INTERNET-DRAFT October 1998
draft-ietf-manet-cedar-spec-00.txt
Expires in six months

Core Extraction Distributed Ad hoc Routing (CEDAR) Specification

Raghupathy Sivakumar Prasun Sinha Vaduvur Bharghavan
University of Illinois, Urbana-Champaign

Status of this Memo

Abstract

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

Table of Contents

1. Introduction

An ad hoc network is a dynamic multi-hop wireless network that is established by a group of mobile hosts on a shared wireless channel by virtue of their proximity t[o each other. These channels, typically being scarce and dynamic, make it difficult to perform efficient resource utilization or to execute critical applications. Hence, QoS routing, as opposed to non-QoS routing, is desirable in such environments. This draft focuses on a restricted QoS Routing problem: the satisfaction of minimum bandwidth requirements. Of course, since the network is highly dynamic, and transmissions are susceptible to fades, interference, and collisions from hidden/exposed stations, CEDAR cannot provide bandwidth guarantees for the computed routes. Rather, its goal is to provide routes that are highly likely to satisfy the bandwidth requirement of a route, so long as such routes exist [1].

Given the nature of the network and the requirements of the applications, the following are the key goals of CEDAR.

(a) Route computation must be distributed because centralized routing in a dynamic network is impossible even for fairly small networks. (b) Route computation should not involve the maintenance of global state, or even significant amounts of volatile non-local state. In particular, link state routing is not feasible for highly dynamic networks because of the significant state propagation overhead when the network topology changes. (c) As few nodes as possible must be involved in state propagation and route computation (without becoming single points of failure), since this involves monitoring and updating at least some state in the network. On the other hand, every host must have quick access to routes on-demand. (d) Each node must only care about the routes corresponding to its destination, and must not be involved in frequent topology updates for parts of the network to which it has no traffic. (e) Stale routes must be either avoided, or detected and eliminated quickly. (f) Broadcasts must be avoided as far as possible because broadcasts are highly unreliable in ad hoc networks. (g) If the topology stabilizes, then routes must converge to the optimal routes, and (h) It is desirable to have a backup route when the primary route has become stale and is being recomputed.

QoS routing for ad hoc networks is relatively unchartered territory. In addition to the above, we have the following goals for QoS routing in ad hoc networks. (a) Applications provide a minimum bandwidth requirement for a connection, and the routing algorithm must efficiently compute a route that can satisfy the bandwidth requirement with high probability. (b) 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. (c) Dynamic links (either unstable or low bandwidth links) must not cause state propagation throughout the network. Only stable high bandwidth link information must be propagated throughout the network, and (d) The QoS route computation algorithm should be simple and robust. Robustness, rather than optimality, is the key requirement.

In order to achieve the above goals, this draft proposes the Core-Extraction Distributed Ad hoc Routing (CEDAR) algorithm for QoS routing in ad hoc networks. Briefly, CEDAR dynamically establishes a "core" of the network, and then incrementally propagates link state of stable high bandwidth links to the nodes of the core. Route computation is on-demand, and is performed by core hosts using local state only. CEDAR is proposed as a QoS routing algorithm for small to medium size ad hoc networks consisting of tens to hundreds of nodes. CEDAR does not compute optimal routes because of the minimalist approach to state management, but the trade-off of robustness and adaptation for optimality is believed to be well justified in ad hoc networks. The following is a brief description of the three key components of CEDAR.

1. Establishment and Maintenance of a core using Local Core Extraction

CEDAR does core extraction in order to extract a subset of nodes in the network that would be the only ones that perform state management and route computation. The core extraction is done dynamically by approximating a minimum dominating set of the ad hoc network using only local computation and local state. Each host in the core then establishes a virtual link (via a tunnel) to nearby core hosts (within the 3rd neighborhood in the ad hoc network). Each core host maintains the local topology of the hosts in its domain, and also performs route computation on behalf of these hosts. The core computation and core management upon change in the network topology are purely local computations to enable the core to adapt efficiently to the dynamics of the network.

