CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm

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

Abstract—**CEDAR is an algorithm for QoS routing in ad hoc network environments. It 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 preliminary performance evaluation shows that CEDAR is a robust and adaptive QoS routing algorithm that reacts effectively to the dynamics of the network while still approximating link-state performance for stable networks.

Keywords—**Ad hoc routing, QoS routing inftro.tex**

I. 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. Ad hoc 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*.

Ad hoc networks are dynamic in nature, and transmissions are susceptible to fades, interference, and collisions from hidden/exposed stations. These characteristics make it a challenging task to design a QoS routing algorithm for ad hoc networks. Following are the main design goals for such an algorithm:

1. The algorithm should be highly robust and should degrade gracefully with increasing mobility.
2. Route computation should not require maintenance of global information.
3. The computed route should be highly likely to sustain the requested bandwidth for the flow.
4. The route computation should involve only a few hosts as broadcast in the whole network is expensive.
5. Hosts should have quick access to routes when connections need to be established.

We propose CEDAR as a QoS routing algorithm, which achieves the above design goals 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 hosts 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* host maintains the local topology of the hosts in its domain, and also performs route computation on behalf of these hosts.
- *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. Slow-moving *increase* waves and fast moving *decrease waves* which denote corresponding changes in available bandwidths on links, are used to propogate 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 that 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 and terminology used in the paper. Section III describes the computation and dynamic management of the *core* of the network. Section IV describes 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 and Section VII summarizes the paper.

II. Network Model and Terminology

We assume that all the hosts communicate on the same shared wireless channel. Neighborhood is assumed to be 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 [1] 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

[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.

[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.

(from receiver to sender). Local data broadcasts are not assumed to be reliable.

We assume that each host can estimate the available bandwidth using some link-level mechanisms. We also assume a close coordination between the MAC and routing layers is assumed. In particular, reception of RTS and CTS control messages at the MAC layer is used to improve the behavior of the routing layer, as explained in Section III.

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). Also, every node not in S chooses one of its neighbors who is in S as its *dominator*. 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. Thus, from the definition of the *core*, if G is connected, then a *core* C of G must also be connected (via virtual links).

III. 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 the rest of 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.

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 [2], [3], [4]. 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 [4] and route probes [2], [3] 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 [5]. 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 set up. 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. Our *core* computation algorithm satisfies the above requirements.

B. 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, we are using 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 each 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. u 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. 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*.

C. 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. 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 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.

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.

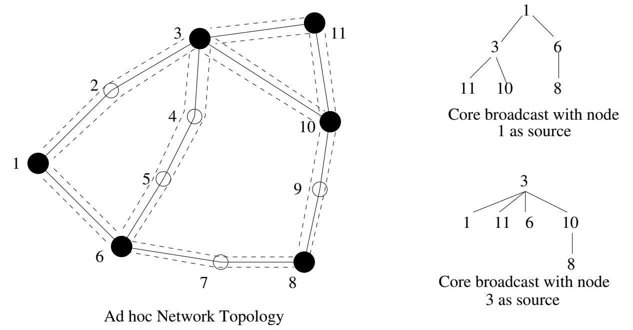

Fig. 1. Example of a *core* broadcast. Nodes in black are *core* nodes. Solid lines denote links in the ad hoc network. Dotted pipes denote virtual links in the *core* graph.

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 *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.
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 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.

IV. QoS State Propagation in CEDAR

Section III 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 an 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.

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 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 ids 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 IV-B.
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)
   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 $(ttl > 0)$,
   $b_{av}(a, b) \leftarrow b(a, b)$
   delete any pending ito/dto message for (a, b)
   from the ito-queue and dto-queue.
   if $(b_{av}(a, b) < b(a, b))$
   add $ito(a, b, dom(a), dom(b), b(a, b), ttl - 1)$
   to the ito-queue.
   else if $(b_{av}(a, b) > b(a, b))$,
   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 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 domina-

tors 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 $dto(a, b, dom(a), dom(b), b(a, b), ttl)$, it is processed in the same way as the *ito* wave is processed in 2 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 (point 2(a) in *decrease wave* propagation). As we have noted before, the increase/decrease waves use the efficient *core* broadcast mechanism for propagation.

B. Issues with increase/decrease waves:

There are three key questions pertaining to the propagation of increase/decrease waves that need to be answered. While finding answers to these questions is still part of ongoing research, we present below some preliminary answers.

- *When should a Increase/Decrease Wave be Generated?* A wave should only be generated when the available bandwidth has changed by a threshold value since the last wave was generated. A simple approach would be to make the threshold a constant system parameter. Another method suggested by [6] uses logarithmic update, which does not wastefully generate increase/decrease waves when the change in link capacity is unlikely to alter the probability of computing admissible routes.
- *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. In other words, the maximum distance that an *increase* wave can travel (i.e. the time-to-live field) is an increasing function of the available bandwidth of the link.
- *How Fast does a Increase/Decrease Wave Propagate?* An increase wave waits for a fixed timeout period (e.g., twice the expected inter-arrival time between the generation of two successive waves for any link) at each node before being forwarded whereas a decrease wave is immediately forwarded to its neighbors. Thus decrease waves move much faster and can kill increase waves for unstable links. The wait-before-forwarding for *increase* waves also naturally leads to the implicit establishment of a source-based breadth-first-search tree for the *core* broadcast described in Section III.

V. QoS Routing in CEDAR

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, 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, 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.

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: $< core\_path\_req, dom(s), d, b, P = null >$.
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)
3. When $dom(t)$ receives the *core* path request message $< core\_path\_req, dom(s), d, b, P >$, 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 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.

