An Adaptive QoS Routing Protocol with Dispersity for Ad-hoc Networks

Youngki Hwang
Syracuse University

Pramod Varshney
Syracuse University, varshney@syr.edu

An Adaptive QoS Routing Protocol with Dispersity for Ad-hoc Networks*

Youngki Hwang and Pramod Varshney, *Fellow*, IEEE
youngki@ecs.syr.edu, varshney@syr.edu
Department of Electrical Engineering and Computer Science
Syracuse University, 121 Link Hall, Syracuse, NY 13244

Abstract

In this paper, we present the design and simulation of a bandwidth constrained multiple path on-demand routing protocol for ad-hoc networks to support end-to-end Quality of Service, which is known as the Adaptive Dispersity QoS Routing (ADQR) Protocol. We propose a route discovery algorithm to find multiple disjoint paths with longer-lived connections, where each path also specifies associated network resource information. This provides an efficient approach for network resource reservation combined with data dispersion and for route maintenance. A longer-lived path generally involves less route maintenance, resulting in an increase of bandwidth utilization. We propose a route maintenance algorithm to proactively monitor network topology changes, providing an interface to network resource management protocols. We also propose a fast rerouting algorithm to significantly reduce data flow and QoS disruptions. Rerouting is proactively carried out before path unavailability occurs. Simulation results show that ADQR reduces data disruptions and provides end-to-end QoS, by employing efficient route discovery and maintenance mechanisms with network resource information.

1 Introduction

In recent years, the need to provide support for multimedia services has raised certain research issues concerning *Quality of Service* (QoS) routing. This is desirable since each multimedia service requires different QoS. The QoS requirement is defined as a set of constraints to be met by a network while in communication on performance metrics, such as bandwidth, delay or delay jitter. A QoS routing protocol finds a feasible path that has sufficient residual network resources to satisfy the QoS constraints of a requested connection. The ultimate goal of QoS routing is to optimize network resource utilization. Generally, QoS routing is used in conjunction with some form of resource reservation mechanism. Thereby, data are transmitted along the same path with guaranteed network resources as long as the connection is available. However, it is difficult to provide QoS guarantees in ad-hoc networks. It is because of the dynamic nature of ad-hoc networks which makes the available state information inherently imprecise. In order to support end-to-end QoS in ad-hoc networks, the routing protocol needs to react promptly to frequent network topology and resource changes that occur due to node mobility and link outages, and also needs to work efficiently with a resource management protocol.

*This work is supported in part by the National Science foundation under grant number ECS-9901361.

For successful communications in ad-hoc networks, a routing protocol should deal with the typical characteristics of these networks, such as limited bandwidth, high error rates, limited power supply and node mobility. In addition, QoS consideration is a must while providing multimedia services in these networks. Therefore, a routing algorithm designed to support QoS for an ad-hoc network should have the following properties :

- Loop free routes in order to provide reliable end-to-end communication.
- Low overhead route discovery and maintenance mechanisms to increase network resource utilization, in terms of delay and bandwidth.
- Less power consumption with less control packets.
- Multiple disjoint routes from a source to destination with network resource information to provide effective data transfer and end-to-end QoS.
- Fast rerouting mechanism to avoid data flow and QoS disruptions.

Based on the properties to be considered as given above, we have recently proposed a new routing protocol for ad-hoc networks namely the Signal Power Adaptive Fast Rerouting (SPAFAR) protocol [?]. We showed that SPAFAR attains improved performance with efficient route discovery and maintenance mechanisms. In this paper, we present a bandwidth constrained QoS routing protocol extended from SPAFAR. Our protocol is known as *Adaptive Dispersity QoS Routing* (ADQR). ADQR is a source-initiated on-demand routing protocol consisting of two phases namely the route discovery phase and the route maintenance phase, and focuses on the inclusion of the last two properties as described below.

