CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm

Raghupathy Sivakumar   Prasun Sinha   Vaduvur Bharghavan

Coordinated Science Laboratory

University of Illinois at Urbana-Champaign

Email:{sivakumr, prasun, 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 high-bandwidth and stable 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.

Keywords—**Ad-hoc routing, QoS Routing, Mobile Networking**

I. INTRODUCTION

An ad hoc network is a dynamic multi-hop wireless network that is established by a group of mobile nodes on a shared wireless channel by virtue of their proximity to each other. 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, as well as in non-military environments such as classrooms and conferences. Military network environments typically require quality of service for their mission-critical applications. In non-military environments, multimedia applications also require routes satisfying quality of service requirements. Hence, the focus of this paper is on providing *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].

CEDAR dynamically establishes a core of the network, and then incrementally propagates the link state of stable high bandwidth links to the nodes of the core. Route computation is on-demand, and is performed by core nodes using only local state. 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*: A set of nodes is distributedly and dynamically elected to form the core of the network by approximating a minimum dominating set 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 route computation on behalf of these nodes.

- *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.

- *Route computation*: Route computation first establishes a core-path from the *dominator* (see Section II) 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 (which then becomes the source for the next iteration) satisfying 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.

The rest of this paper is organized as follows. Section II describes the network model, terminology, and the goals of CEDAR. Section III describes the computation and dynamic management of the core of the network. Section IV describes the link state propagation through the core using increase and decrease waves. Section V describes the route computation algorithm of CEDAR, and puts together the algorithms described in the previous sections. Section VI analyzes the performance of CEDAR through simulations. Section VII compares CEDAR to related work, and Section VIII concludes the paper.

II. 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.

A. Network Model

We assume that all the nodes communicate on the same shared wireless channel. For Frequency Hopping Spread Spectrum, this implies that all nodes have the same frequency hopping pattern, while for Direct Sequence Spread Spectrum, this implies that all nodes have the same pseudo-random sequence. We assume that each transmitter has a fixed transmission range, and that neighborhood is a commutative property (i.e. if A can

---

[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.

hear B, then B can hear A). Because of the local nature of transmissions, hidden and exposed stations are typically present in an ad hoc network. We assume the use of a CSMA/CA like algorithm such as MACAW [2] 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 (Request to Send from sender to receiver) - CTS (Clear To Send from receiver to sender) - Data (from sender to receiver) - Ack (from receiver to sender).

We assume small to medium size networks ranging between tens to hundreds of nodes. For larger networks, we propose a clustering algorithm in a related work [3] and apply CEDAR hierarchically within each cluster, for a cluster of clusters, etc. We assume that the change in network topology is frequent, but not frequent enough to render any kind of route computation useless.

We assume that the MAC/link layer can estimate the available link bandwidth. 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 III.

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.

B. 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 ith *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 ith 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.
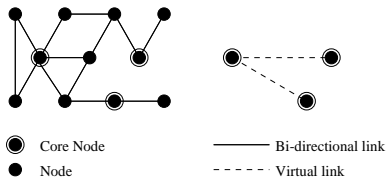

Fig. 1. An example showing a network with a possible set of core nodes and the corresponding core graph.

Given an MDS V_C of a graph G, we define a core of the graph $C = (V_C, E_C)$, where $E_C = \{[u, v] \mid u \in V_C, v \in V_C, u \in N'_3(v)\}$. 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 (see Figure 1).

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).

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

C. Goals of CEDAR

Ad hoc networks are typically dynamic and hence, routing in ad hoc networks has the following goals.

- Route computation must be distributed, because centralized routing in a dynamic network is impossible even for fairly small networks.
- 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.
- 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 node must have quick access to routes on-demand.
- 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.
- Stale routes must be either avoided or detected and eliminated quickly.
- Broadcasts must be avoided as far as possible because broadcasts are highly unreliable in ad hoc networks.
- If the topology stabilizes, then routes must converge to the optimal routes.
- It is desirable to have a backup route when the primary route has become stale and is being recomputed.

QoS routing in ad hoc networks is relatively unchartered territory. We have the following goals for QoS routing in ad hoc networks.

- 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, if such a route exists.
- 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.
- Unstable or low bandwidth links must not cause state propagation throughout the network. Only stable high bandwidth link information must be propagated to distant nodes which are involved in route computation.
- As the network becomes stable, the routing algorithm should start providing near-optimal routes.