2. Link State Propagation using Increase/Decrease waves

While it is possible to execute ad hoc routing algorithms using only local topology information at the core nodes, QoS routing in CEDAR is achieved by propagating, in the core, the bandwidth availability information of stable links. The basic idea is that the information about stable high-bandwidth links can be made known to core nodes far away in the network, while information about dynamic links or low bandwidth links should remain local. By means of this link state propagation mechanism, CEDAR approximates

a minimalist local state algorithm in highly dynamic networks while it approaches the maximalist link state algorithm in highly stable networks. The propagation of link-state is achieved through slow-moving 'increase' waves (which denote increase of bandwidth) and fast-moving 'decrease' waves (which denote decrease of bandwidth), which traverse the core. The key questions to answer in link state propagation are: when should an increase/decrease wave be initiated, how far should a wave propagate, and how fast should a wave propagate.

3. Route Computation

Route computation first establishes a core path from the domain of the source to the domain of the destination. This initial phase involves probing on the core, and the resultant core path is cached for future use. The core path provides the directionality of the route from the source to the destination. Using this directional information, CEDAR iteratively tries to find a partial route from the source to the domain of the furthest possible node in the core path (which then becomes the source for the next iteration) which can satisfy the requested bandwidth, using only local information. Effectively, the computed route is a concatenation of the shortest-widest-furthest (the least hop path among the maximum bandwidth paths) paths found locally using the core path as the guideline.

Several algorithms have been proposed for routing in ad hoc networks. The ad hoc routing algorithms proposed in [2,3,4] provide a single route in response to a route query from a source. Previous work on tactical packet radio networks had led to many of the fundamental results in ad hoc networks. [3] 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 core infrastructure. The multipath routing algorithms proposed in [6,7,8] are more robust than the single route on demand algorithms, at the cost of higher memory and message requirements.

2. Network Model

This section describes the network model and the terminology used in this draft.

Network Model

CEDAR assumes that all the hosts communicate on the same shared

logical wireless channel. CEDAR assumes that each transmitter has a fixed transmission range, and that neighborhood is a commutative property (i.e. if A can hear B, then B can hear A). Because of the local nature of transmissions, hidden and exposed stations abound in an ad hoc network. CEDAR assumes the use of a CSMA/CA like algorithm for reliable unicast communication, and for solving the problem of hidden/exposed stations. Essentially, data transmission is preceded by a control packet handoff, and the sequence of packets exchanged in a communication is the following: RTS (from sender to receiver) - CTS (from receiver to sender) - Data (from sender to receiver) - ACK (from receiver to sender). The RTS and CTS packets avoid collisions from the exposed stations and the hidden stations respectively. Local broadcasts are not guaranteed to be reliable (because it is unreasonable to expect a CTS from every receiver before commencing data transmission), and are typically quite unreliable due to the presence of hidden and exposed stations.

CEDAR assumes small to medium networks ranging upto to hundreds of hosts. For larger networks, we believe that a clustering algorithm [6] can be used to reduce the cluster size and apply CEDAR hierarchically within each cluster, for a cluster of clusters, etc. CEDAR also assumes that mobility and extended fades are the main causes of link failures and topology changes. We assume that the change in network topology is frequent, but not frequent enough to render any sort of route computation useless. Note that CEDAR only cares about the relative mobility of the hosts, not the absolute mobility of the hosts. In particular, even if all the hosts are moving, the ad hoc network would be considered to be stable so long as the neighborhood of each host does not change.

As in most QoS routing algorithms, CEDAR assumes that the MAC/link layer can estimate the available link bandwidth. Because all the hosts in a region share the same channel, each host must share the link bandwidth with the hosts in its second neighborhood [7]. In related work on providing QoS in wireless channels, a mechanism is provided for each host to fairly access a shared channel, and claim at least B/N bandwidth, where B is the effective channel bandwidth and N is the number of hosts locally contending for the bandwidth [8]. This work is currently being extended to ad hoc networks. While details of bandwidth sharing and estimation are beyond the scope of this draft, it is assumed that each host can estimate the available bandwidth of its links using some link-level mechanisms.

CEDAR assumes a close coordination between the MAC layer and the routing layer. In particular, it uses the reception of RTS and CTS control messages at the MAC layer in order to improve the behavior

of the routing layer, as explained in Section 3.

