	13	2	65	*yes*	*yes*	*yes*
98	99	3	2	65	*yes*	*yes*	*no*
101	110	26	11	45	*yes*	*yes*	*yes*
111	105	19	28	105	*no*	*no*	*no*
110	120	26	25	5	*yes*	*yes*	*yes*

Table 5: Performance improvement of CEDAR with the advent of *ito* and *dto* waves. The accept/reject ratio for optimal, CEDAR with waves and CEDAR without waves are 9:1, 9:1 and 6:4 respectively. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5).

For the results in this section, we use the 30 node graph in Figure 2 with link bandwidths randomly set to either 50 units or 100 units. Note from Tables 3 and 4 that CEDAR approximates the optimal algorithm for the scenarios simulated. Further, from Tables 5 and 6, we can see the utility of the *ito* and *dto* waves to CEDAR. In these tables, the column headers $accept_{OPT}$, $accept_{waves}$ and $accept_{plain}$represent whether the connection request was accepted in the optimal algorithm, *CEDAR with waves* and *CEDAR without waves* respectively.

6.3 Effect of Link Failures on Ongoing Flows in CEDAR

While the previous sets of results evaluated the performance of CEDAR in terms of generating initial routes, we now turn our attention to the ability of CEDAR to provide seamless connectivity in ad hoc networks inspite of the dynamics of the network topology.

The following is the sequence of events that occurs on a link failure:

- Link (u, v) fails on path from s to d.
- u sends back notification to source and starts recomputation of route from u to d.
- For each subsequent packet that u receives, it drops the packet if the recomputation of the previous step is not yet completed. Otherwise u forwards the packet along the new route with the modified source route.

$starttime$	$endtime$	$source$	$destn$	bw_{req}	$accept_{OPT}$	$accept_{waves}$	$accept_{plain}$
0	8	8	13	55	*yes*	*yes*	*yes*
2	25	3	13	55	*yes*	*yes*	*yes*
20	31	3	13	55	*yes*	*yes*	*no*
22	32	5	0	20	*yes*	*yes*	*yes*
28	38	6	7	16	*yes*	*yes*	*yes*
54	64	16	7	41	*yes*	*yes*	*yes*
56	64	16	4	45	*yes*	*no*	*no*
63	73	13	19	44	*yes*	*yes*	*no*
64	74	16	11	23	*yes*	*yes*	*yes*
79	79	4	9	38	*yes*	*yes*	*yes*

Table 6: Performance improvement of CEDAR with the advent of *ito* and *dto* waves. The accept/reject ratio for optimal, CEDAR with waves and CEDAR without waves are 10:0, 9:1 and 7:3 respectively. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5)

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	276	247	29	0	0.140
2	294	237	57	2	0.134
3	298	260	38	63	0.136
4	294	247	47	59	0.128
5	298	297	1	122	0.138
6	300	299	1	141	0.152

(a)

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	239	214	25	0	0.136
2	247	199	48	1	0.104
3	255	248	7	50	0.138
4	253	224	29	52	0.138
5	255	254	1	105	0.138
6	268	267	1	118	0.140

(b)

Table 7: Performance of CEDAR's recovery mechanism on a link failure. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5) with link failure on path for flow from node 24 to 20 (a) input traffic generated using Poisson distribution (b) input traffic generated using MMPP distribution

- Upon receiving a link failure notification, s stops sending packets for that flow immediately and starts recomputation of the route from s to d.
- Once the recomputation of the previous step is complete, the source once again starts sending packets for that flow along the new route.

This sequence of events is also illustrated in Figure 3.

We again use the 30 node graph in Figure 2 for evaluating the performance of CEDAR in the presence of link failures. For an arbitrary flow, we bring down links that are progressively farther away from the source and we show the impact of that link failure in terms of number of packets lost, number of packets re-routed and delay for subsequent packets.

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	279	261	18	0	0.132
2	280	241	39	0	0.130
3	288	267	21	39	0.126
4	307	306	1	95	0.128
5	309	308	1	106	0.130

(a)

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	235	210	25	0	0.128
2	239	205	34	0	0.126
3	245	239	6	35	0.126
4	267	266	1	81	0.126
5	270	269	1	89	0.120

(b)

Table 8: Performance of CEDAR's recovery mechanism on a link failure. (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5) with link failure on path for flow from node 16 to 24 (a) input traffic generated using Poisson distribution (b) input traffic generated using MMPP distribution

As can be observed from Tables 7 and 8, the relative location of the link failure with respect to the source has a significant impact on the above mentioned parameters. In particular,

- If the link failure is very close to the source, the recomputation time at the node before the failure is large and hence a considerable number of packets can potentially be lost. But the source notification message, described earlier in Section 5, reaches the source almost immediately and hence prevents a large number of packets from getting dropped.
- If the link failure is very close to the destination, the recomputation time at the node before the failure is very small and hence very few packets get dropped. But the source notification message reaches the source with some delay and hence the number of packets that get re-routed is large.
- If the link failure is somewhere midway between the source and destination, both mechanisms (route recomputation and source notification) fail to react fast enough to prevent loss of packets and hence the number of packets lost and re-routed is relatively large.