- 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. As a result, we sacrifice optimality of routes. However, we show in Section VI that by virtue of our algorithm design, CEDAR is able to approximate the state-intensive shortest-widest path algorithm in the average case, though it still adapts efficiently to the network dynamics.

III. CEDAR Architecture and the Core

In this section, 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 the propagation of increase/decrease waves, and for the establishment of the core path in the route computation phase.

A. 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 [4], [5], [6]. In contrast, the *spine* architecture [3] only involves the nodes of an approximate minimum connected dominating set of the ad hoc network. Similarly, CEDAR also uses only the core nodes for state management and route computation. Moreover, we believe that the core provides the benefits of the spine architecture without incurring the high maintenance overhead of the spine. Following are the reasons for using a core-based infrastructure in CEDAR.

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 presence of hidden and exposed stations. Route probes [4] 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. On the other hand, if only a core subset of the 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.

B. Generation and Maintenance of the Core in CEDAR

Ideally, the core consists 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. The best known distributed algorithm for MDS approximation [7] 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 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: $(u, \{(w, dom(w)) \mid \forall w \in N_1'(u)\})$. 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 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. As shown in Section II, 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.

C. Core Broadcast and its Application to CEDAR

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, our experience has shown otherwise. In particular, we have observed that flooding probes, which causes repeated local broadcasts, is highly unreliable because of the presence of hidden and exposed stations. Thus, we 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, 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. 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. 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, we believe that 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 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 at most one extra message transmission. 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.
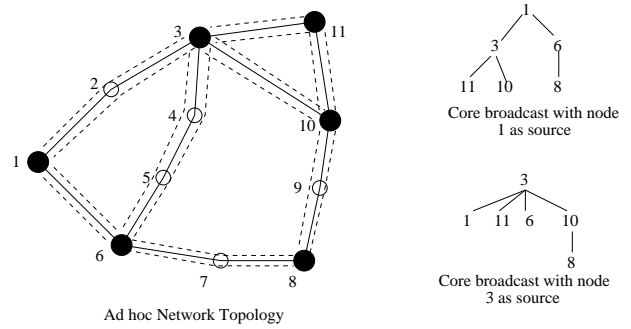

Fig. 2. Example of a core broadcast. Nodes in black are core nodes. Solid lines denote links in the ad hoc network. Dotted pipes denote virtual links in the core graph.

In the ad hoc network shown in Figure 2, 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.

Core broadcast in CEDAR 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.

2. The number of messages is $O(|V_C|)$ in the average case. In particular, the only case we transmit extra data 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 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.

IV. QoS State Propagation in CEDAR

As far as the nature of state maintained at each core node is concerned, 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 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 for stable networks, 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. 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.

CEDAR achieves this goal using increase and decrease waves. A slow-moving increase-wave denotes an increase of bandwidth on a link, and a fast-moving decrease-wave 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 stops the slower moving increase wave from propagating, thus ensuring that the link-state corresponding to dynamic links is kept 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 discuss some issues related to their implementation.

A. 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 (a, b) about which a core node caches link-state, the core node contains the cached available bandwidth $bw_{cached}(a, b)$.

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 increases beyond a threshold value, then the two end-points of l inform their dominators for initiating a core broadcast for an increase wave:
$ito(a, b, dom(a), dom(b), bw(a, b), ttl)$
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, $bw(a, b)$ denotes the available bandwidth on the link, and ttl is a 'time-to-live' field that denotes the maximum distance to which this wave can be propagated as an increase wave. The ids of the dominators of the link end-points are required by the routing algorithm. ttl is an increasing function of the available bandwidth, as described in Section IV-B.