Finally, bandwidth is the QoS parameter of interest in this draft. 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.

Graph Terminology

The ad hoc network is represented 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, Ni(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 Ni[x] of node x is N(i) U {x}.

A dominating set S (a subset of 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. The draft uses the term tunnel interchangeably with virtual link.

Given an MDS Vc of graph G, define a core of the graph C=(Vc, Ec), where Ec = { [u,v] u in Vc, v in Vc, u in N3(v) } . Thus, the core graph consists of the MDS nodes Vc, and a set of virtual links between every two nodes in Vc that are within a distance 3 of each other in G. Two nodes u and v which have a virtual link [u,v] in the core are said to be "nearby" nodes.

For a connected graph G, consider any dominating set S. If the diameter of G is greater than 2, then for each node v in S, there must be at least one other node of S in N3(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).

3. CEDAR Architecture and the Core

This section focuses on the establishment and maintenance of the core. Briefly, CEDAR extracts the core of the ad hoc network by approximating the minimum dominating set (MDS) of the ad hoc network. The nodes in the MDS comprise the core nodes of the network. Each core node establishes a unicast virtual link (via a tunnel) with nearby core nodes that are a distance of 3 or less away from it in the ad hoc network. This guarantees that the core is connected so

long as the network is connected. The core nodes then collect local topology information and information about stable high-bandwidth links far away and perform routing for the nodes in their domain (or immediate neighborhood). Each node that is not in the core chooses a core neighbor as its dominator, i.e. the node which performs route computations on its behalf. The core is merely an infrastructure for facilitating route computation, and is itself independent of the routing algorithm. In particular, it is possible to use any of the well known ad hoc routing algorithms such as DSR [2], LMR [4], TORA [5], DSDV [9], etc. in the core graph.

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

3a. Rationale for a Core-based Architecture in CEDAR

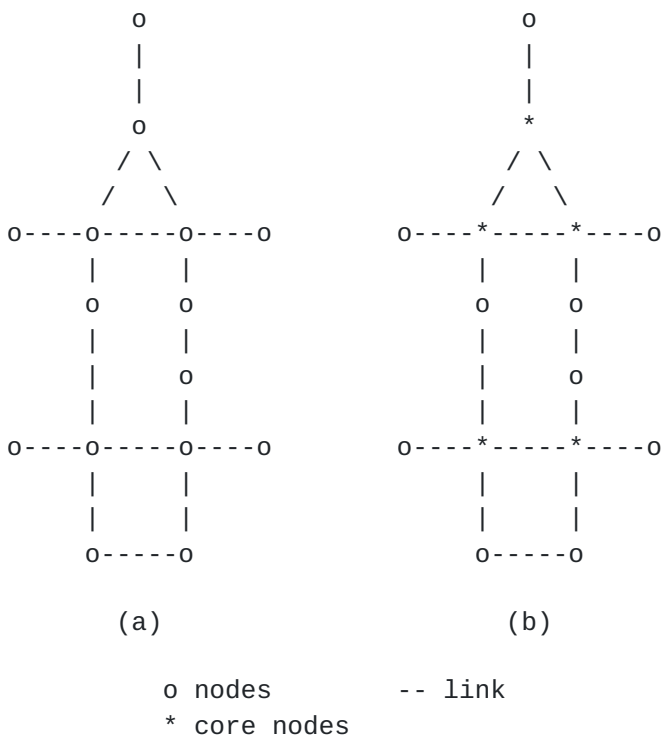
Many contemporary proposals for ad hoc networking require every node in the ad hoc network to perform route computations and topology management [2,6,19,20]. However, CEDAR uses a core-based infrastructure for QoS routing due to two compelling reasons.

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. The problem of local broadcasts in ad hoc networks has also been a recent subject of discussion in the MANET working group. Topology information propagation [19,20] and route probes [2,6] are inevitable in order to establish routes and will, of necessity, need to be broadcast if every node performs route computation. On the other hand, if only a core subset of nodes in the ad hoc network perform route computations, it is possible to set up reliable unicast channels between nearby core nodes and accomplish both the topology updates and route probes much more effectively.