For the generation of packets in a flow, we use both Poisson and MMPP (Markov modulated poisson process) distributions in our simulations.

7 Related Work

We present a brief survey of related work in two areas: routing in ad hoc networks, and QoS routing in wireline networks.

7.1 Routing in ad hoc networks

Most ad hoc routing algorithms that we are aware of generously use flooding or broadcasts for route computation. As we have mentioned before, our experience has been that flooding in ad hoc networks does not work well due to the abundance of hidden and exposed stations.

The ad hoc routing algorithms in [8, 17, 18] provide a single route in response to a route query from a source; these algorithms have low overhead but sometimes use sub-optimal and stale routes. [8] uses flooding, in the worst case, for finding routes. [17] considers signal strength as a metric for routing. [18] uses additional criteria to judge routes: the relaying load, or number of existing connections passing through an intermediate node; and location stability, as measured in associativity ticks. [2] uses a *spine* structure for route computation and maintenance. It provides optimal or near optimal routes depending upon the nature of information stored in the spine nodes, but incurs a large overhead for state and spine management.

Previous work on tactical packet radio networks had led to many of the fundamental results in ad hoc networks. [19] has proposed an architecture similar to the *core* called the *linked clusterhead* architecture but it uses gateways for communication between clusterheads and does not attempt to minimize the size of the infrastructure. [3] uses minimum-hop distance-vector routing. To extend routing to larger networks, hierarchical routing has been proposed [20], with either distance-vector routing [21] or link-state routing [22] used within each cluster.

Other shortest-paths routing algorithms incorporate measures of delay or congestion into the path weights [23, 24], but these algorithms usually have some centralized computation of the delay and congestion metrics. In a different tack, neural-network-based computation is used to select routes other than via minimum hop count with some measure of success in reducing congestion [25].

The multipath routing algorithms are more robust than the single route on demand algorithms, at a cost of higher memory and message requirements. In [9], a source may learn of more than one

route to a destination, hence the routing decision is flexible and fault tolerant. The hybrid routing algorithm in [26] combines the robustness of multipath routing with the low overhead of single route on demand: when node mobility is high, [8] is used; when node mobility is low, [9] is used. Finally, the temporally-ordered routing algorithm in [10] improves on [9] by using time tags to localize topology changes.

As is apparent from our work, we have used many of the results from contemporary literature. The notion of on-demand routing, use of stability as a metric to propagate link-state information, clustering, and the use of cluster-heads for local state aggregation have all been proposed in previous work in one form or the other. We believe that our contribution in this paper is to propose a unique combination of several of these ideas in conjunction with the novel use of the core, increase/decrease waves, core broadcast, and local state-based routing in the domain of QoS routing. Consequently, we are able to compute good admissible routes with high probability and still adapt effectively with low overhead to the dynamics of the network topology.

7.2 QoS Routing

QoS routing algorithms can be mainly classified into two categories: distributed [27, 28, 29, 30, 31, 32] and centralized [33, 34, 35, 36].

Wang and Crowcroft [27] show that if the total number of independent additive and multiplicative QoS constraints is more than one, then the QoS routing problem is NP complete. Assuming that all routers are using Weighted Fair Queuing scheduling, Ma and Steenkiste [28] and Pornavalai et. al. [29] show that the relationships between various QoS parameters (bandwidth, delay, delay-jitter and buffer space) can be utilized to find QoS routes in polynomial time. Wang and Crowcroft [27] and Guerin et. al. [30] propose *shortest-widest* path. A comparison of shortest-widest, widest-shortest, dynamic alternative and the shortest distance path is presented in [28]. Salama et. al. [31] propose a distributed algorithm for finding delay constrained unicast routes (DCUR). The algorithm requires every node to keep a least cost next hop and a least delay next hop corresponding to every destination. The worst case time complexity of $O(V^3)$ has been improved to $O(V)$ by Sun and Langendoerfer [32]. We have used the optimal global knowledge-based algorithm for comparison with our CEDAR algorithm.