2. When a core node u receives an *ito* wave $ito(a, b, dom(a), dom(b), bw(a, b), ttl)$,
  (a) if u has no state cached for (a, b) and $(bw(a, b) > 0)$,
     $bw_{cached}(a, b) \leftarrow bw(a, b)$
     if($ttl > 0$), then
        add $ito(a, b, dom(a), dom(b), bw(a, b), ttl - 1)$
        to the ito-queue.
  (b) else if u has cached state for (a, b) and $(ttl > 0)$,
     $bw_{cached}(a, b) \leftarrow bw(a, b)$
     delete any pending ito/dto message for (a, b)
        from the ito-queue and dto-queue.
     if $(bw_{cached}(a, b) < bw(a, b))$
        add $ito(a, b, dom(a), dom(b), bw(a, b), ttl - 1)$
        to the ito-queue.
     else if $(bw_{cached}(a, b) > bw(a, b))$,
        add $dto(a, b, dom(a), dom(b), bw(a, b), ttl - 1)$
        to the dto-queue.
  (c) else if u has cached state for (a, b) and $(ttl = 0)$,
     $bw_{cached}(a, b) \leftarrow bw(a, b)$
     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 $bw(a, b)$ decreases beyond a threshold value, then the two end-points of l inform their dominators for initiating a core broadcast for a decrease wave: $dto(a, b, dom(a), dom(b), bw(a, b), ttl)$, 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 $dto(a, b, dom(a), dom(b), bw(a, b), ttl)$,
  (a) if u has no state cached for (a, b) and $(bw(a, b) = 0)$,
        the wave is not propagated any further.
  (b) else it is processed in the same way as the *ito* wave is processed above.

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. As we have noted before, the increase/decrease waves use the efficient core broadcast mechanism for propagation.

B. Issues with implementing Increase/Decrease Waves

We have looked at how the waves are propagated, but there are several implementational issues which are worth exploring.

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. Effectively, the range of available bandwidth is divided into equal intervals, and a wave is initiated only when a new interval in entered. A logarithmic scale, where the size of the interval is not a constant, but increases with bandwidth, has been proposed in [8]. This might be used as an alternative method for deciding when a wave needs to be initiated.

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.

As discussed earlier, the decrease-waves travel faster than the increase waves. The time intervals for which these waves are buffered at a node are another set of parameters which need to carefully studied.

V. QoS Routing in CEDAR

It is possible to use any of the well known ad hoc routing algorithms such as DSR [4], TORA [5], AODV [6], ZRP [9], [10] etc. in the core graph. CEDAR has its own QoS route computation algorithm, which 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) 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 upto 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 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.

A. 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 (say with required bandwidth b), 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:
$(dom(s), d, b, P \leftarrow null)$, where P is the path traversed by this message so far, and is initialized to *null*.
2. When a core node u receives the core path request message $(dom(s), d, b, P)$, it appends u to P, and forwards the message to each of its nearby core nodes (according to the core broadcast algorithm)
3. When $dom(t)$ receives the core path request message $(dom(s), d, b, P)$, it sends back a source routed 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 rooted 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.

B. QoS Route Computation

Recall from Sections III and IV that $dom(s)$ 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 on 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 [11].

3. Let t be the end point of the chosen path. $dom(s)$ sends the following message to $dom(t)$: $(s, d, b, P, p(s, t), dom(s), t)$, 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 route computed so far.

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. Also note that CEDAR uses source routing for both control as well as data packets. As source routing has an overhead, modifying CEDAR for next hop routing is part of our ongoing work.

C. Dynamic QoS Route Recomputation for Ongoing Connections

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

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 V-B. 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 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 V-B, 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.

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. Note that updation of source routes at intermediate nodes might have implications on authentication and security.

VI. 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 CEDAR.

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 nodes in a 100 by 100 square region) and (f) 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 I-II, 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 III-IV, we evaluate the QoS routing algorithm of CEDAR. We use bandwidth as the quality of service parameter. Table III compares 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. Table IV compares 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 V and VI, we evaluate the performance of CEDAR for ongoing connections upon topology change (induced by link failures and node 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).

A. Performance of CEDAR without QoS Routing

We use 3 randomly generated graphs for the results in this section. The graphs are of sizes 9, 15 and 20 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 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. The source and destination pairs are chosen randomly.

As can be seen from the results, CEDAR performs reasonably

well before the introduction of *ito/dto* waves, but converges very fast to a near optimal performance once these waves are introduced. The tables show the different measures, *APL, MC, TC* and *CU* at various time instants, till CEDAR converges. 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 edge 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 (as indicated by the decrease in *CU*) as more topology information becomes available. Thus the results shown in Tables I and II indicate the near optimal nature of CEDAR with increase in network stability.