The issues with having only a core subset of nodes performing

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



(a) (b)

Figure 1 : (a) The example network (b) The example network with core nodes chosen

3b. Generation and Maintenance of the Core in CEDAR

Ideally, the core comprises of the nodes in a minimum dominating set Vc of the ad hoc network G=(V,E). While a greedy algorithm can be used to generate the best known approximation for the MDS, CEDAR uses 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 N1(u), degree d(u) = |N1(u)|, dominator dom(u), and effective degree d*(u), where d*(u) is the number of nodes in the closed neighborhood N1[u], who have chosen u as their dominator. The core computation algorithm, which is performed periodically, works as follows at node u.

1. u broadcasts a beacon which contains the following information pertaining to the core computation:

u	d*(u)	d(u)	dom(u)

2. u sets dom(u) <- v, where v is the node in N1[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)) : for all w in N1(u)}

Upon reception of the message, v increments d*(v) if u is not already in v's dominated list.

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 N3(u). A piggybacked broadcast is accomplished as follows. In its beacon, u transmits a message:

u	DOM	3	path_traversed=null

When node w hears a beacon that contains a message <u, DOM, i, path_traversed>, it piggybacks the message

u	DOM	i-1	path_traversed+w

in its own beacon if (i-1 > 0). Thus, the piggybacked broadcast of a core node advertises its presence in its third neighborhood. As mentioned in Section 2, this guarantees that each core node identifies its nearby core nodes, and can set up virtual links to these nodes using the path_traversed field in the broadcast messages. The state that is contained in a core node u is the following:

1. its nearby core nodes (i.e. the core nodes in N3(u))
2. N*(u), the nodes that it dominates
3. for each node v in N*(u), <forall w in N1(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. Figure 1 shows an example network and the corresponding core for the network.

Maintaining the core in the presence of network dynamics is very simple as the periodic computation of the original core computation algorithm ensures nomination of an appropriate dominator for each node that loses connectivity with its dominator during mobility.

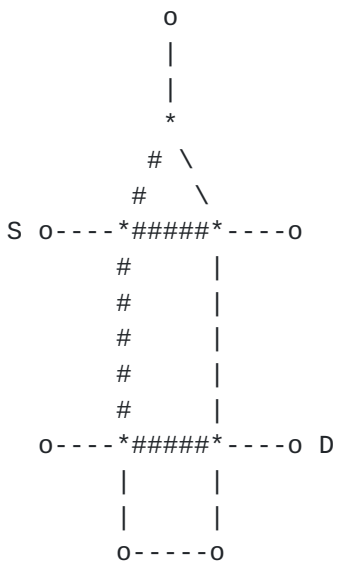
3c. Core Broadcast and its Application to CEDAR

As mentioned earlier, broadcasts do not work well in an ad hoc network (because of hidden and exposed stations). Hence, CEDAR uses a unicast based mechanism for achieving a 'core broadcast'. Note that it is reasonable to assume a unicast based mechanism to achieve broadcast in the core, because each core node is expected to have few nearby core nodes. Besides, the core broadcast mechanism ensures that each core node does not transmit a broadcast packet to every nearby core node, as described before. 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 an 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, it is assumed 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 to u. If u and v are a distance 3 apart, using the same argument there

will be 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.



S:source; D:destination; #:tunnels used in the core broadcast

Figure 2 : A core broadcast initiated by dom(S) for finding a route to D

Note that the core broadcast has the following properties:

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

2. The number of messages is O(|V|) in the worst case, and O(|Vc|) in the average case. In particular, the only case extra messages are transmitted 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 explicit recomputation of the implicitly generated source tree. 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 the core broadcast in CEDAR has 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. Figure 2 illustrates a core broadcast in an example network. Notice that all tunnels need not be used for core broadcast, as the core broadcast dynamically establishes a source-based tree, as mentioned above.

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

4. QoS State Propagation in CEDAR

Section 3 described the core routing infrastructure of CEDAR. Since each core node uses only the locally cached state to compute the shortest-widest furthest path along the core path in the route computation phase, the focus is now turned 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 may compute 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. CEDAR achieves the goal of having stability and bandwidth based link-state propagation using increase and decrease waves, as described in this section.