Ma and Steenkiste [33] propose a centralized algorithm for finding the fair share of a best effort flow. The fair share of a bandwidth can be used for shortest-widest, widest-shortest or any other algorithm for routing the best effort traffic. Widyono [34] proposes a centralized Constrained Bellman Ford (CBF) algorithm and a path merging algorithm to find a minimum cost source rooted tree to a set of destinations with delay bounds. Effects of uncertain parameters on QoS routing with end-to-end delay requirements is discussed in [35]. For a wide class of probability distributions, Lorenz and Orda [35] and Guerin and Orda [36] propose efficient exact solutions to *optimal delay partition problem* (OP) and a pseudo polynomial solution to optimally partitioned most probable path (OPMP). This work is currently being applied in order to extend the CEDAR approach to support delay as a QoS parameter in ad hoc network environments.

A simulation based study of the relationship between routing performance and the amount of update traffic is reported by Apostolopoulos et. al. [37].

8 Conclusion

In this paper, we have presented CEDAR, a Core-Extraction Distributed Ad hoc Routing algorithm for providing QoS in ad hoc network environments. CEDAR has three key components: (a) the establishment and maintenance of the core of the network for performing the route computations, (b) propagation and use of bandwidth and stability information of links in the ad hoc network, and (c) the QoS route computation algorithm. While the core provides an efficient and low-overhead

infrastructure to perform routing and broadcasts in an ad hoc network, the increase/decrease wave based state propagation mechanism ensures that the core nodes have the important link-state they need for route computation without incurring the high overhead of state maintenance for dynamic links. The QoS routing algorithm is robust and uses only local state for route computation at each core node.

We believe that CEDAR is a robust and adaptive algorithm that reacts quickly and effectively to the dynamics of the network while still approximating link-state performance for stable networks. Our simulations show that CEDAR produces good stable admissible routes with a high probability if such routes exist. Furthermore, CEDAR does not require high maintenance overhead even for highly dynamic networks. Ongoing work on CEDAR is focusing on two areas. (a) While we have shown that CEDAR is effective for small to medium size networks, we are working on a hierarchically clustered version of CEDAR that can provide QoS routing in large ad hoc networks. (b) While we have only considered bandwidth as the QoS parameter in this work, we are extending CEDAR to support delay also as a QoS parameter.

References

[1] R. Nair, B. Rajagopalan, Hal Sandick, and Eric Crawley. A framework for QoS-based routing in the internet. Internet Draft draft-ietf-qosr-framework-05.txt, May 1998.

[2] R. Sivakumar, B. Das, and V. Bharghavan. Spine routing in ad hoc networks. *ACM/Baltzer Cluster Computing Journal (special issue on Mobile Computing). To appear*, 1998.

[3] J. Jubin and J. D. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[4] D. A. Hall. Tactical internet system architecture for task force XXI. In *Proceedings of the Tactical Communications Conference*, Ft. Wayne, Indiana, May 1996.

[5] V. Bharghavan, S. Shenker A. Demers, and L. Zhang. MACAW: A medium access protocol for wireless LANs. In *Proceedings of ACM SIGCOMM*, London, England, August 1994.

[6] Chane L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[7] Songwu Lu, Vaduvur Bharghavan, and Rayadurgam Srikant. Fair queuing in wireless packet networks. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[8] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad-hoc wireless networks. In *Mobile Computing*, (ed. T. Imielinski and H. Korth), Kluwer Academic Publishers, 1996.

[9] M. S. Corson and A. Ephremides. A highly adaptive distributed routing algorithm for mobile wireless networks. *ACM/Baltzer Wireless Networks Journal*, 1(1):61–81, February 1995.

[10] V. D. Park and M. S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of 1997 IEEE Conference on Computer Communications*, April 1997.

[11] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *Proceedings of ACM SIGCOMM*, pages 234–244, London, England, August 1994.

[12] S. Murthy and J. J. Garcia-Luna-Aceves. A routing protocol for packet radio networks. In *Proceedings of ACM SIGCOMM '97*, Cannes, France, September 1997.

[13] V. Bharghavan. Performance of multiple access protocols in wireless packet networks. In *International Performance and Dependability Symposium*, Durham, North Carolina, September 1998.

[14] S. Guha and S. Khuller. Approximation algorithms for connected dominating sets. Technical Report 3660, Inst. for Adv. Computer Studies, Dept. of Computer Sci., Univ. of Maryland, College Park, June 1996.