	APL	MC	TC	CU
optimal	2.47	4.94	0.028	N/A
$CEDAR_{t_0}$	2.52	5.16	0.029	0.63
$CEDAR_{t_0+d}$	2.56	5.20	0.030	0.59
$CEDAR_{t_0+2d}$	2.56	5.20	0.030	0.59

(a) $(n,m,C,diam_C,Avgdeg) = (9,10,4,3,2)$

	APL	MC	TC	CU
optimal	2.63	5.27	0.043	N/A
$CEDAR_{t_0}$	2.77	5.59	0.047	1.54
$CEDAR_{t_0+d}$	2.88	5.69	0.048	1.37
$CEDAR_{t_0+2d}$	2.88	5.69	0.048	1.37

(b) $(n,m,C,diam_C,Avgdeg) = (15,26,8,5,3)$

TABLE I

PERFORMANCE OF CEDAR COMPARED TO AN OPTIMAL APPROACH.

	APL	MC	TC	CU
optimal	2.43	4.86	0.059	N/A
$CEDAR_{t_0}$	2.50	5.47	0.072	0.54
$CEDAR_{t_0+d}$	2.66	5.58	0.073	0.48
$CEDAR_{t_0+2d}$	2.44	5.28	0.069	0.48
$CEDAR_{t_0+3d}$	2.44	5.28	0.069	0.48

TABLE II

PERFORMANCE OF CEDAR COMPARED TO AN OPTIMAL APPROACH.
$(n,m,C,diam_C,Avgdeg) = (20,56,6,5,5)$

B. 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 (*bw*) along the computed path, and the path length (in *number of hops*). The time field in Table III 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, i.e. we assume instantaneous reservation.

Next, we present the improvement in the performance of CEDAR with the advent of the *ito* and *dto* waves. We use the

$time$	src	dst	bw_{req}	h_C	bw_C	h_O	bw_O
$t_0 + \delta$	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 III

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.

t_s	t_e	s	d	bw_r	acc_o	acc_w	acc_{nw}
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 IV

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. $(n,m,C,diam_C,Avgdeg) = (30,79,11,7,5)$.

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	276	247	4	0	0.140
2	294	237	8	1	0.134
3	298	260	5	8	0.136
4	294	247	6	8	0.128
5	298	297	1	16	0.138
6	300	299	1	18	0.152

(a) input traffic generated using Poisson distribution

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	239	214	4	0	0.136
2	247	199	6	1	0.104
3	255	248	1	7	0.138
4	253	224	4	7	0.138
5	255	254	1	14	0.138
6	268	267	1	15	0.140

(b) input traffic generated using MMPP distribution

TABLE V

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.

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	279	261	3	0	0.132
2	280	241	5	0	0.130
3	288	267	3	5	0.126
4	307	306	1	12	0.128
5	309	308	1	14	0.130

(a) input traffic generated using Poisson distribution

RLP	$sent$	$recvd$	$dropped$	$rerouted$	$delay$
1	235	210	4	0	0.128
2	239	205	5	0	0.126
3	245	239	1	5	0.126
4	267	266	1	11	0.126
5	270	269	1	12	0.120

(b) input traffic generated using MMPP distribution

TABLE VI

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.

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.

For the results in this section, we use the 30 node graph in Figure 3 with link bandwidths randomly set to either 50 units or 100 units. In the column headers in Table III, h_C, bw_C and h_O, bw_O stand for the hop-count and available bandwidth of routes computed by CEDAR and the optimal algorithm respectively. Note from Table III that CEDAR approximates the optimal algorithm for the scenarios simulated. Further, from Table IV, we can see the utility of the *ito* and *dto* waves to CEDAR. In this table, the column headers acc_o, acc_w and acc_{nw} represent whether the connection request was accepted in the optimal algorithm, *CEDAR with waves* and *CEDAR with no waves* respectively. t_s and t_e denote the start and end times for the connection and s and d denote the source and the destination, respectively.