In the rest of this section, the draft first describes the mechanics of the increase and decrease waves, and then answers 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.

4a. Increase and Decrease Waves

For every link l=(a,b), the node b is responsible for monitoring the available bandwidth on l and informing a of the same if l is bi-directional. b and a in turn notify their respective dominators for initiating the increase or decrease waves, when the bandwidth changes by some threshold value. These waves are then propagated by the dominators (core nodes) to a subset of 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 bav(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 4c.

2. When a core node u receives an ito wave

ito(<a, b, dom(a), dom(b), b(a, b), ttl>),

```
1       if u has no state cached for (a,b),
2          bav(a,b) <- b(a,b)
3          if (ttl > 0), then add the following message
                to the ito-queue:
4             ito(<a, b, dom(a), dom(b), b(a,b), ttl - 1>)
5       else if u has cached state for (a,b) and (ttl > 0),
6          if (bav(a,b) < b(a,b))
7             bav(a,b) <- b(a,b)
8             delete any pending ito/dto message for (a,b) from the
9                ito-queue and dto-queue.
10            add the following message to the ito-queue:
11               ito(<a, b, dom(a), dom(b), b(a,b), ttl - 1>)
12         else if (bav(a,b) > b(a,b)),
13            bav(a,b) <- b(a,b)
14            delete any pending ito/dto message for (a,b) from the
15                    ito-queue and dto-queue.
16            add the following message to the dto-queue:
17               dto(<a, b, dom(a), dom(b), b(a,b), ttl - 1>)
18      else if u has cached state for (a,b) and (ttl = 0),
19         bav(a,b) <- b(a,b)
20         delete any pending ito/dto message for (a,b) from the
                ito-queue and dto-queue.
21         add the following message to the dto-queue:
22              dto(<a, b, dom(a), dom(b), 0, infinity>)
```

The ito-queue and the dto-queues are flushed periodically, depending on the speed of propagation of the increase/decrease waves.

The following is the sequence of actions for a decrease wave:

1. When a link l=(a,b) goes down, or when the available bandwidth b(a, b) decreases beyond a threshold value, then the two end-points of l inform their dominators for initiating a core broadcast for a decrease wave:

dto(<a, b, dom(a), dom(b), b(a,b), ttl(b)>),

where dto (decrease to) denotes the type of the wave, and the other parameters are as defined before.

2. When a core node u receives a dto wave

dto(<a, b, dom(a), dom(b), b(a,b), ttl>),

```
1       if u has no state cached for (a,b) and (b(a,b) = 0),
2          the wave is killed.
3       else if u has no state cached for (a,b) and (b(a,b) > 0),
4          bav(a,b) <- b(a,b)
5          if (ttl > 0), then add the following message
                to the ito-queue:
6             ito(<a, b, dom(a), dom(b), b(a,b), ttl - 1>)
7       else if u has cached state for (a,b) and (ttl > 0),
8          if (bav(a,b) < b(a,b)),
9             bav(a,b) <- b(a,b)
10            delete any pending ito/dto message for (a,b) from the
11                  ito-queue and dto-queue.
12            add ito(<a, b, dom(a), dom(b), b(a,b), ttl - 1>) to
13                  the ito-queue.
14         else if (bav(a,b) > b(a,b)),
15            bav(a,b) <- b(a,b)
16            delete any pending ito/dto message for (a,b) from the
17               ito-queue and dto-queue.
18            add the following message to the dto-queue.
19               dto(<a, b, dom(a), dom(b), b(a,b), ttl - 1>)
20      else if u has cached state for (a,b) and (ttl = 0),
21           bav(a,b) <- b(a,b)
22           delete any pending ito/dto message for (a,b) from the
23              ito-queue and dto-queue.
24           add the following message to the dto-queue.
25              dto(<a, b, dom(a), dom(b), 0, infinity>)
```

There are several key 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, infinity>) 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, infinity>) 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 (line 2 in decrease wave propagation). The increase/decrease waves use the efficient core broadcast mechanism for propagation. Figure 3 illustrates a decrease wave cancelling a previously generated increase wave for a link l.