1.1 Multiple Disjoint Paths

Multiple path routing has previously been proposed in ad-hoc networks [?][?][?][?][?][?][?][?]. Even though multiple paths were provided in the previous algorithms, they need not be disjoint in most of the algorithms, intermediate links are shared by multiple paths, and only one of the paths is used at one time. Multiple disjoint paths are important in that they can provide dispersed resource reservation and data dispersion. This results in an increase in the effective bandwidth, and reduction of congestion and the probability of a dropped packet in a network [?]. A service can be provided over multiple paths even though none of the multiple paths can meet the QoS requirements individually. In this paper, we propose a route discovery algorithm to find multiple disjoint paths, to reserve the network bandwidth and to disperse data over the reserved paths. Resource reservation needs to be applied on multiple paths, if none of the paths individually meets the QoS requirements. A path which consists of links between physically closer nodes is selected for data transfer. This is because it is more likely to result in a longer-lived connection and to involve less route maintenance. Signal strength is used for route discovery.

1.2 Fast Rerouting

Rerouting is needed whenever an established path is broken. When a path is broken, data transfers and QoS are disrupted until a new path is established. If multiple paths are provided, the delay in finding an alternate path is significantly reduced [?][?]. In existing algorithms, a rerouting mechanism is initiated when a broken path is detected, even though multiple paths have already been found. Therefore, these algorithms still suffer from some delays while rerouting. In the process of rerouting, QoS-aware data are dropped in [?] or are transmitted as best-effort data in [?], which results in QoS disruption. Redundant data transfer over multiple paths to decrease QoS disruption such as proposed in [?] may significantly decrease bandwidth utilization in ad-hoc networks. In this work, we propose a fast rerouting algorithm in conjunction with resource reservation to significantly reduce data flow and QoS disruptions. Fast rerouting is supported by effective monitoring of network topology status. We also propose an effective network monitoring mechanism. Rerouting is done before path unavailability occurs. Signal strength is used for route maintenance.

2 The Problem Statement

2.1 Network Model

A network is modeled as a graph $G = (V, E)$, where V is a finite set of nodes and E is a set of links. Each mobile node $i \in V$ has a unique ID and moves around arbitrarily. A radius R defines a coverage area within which each node can communicate with each other directly. Neighbors of a node i, N_i are defined as a set of nodes $j \neq i$, which are within the radius R and reachable directly from the node i. It is assumed that every pair of neighbors can communicate with each other in either direction. Therefore, there exists a link $l_{i,j} \in E$ between neighbors i and j. A link $l_{i,j}$ may appear and disappear frequently due to node mobility and channel impairments. A path from a source node $i \in V$ to a destination node $j \in V$, $p(i, j)$ is defined as a sequence of intermediate nodes, such that $p(i, j) = \{i, \ldots, j\}$ without loops. $P(i, j)$ is defined as the set of all possible disjoint paths from i to j, such that $P(i, j) = \{p(i, j)\}$.

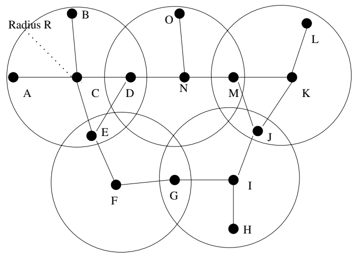

Figure 1. An ad-hoc network model

For illustration, consider the ad-hoc network shown in Figure **??**, where the set of nodes V={A,B,C,D,E,F,G,H, I,J,K,L,M,N,O}, the neighbors of the node C, N_C= {A,B,E,D}, the set of all possible disjoint paths $P(C, J)$= $\{\{C, D, N, M, J\}, \{C, E, F, G, I, J\}\}$.

2.2 QoS Metrics

Let $B(l_{i,j})$ be the available bandwidth of the link between node i and j. A possible path from a source node i to a destination node m may include several intermediate nodes, i.e. $p(i, m) = \{i, j, k, \ldots, l, m\}$. The available bandwidth of the path, $B(p(i, m))$ is defined as,

$$B(p(i, m)) = min\{B(l_{i,j}),\ B(l_{j,k}),\ \ldots,\ B(l_{l,m})\}$$

2.3 The Routing Problem

Bandwidth constrained multiple path routing : Given a network $G = (V, E)$, available bandwidths $B(l_{i,j})$ for each link $l_{i,j} \in E$, the source node S, a destination node D, the set of all possible disjoint paths from the source to

destination $P(S, D) = \{p_1, \ldots, p_n\}$, and a bandwidth constraint δ, where δ is the minimum bandwidth which a connection needs,
(1)find a path $p_k \in P(S, D)$, where $B(p_k) \geq \delta, 1 \leq k \leq n$. If no solution to (1) exists, then
(2)find multiple paths $P'(S, D) \subseteq P(S, D)$, where $\sum B(p_k) \geq \delta,\ \forall p_k \in P'(S, D),\ 1 \leq k \leq n$.