C. 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 in spite 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.
- Upon receiving a link failure notification, s stops sending packets for that flow immediately and starts recomputation
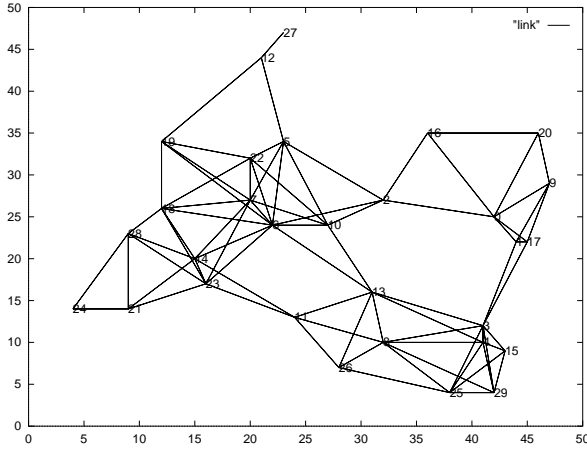
Fig. 3. Graph used for Performance Evaluation Simulations $(n,m,C,diam_C,Avgdeg) = (30,79,11,7,5)$

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.

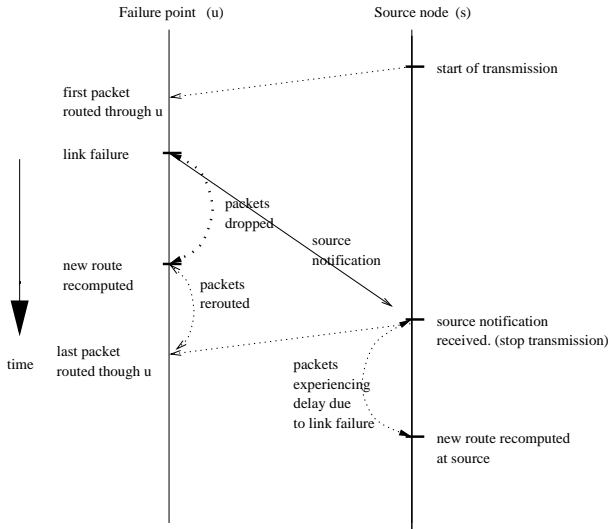This sequence of events is also illustrated in Figure 4.

Fig. 4. Effect of a link failure on an ongoing flow.

Figure 3 shows a 30 node graph that is used for evaluating the performance of CEDAR in the presence of link failures. For an arbitrary flow that transmits 1KB packets at a mean rate of 500 Kbps with Poisson and MMPP (on/off = 0.9/0.1) traffic source, 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.

As can be observed from Tables V and VI, 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 small and hence 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.

VII. RELATED WORK

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

A. 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 presence of hidden and exposed stations.

Ad hoc routing algorithms which provide a single route in response to a route query from a source ([4], [12], [13]), have low overhead but sometimes use sub-optimal and stale routes. [4] uses flooding, in the worst case, for finding routes. [12] considers signal strength as a metric for routing. [13] 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. [3] 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. [14] uses minimum-hop distance-vector routing. To extend routing to larger networks, hierarchical routing has been proposed [15], with either distance-vector routing [16] or link-state routing [17] used within each cluster.

Other shortest-paths routing algorithms incorporate measures of delay or congestion into the path weights [18], but these algorithms usually have some centralized computation of the delay and congestion metrics.

The multipath routing algorithms are more robust than the single route on demand algorithms, at a cost of higher memory and message requirements. In [5], 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 [19] combines the robustness of multipath routing with the low overhead of single route on demand: when node mobility is high, [4] is used; when node mobility is low, [5] is used.

Currently the IETF MANET working group is considering several ad hoc routing proposals such as AODV [6], DSR [4], TORA [5], ZRP [9], [10] etc.

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. The core architecture is similar to the Landmark Hierarchy [16] and also the Viewserver Hierarchy [20]. 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.

B. QoS Routing

QoS routing algorithms can be mainly classified into two categories: distributed ([21], [11], [22], [23]) and centralized ([11], [24], [25]).

Wang and Crowcroft [21] 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 [11] and Pornavalai et. al. [22] 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 [21] propose shortest-widest path. A comparison of shortest-widest, widest-shortest, dynamic alternative and the shortest distance path is presented in [11]. A distributed algorithm for finding delay constrained routes has been proposed by Sun and Langendoerfer [23].

Ma and Steenkiste [11] propose a centralized algorithm for finding the fair share of a best effort flow, which can be used for shortest-widest path, widest-shortest path or any other algorithm for routing the best effort traffic. Effects of uncertain parameters on QoS routing with end-to-end delay requirements is discussed in [24]. For a wide class of probability distributions, Lorenz and Orda [24] and Guerin and Orda [25] 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. [26].