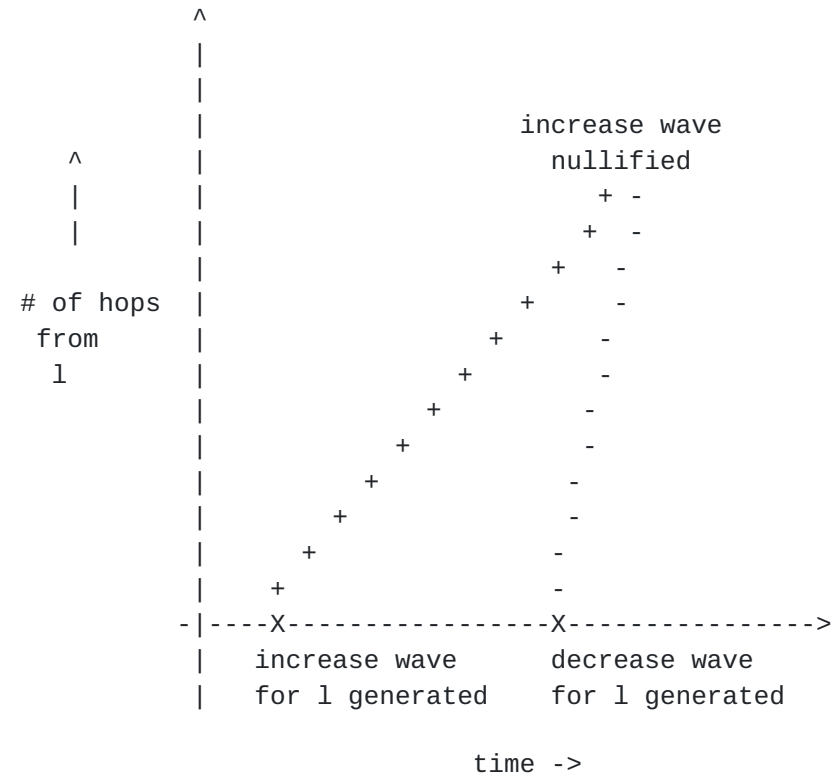


Figure 3 : A decrease wave cancelling an increase wave for l

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

4b. Issues in link state propagation

In this subsection, the three key questions pertaining to the propagation of increase/decrease waves are discussed: when should a wave be generated, how fast should a wave propagate, and how far should a wave propagate.

When is a wave generated ?

To avoid a large overhead, CEDAR generates waves only when the bandwidth has changed by some threshold value. We suggest the use of a constant threshold when the bandwidth request sizes are comparable to the available bandwidth and a logarithmic scale [10] for the threshold when the typical request sizes are an order of a magnitude less than the available bandwidths. The advantage of the logarithmic update is that it does not wastefully generate increase/decrease waves when the change in link capacity is unlikely to alter the probability of computing admissible routes. Further work is needed to substantiate the above heuristics.

How Far does a Increase/Decrease Wave Propagate?

The goal is to propagate link bandwidth information to a number of nodes that is proportional to the amount of bandwidth being propagated. The motivation for this approach is the fact that every node that has knowledge about a particular link would potentially contend for the link, and a higher percentage of requests can be satisfied if the contention on a link is proportional to its bandwidth. Hence we suggest that the maximum distance that the link state travels (time to live - ttl) be an increasing function of the available bandwidth of the link. Although the current CEDAR simulation uses a linear function of the available bandwidth for computing the ttl, a fluid model analysis of an ad hoc network suggests that in general, the ttl should be a function of b^(1/k), where k is a small number between 1 and 3.

How Fast does a Increase/Decrease Wave Propagate?

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

increase waves for unstable links.