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 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 [7].
3. Let t be the end point of the chosen 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 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.

C. 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 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.

VI. Performance Evaluation

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 all our simulations, the notation CEDAR_t stands for a simulation run of CEDAR at time t (note that the amount of remote link information a *core* node has and consequently the optimality of routes it computes increases with time)

A. Performance of CEDAR in a best-effort service environment

We use a randomly generated graph having 30 nodes (Figure 2) for the results in this section. 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 Figure 2. For evaluating CEDAR in a best-effort environment, we measure 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 *increase* waves. The time d used in the tables is the constant time for which *increase* waves are delayed at each hop.

As can be seen from the results, CEDAR performs reasonably well before the introduction of *increase/decrease 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. 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 Table I indicate the near optimal nature of CEDAR with increase in network stability.

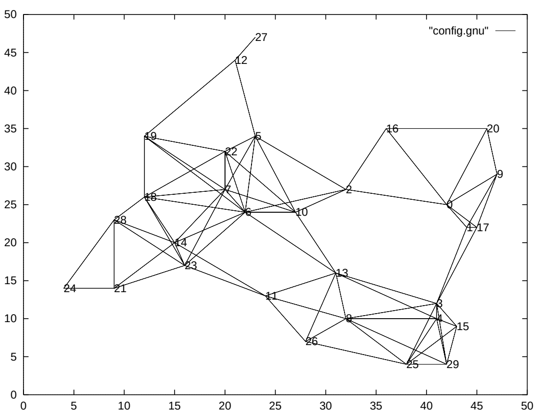

Fig. 2. Graph used for Performance Evaluation Simulations (n,m,C,$diam_C$,$Avgdeg$) = (30,79,11,7,5)

	APL	MC	TC	CU
optimal	2.82	5.65	0.07	N/A
$CEDAR_{t_0}$	3.40	6.82	0.08	0.27
$CEDAR_{t_0+d}$	3.35	6.70	0.08	0.10
$CEDAR_{t_0+2d}$	3.30	6.64	0.08	0.09
$CEDAR_{t_0+3d}$	3.10	6.37	0.07	0.09

TABLE I

PERFORMANCE OF CEDAR COMPARED TO AN OPTIMAL APPROACH.

$time$	src	dst	bw_{req}	h_C	bw_C	h_O	bw_O
$t_0+\delta$	23	11	50	1	100	1	100
$t_0+2\delta$	26	18	30	4	50	4	50
$t_0+3\delta$	21	24	50	1	50	2	100
$t_0+4\delta$	15	17	50	2	50	2	50
$t_0+5\delta$	17	10	30	5	50	4	50

TABLE II

PERFORMANCE OF CEDAR COMPARED TO AN OPTIMAL APPROACH WITH CONNECTION REQUESTS ISSUED AT TIMES SHOWN.

t_s	t_e	s	d	bw_r	acc_O	acc_w	acc_{nw}
0	8	8	13	55	*yes*	*yes*	*yes*
20	31	3	13	55	*yes*	*yes*	*no*
28	38	6	7	16	*yes*	*yes*	*yes*
56	64	16	4	45	*yes*	*no*	*no*
64	74	16	11	23	*yes*	*yes*	*yes*

TABLE III

PERFORMANCE IMPROVEMENT OF CEDAR WITH THE ADVENT OF *increase* AND *decrease* WAVES. THE ACCEPT/REJECT RATIO FOR OPTIMAL, CEDAR WITH WAVES AND CEDAR WITHOUT WAVES ARE 10:0, 9:1 AND 7:3 RESPECTIVELY.

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 *hops*). The time field in Table II 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 *increase* and *decrease* 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 *increase* waves travel from one hop to another with a constant delay, *decrease* waves 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 *increase/decrease* 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 2 with link bandwidths randomly set to either 50 units or 100 units. In the column headers in Table II, h_C, bw_C and h_O, bw_O stand for the hopcount and available bandwidth of routes computed by CEDAR and the optimal algorithm respectively. Note from Table II that CEDAR approximates the optimal algorithm for the scenarios simulated. Further, from Table III, we can see the utility of the *increase* and *decrease* waves to CEDAR. In Table III, the column headers t_s, t_e, s, d and bw_r stand for the start time of the connection, end time of the connection, the source, the destination and the bandwidth requested respectively, while acc_O, acc_w and acc_{nw} represent whether the connection request was accepted in the optimal algorithm, *CEDAR with waves* and *CEDAR without waves* respectively.

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

TABLE IV

PERFORMANCE OF CEDAR'S RECOVERY MECHANISM ON A LINK FAILURE WITH LINK FAILURE ON PATH FOR FLOW FROM NODE 24 TO 20. INPUT TRAFFIC GENERATED USING POISSON DISTRIBUTION

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 inspite of the dynamics of the network topology.

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 (RLP - relative link position - represents the position of the link failure relative to the source) 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 IV, 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 failure point 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.

VII. SUMMARY

Unlike CEDAR, 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 [2], [8] 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. [9] uses a *spine* structure for route computation and maintenance. While it provides optimal or near optimal routes depending upon the nature of information stored in the spine nodes, it 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. [10] 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.

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.

REFERENCES

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

[2] 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.

[3] M. S. Corson and A. Ephremides, "A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks," *ACM/Baltzer Wireless Networks Journal*, vol. 1, no. 1, pp. 61–81, Feb. 1995.

[4] C. E. Perkins and P. Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *Proceedings of ACM SIGCOMM*, London, England, Aug. 1994, pp. 234–244.

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

[6] 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.

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

[8] 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.

[9] 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.

[10] 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*, Jan. 1987, pp. 56–73.