VIII. 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.

References

[1] R. Nair, B. Rajagopalan, H. Sandick, and E. Crawley, "A Framework for QoS-based Routing in the Internet," Internet Draft draft-ietf-qosr-framework-05.txt, May 1998.

[2] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: A medium access protocol for wireless LANs," in *Proceedings of ACM SIG-COMM '94*, London, England, Aug. 1994.

[3] R. Sivakumar, B. Das, and V. Bharghavan, "Spine Routing in Ad Hoc Networks," *ACM/Baltzer Cluster Computing Journal (special issue on Mobile Computing)*, vol. 1, no. 2, Nov. 1998.

[4] J. Broch, D. B. Johnson, and D. A. Maltz, "The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks," Internet Draft draft-ietf-manet-dsr-01.txt, Dec. 1998.

[5] 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*, Apr. 1997.

[6] C. E. Perkins and E. M. Royer, "Ad Hoc On Demand Distance Vector (AODV) Routing," Internet Draft draft-ietf-manet-aodv-02.txt, Nov. 1998.

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

[8] B. Awerbuch, Y. Du, B. Khan, and Y. Shavitt, "Routing Through Networks with Topology Aggregation," in *IEEE Symposium on Computers and Communications*, Athens, Greece, June 1998.

[9] Z. J. Haas and M. R. Pearlman, "The Performance of Query Control Schemes for the Zone Routing Protocol," in *Proceedings of ACM SIG-COMM '98*, Vancouver, British Columbia, Sept. 1998.

[10] Z. J. Haas and M. R. Pearlman, *Ad Hoc Networks*, chapter Providing Ad-Hoc Connectivity with the Reconfigurable Wireless Networks, Addison Wesley, 1999.

[11] Q. Ma and P. Steenkiste, "On Path Selection for Traffic with Bandwidth Guarantees," in *Proceedings of Fifth IEEE International Conference on Network Protocols*, Atlanta, Oct. 1997.

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

[13] 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*, 1996, pp. 480–486.

[14] J. Jubin and J. D. Tornow, "The DARPA packet radio network protocols," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 21–32, Jan. 1987.

[15] N. Shacham and J. Westcott, "Future directions in packet radio architectures and protocols," *Proceedings of the IEEE*, vol. 75, no. 1, pp. 83–99, Jan. 1987.

[16] P. F. Tsuchiya, "The landmark hierarchy: A new hierarchy for routing in very large networks," in *ACM SIGCOMM '88*, Stanford, California, 1988, pp. 35–42.

[17] W. T. Tsai, C. V. Ramamoorthy, W. K. Tsai, and O. Nishiguchi, "An adaptive hierarchical routing protocol," *IEEE Transactions on Computers*, vol. 38, no. 8, pp. 1059–1075, Aug. 1989.

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

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

[20] C. Alaettinoglu and A. U. Shankar, "The Viewserver Hierarchy for Interdomain Routing: Protocols and Evaluation," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1396–1410, Oct. 1995.

[21] Z. Wang and J. Crowcroft, "QoS Routing for Supporting Resource Reservation," *IEEE Journal on Selected Areas in Communications*, vol. 14, no. 7, pp. 1228–1234, Sept. 1996.

[22] C. Pornavalai, G. Chakraborty, and N. Shiratori, "QoS Based Routing

Algorithm in Integrated Services Packet Networks," in *International Conference on Network Protocols*, Atlanta, USA, Oct. 1997.

[23] 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.

[24] D. H. Lorenz and A. Orda, "QoS Routing in Networks with Uncertain Parameters," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, San Francisco, California, Apr. 1998.

[25] R. A. Guerin and A. Orda, "QoS-based Routing in Networks with Inaccurate Information: Theory and Algorithms," in *Proceedings of IEEE Conference on Computer Communications (INFOCOM)*, Kobe, Japan, Apr. 1997.

[26] G. Apostolopoulos, R. Guerin, S. Kamat, and S. K. Tripathi, "Quality of Service Routing: A Performance Perspective," in *Proceedings of ACM SIGCOMM '98*, Vancouver, Canada, Sept. 1998.