5. QoS Routing in CEDAR

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

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

5a. Establishment of the Core Path

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

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

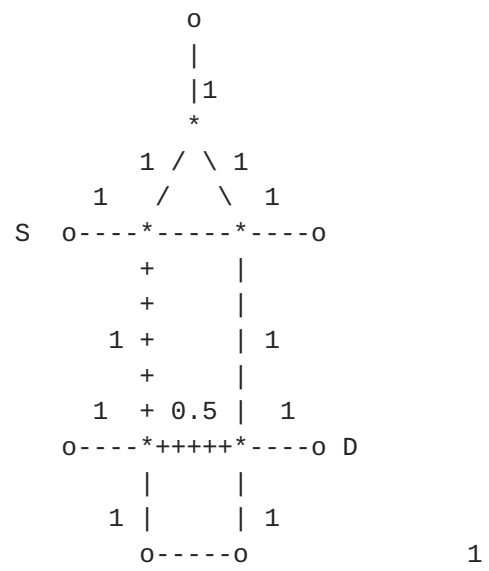
2. When a core node u receives the core path request message <core_path_req, dom(s), d, b, P>, it appends u to P, and forwards the message to each of its nearby core nodes (according to the core broadcast algorithm) to whose domain there exists atleast one path (from u's domain) satisfying bandwidth b.

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

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

Note that by virtue of the core broadcast algorithm, the core path

request traverses an implicitly (and dynamically) established source routed tree from dom(s) which is typically a breadth-first search tree. Thus, the core path is approximately the shortest admissible path in the core graph from dom(s) to dom(d), and hence provides a good directional guideline for the QoS route computation phase. Figure 4 shows an example for a core path. The example assumes that the link marked with 0.5 has an available bandwidth of 0.5 units, whereas all other links have 1 unit of bandwidth available. The route request has a QoS requirement of 1 unit.



+ core path S source D destination

Figure 4 : Core path from dom(S) to dom(D)

5b. QoS Route Computation

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

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

2. Among all the admissible paths (known using local state) to the domain of the furthest possible core node in the core path, dom(s) picks the shortest-widest path using a two phase Dijkstra's algorithm [11]. The first phase is used to find the

available bandwidth B of the widest path. In the subsequent phase, links with available bandwidth less than B are eliminated before computing the shortest path in the resulting graph.

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

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

5. Eventually, either there is an admissible path to d or the local route computation will fail to produce a path at some core node. The concatenation of the partial paths computed by the core nodes provides an end-to-end path that can satisfy the bandwidth requirement of the connection with high probability. Figure 5 shows how the core path found in Figure 4 is used to find a QoS route satisfying the 1 unit bandwidth request. As mentioned earlier, all links except the one indicated with 0.5, have an available bandwidth of 1 unit. This example also illustrates that the QoS route found in CEDAR can be non-optimal as it uses a core path as the guiding direction (the path along the core has 7 hops whereas there is another feasible path - not along the chosen core path - with 6 hops).

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.

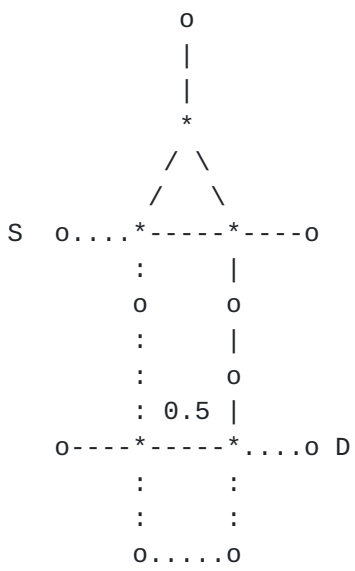


Figure 5 : QoS route from S to D satisfying a bandwidth requirement of 1 unit.

5c. 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 or by a reduction in available bandwidth on that link such that the connection can no longer be served). 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 5b. Once the route is recomputed, u updates the source route in all packets from s to t accordingly. If the link failure happens near the destination, then dynamic route recomputation at the intermediate node works very well because the route recomputation time to the destination is expected to be small, and packets in-flight are re-routed seamlessly.

2. QoS Route Recomputation at the Source: Consider that a link (u, v) fails on the path of an ongoing connection from s to t. The node nearest to the sender, u, then notifies s that the link (u, v) has failed. Upon receiving the notification, u stops its packet transmission, initiates a QoS route computation as in Section 5b, and resumes transmission upon the successful re-establishment of an admissible route. If the link failure happens near the source, then source-initiated recomputation is effective, because the source can quickly receive the link-failure notification and temporarily stop transmission.