[15] Baruch Awerbuch, Yi Du, Bilal Khan, and Yuval Shavitt. Routing through networks with topology aggregation. In *IEEE Symposium on Computers and Communications*, Athens, Greece, June 1998.

[16] Qingming Ma and Peter Steenkiste. On path selection for traffic with bandwidth guarantees. In *Proceedings of Fifth IEEE International Conference on Network Protocols*, Atlanta, October 1997.

[17] R. Dube, C. D. Rais, K.-Y. Wang, and S. K. Tripathi. Signal stability based adaptive routing (SSA) for ad-hoc mobile networks. Technical Report UMCP-CSD:CS-TR-3646, Dept. of Computer Science, Univ. of Maryland, College Park, September 1996.

[18] C.-K. Toh. A novel distributed routing protocol to support ad-hoc mobile computing. In *Proceedings of 15th IEEE Annual International Phoenix Conference on Computers and Communications*, pages 480–486, 1996.

[19] A. Ephremides, J. E. Wieselthier, and D. J. Baker. A design concept for reliable mobile radio networks with frequency hopping signaling. In *Proceedings of the IEEE*, pages 56–73, January 1987.

[20] N. Shacham and J. Westcott. Future directions in packet radio architectures and protocols. *Proceedings of the IEEE*, 75(1):83–99, January 1987.

[21] P. F. Tsuchiya. The landmark hierarchy: A new hierarchy for routing in very large networks. In *ACM SIGCOMM*, pages 35–42, Stanford, California, 1988.

[22] W. T. Tsai, C. V. Ramamoorthy, W. K. Tsai, and O. Nishiguchi. An adaptive hierarchical routing protocol. *IEEE Transactions on Computers*, 38(8):1059–1075, August 1989.

[23] R. L. Hamilton, Jr. and H.-C. Yu. Optimal routing in multihop packet radio networks. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, pages 389–396, San Francisco, June 1990.

[24] D. L. Mills. Wiretap: An experimental multiple-path routing algorithm. In *Computer Communications Review*, volume 19, pages 85–98, January 1989.

[25] J. E. Wieselthier, C. M. Barnhart, and A. Ephremides. A neural network approach to routing in multihop radio networks. In *Proceedings of 1991 IEEE Conference on Computer Communications (INFOCOM)*, pages 1074–1083, Miami, Florida, April 1991.

[26] S. Corson, J. Macker, and S. Batsell. Architectural considerations for mobile mesh networking, May 1996.

[27] Z. Wang and J. Crowcroft. QoS routing for supporting resource reservation. *IEEE Journal on Selected Areas in Communications*, 14(7):1228–1234, September 1996.

[28] Qingming Ma and Peter Steenkiste. Quality-of-service routing for traffic with performance guarantees. In *Proceedings of IFIP Fifth International Workshop on Quality of Service*, pages 115–126, New York, May 1997.

[29] C. Pornavalai, G. Chakraborty, and N. Shiratori. QoS based routing algorithm in integrated services packet networks. In *International Conference on Network Protocols*, Atlanta, USA, October 1997.

[30] Roch A. Guerin, Ariel Orda, and D. Williams. QoS routing mechanisms and OSPF extensions. Internet Draft, draft-guerin-qos-routing-ospf-00.txt, 1996.

[31] H. F. Salama, D. S. Reeves, and Y. Viniotis. A distributed algorithm for Delay-Constrained Unicast Routing. Technical Report TR-96/26, Center for Advanced Computing and Communication, North Carolina State University, June 1996.

[32] Q. Sun and H. Langendoerfer. A new distributed routing algorithm for supporting delay-sensitive applications. Internal Report, Institute of Operating Systems and Computer Networks, Braunschweig, Germany, March 1997.

[33] Qingming Ma, Peter Steenkiste, and Hui Zhang. Routing high-bandwidth traffic in max-min fair share networks. In *Proceedings of ACM SIGCOMM*, pages 206–217, Stanford, California, August 1996.

[34] Ron Widyono. The design and evaluation of routing algorithms for real-time channels. Technical Report TR-94-024, International Computer Science Institute, Berkeley, CA, June 1994.

[35] Dean H. Lorenz and Ariel Orda. QoS routing in networks with uncertain parameters. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, California, April 1998.

[36] Roch A. Guerin and Ariel Orda. QoS-based routing in networks with inaccurate information: Theory and algorithms. In *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Kobe, Japan, April 1997.

[37] George Apostolopoulos, Roch Guerin, Sanjay Kamat, and Satish K. Tripathi. Quality of service routing: A performance perspective. In *Proceedings of ACM SIGCOMM*, Vancouver, Canada, September 1998.