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

6. Performance Results

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 Photonics (Data Technology) 1 Mbps 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 proof of concept and has exposed some difficulties in implementing CEDAR, our detailed performance evaluation [12] has been using a simulator that faithfully implements the CEDAR algorithms.

While the entire gamut of results obtained from the tests are not presented due to space constraints, the rest of this section briefly summarizes the performance of CEDAR as observed from these tests. For tests in a best-effort environment we assume the optimal performance to be the performance of a global algorithm that does shortest path computation, while in a QoS environment we assume this to be the performance of a global algorithm doing a shortest widest path computation. The metrics used for comparison in these results are: (i) stretch - the ratio of the number of hops in a route computed by CEDAR to the number of hops in a route computed by the global algorithm, (ii) bandwidth - the ratio of bandwidth available on the routes computed by CEDAR to that of the global algorithm, (iii) message complexity, (iv) time complexity and (v) crankbacks - the ratio of the number of rejects to the number of connection requests.

In a best-effort environment, CEDAR performs reasonably well before the introduction of ito/dto waves, and progressively converges to a near optimal performance once these waves are introduced. In particular, for dynamic networks we observed a stretch of around 1.2 before the waves were introduced. The stretch came down to 1.1 once the waves were introduced. For stable networks, the stretch observed was 1. Additionally, the message and time complexities of CEDAR were also comparable to the optimal performance [12].

In a QoS environment, CEDAR was compared to the optimal algorithm in terms of the number of hops and the bandwidth available on the computed path. In terms of bandwidth, CEDAR's performance was worse than the optimal performance by an average of 3%. For the number of hops on the computed path, CEDAR in fact performed better in some cases (the global algorithm could pick a longer path with a higher bandwidth). When the number of crank-backs was observed in these tests, CEDAR had 30% more crank-backs than the optimal algorithm before the introduction of waves, while after the introduction of ito/dto waves, the number of crank-backs were the same in both CEDAR and the optimal algorithm.

For detailed performance results, please refer to [12].

7. Conclusions

This draft presents 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 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. 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.

CEDAR is a robust and adaptive algorithm that reacts quickly and efficiently to the dynamics of the network while still approximating link-state performance for stable networks. Our simulations show that CEDAR produces good stable admissible routes with a high probability if such routes exist. Furthermore, CEDAR does not require high maintenance overhead even for highly dynamic networks. Ongoing work

on CEDAR is focusing on three areas. (a) While it is shown that CEDAR is effective for small to medium size networks, work is being done on a hierarchically clustered version of CEDAR that can provide QoS routing in large ad hoc networks. (b) While this draft has only considered bandwidth as the QoS parameter in this work, current work is extending CEDAR to include delay as a QoS parameter and (c) The heuristics mentioned in Section 4 need more study and are in the process of being refined.

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

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

[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, April 1997.

[6] 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.

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

[8] S. Lu, V. Bharghavan, and R. Srikant. "Fair queuing in wireless packet networks". In Proceedings of ACM SIGCOMM '97, Cannes, France, September 1997.

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

[10] B. Awerbuch, Yi Du, B. Khan, and Y. Shavitt. "Routing through networks with topology aggregation". In IEEE Symposium on Computers and Communications, Athens, Greece, June 1998.

[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, October 1997.

[12] R. Sivakumar, P. Sinha and V. Bharghavan. "CEDAR: a Core-Extraction Distributed Ad hoc Routing algorithm". TIMELY Group Technical Report,http://www.timely.crhc.uiuc.edu/Papers/cedar.ps

9. Author's Addresses

Raghupathy Sivakumar
458 C&SRL, Coordinated Science Lab
University of Illinois, Urbana-Champaign
1308 W. Main St., Urbana, IL 61801
USA
Email: sivakumr@timely.crhc.uiuc.edu

Prasun Sinha
458 C&SRL, Coordinated Science Lab
University of Illinois, Urbana-Champaign
1308 W. Main St., Urbana, IL 61801
USA
Email: prasun@timely.crhc.uiuc.edu

Vaduvur Bharghavan
457 C&SRL, Coordinated Science Lab
University of Illinois, Urbana-Champaign
1308 W. Main St., Urbana, IL 61801
USA
Email: bharghav@timely.crhc.uiuc.edu