The criteria for the selection of the paths is discussed in the next section.

3 The QoS Routing Protocol

3.1 Overview

• Two Thresholds
Using the minimum signal strength, denoted by S_R, that results in the satisfactory reception of a packet from a neighbor, ADQR determines the maximum transmission range R to the neighbor. ADQR defines two other threshold values in terms of signal strengths as showin in Figure **??**, Th_1 and Th_2 associated with the neighbor as follows.

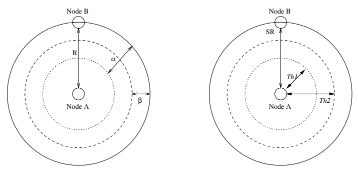

Figure 2. Two thresholds

- The first threshold value for a node, $Th_1 = S_{(R-\alpha)}$
- The second threshold value for a node, $Th_2 = S_{(R-\beta)}$ where, $0 \leq \alpha, \beta \leq R, \quad \beta \leq \alpha \quad and \quad S_R \leq Th_2 \leq Th_1$. Details on how these thresholds are used for routing decision will be defined later in this section.
• Sets, Classes
ADQR defines three different sets; *node*, *link* and *route*.

The node set is divided into three classes. A node whose received signal strength from a neighbor is above the first threshold Th_1 belongs to the *first node class* of the neighbor. A node whose received signal strength from a neighbor is between the first threshold Th_1 and the second threshold Th_2 belongs to the *second node class* of the neighbor. Finally the remaining nodes whose received signal strength is between the second threshold Th_2 and S_R from a neighbor belong to the *third node class* of the neighbor. Therefore, each node can belong to more than two classes at the same time depending on the neighbors.

The link set is divided into three classes. A link between two nodes belongs to the *first link class*, if the nodes belong to the first node class with respect to each other. A link between two nodes belongs to the *second link class*, if the nodes belong to the second node class with respect to each other. A link between two nodes belongs to the *third link class*, if the nodes belong to the third node class with respect to each other.

The route set is divided into three sub classes. A route consists of one or more links between a source and a destination node. For a better understading of this paper, let us define the weakness of a link. That is, a link which belongs to the *third link class* is defined to be weaker than a link which belongs to the *second link class* or *first link class*. Likewise, a link which belongs to the *second link class* is defined to be weaker than a link which belongs to the *first link class*. Therefore, a link which belongs to the *third link class* is defined as a weakest link of a route. A route belongs to the *first route class*, if the weakest link of the route belongs to the first link class. A route belongs to the *second route class*, if the weakest link of the route belongs to the second link class. A route belongs to the *third route class*, if the weakest link of the route belongs to the third link class.
• Neighbors_Table and Routing_Table
Each node of the ad-hoc network keeps a *Neighbors_Table* (NT) which has an updated list of its neighbors, updated signal strength received and the minimum signal strength that results in the satisfactory reception of a packet from the neighbors. Received signal strength from a neighbor is updated based on a cumulative averaging approach such as : $NewSignalStrength, SS_{new} = \delta \times SS_{old} + (1 - \delta) \times SS_{new}$ where, δ can be adjustable depending on the network condition. Two threshold values (Th_1 and Th_2) associated with each node can be obtained from the minimum signal strength for each node. The NT can be easily obtained by periodic broadcasts of the link layer BEACON. Based on the history of accumulated received signal strength from a neighbor, the information of relative direction to the neighbor is kept in the NT. That is, '+' means that the neighbor node is getting away, while '-' means that the neighbor node is getting closer.

Each node also keeps a *Routing_Table* (RT) which has an updated list of all the possible routes to the desired destinations. Each element in the RT is a eleven-tuple of the form <*source, destination, next_hop, hop_count, available_bw, reserved_bw, active, route_class, first_class_link, second_class_link, third_class_link*>. The $source$ and $destination$ fields contain the unique addresses of the source and the destination node, respectively. The $next\_hop$ field contains the address of the neighbor node to which data packets need to be forwarded. The hop_cnt field contains the number of intermediate nodes from the source to the destination node on this route. The $available\_bw$ and $reserved\_bw$ fields are currently available bandwidth and reserved bandwidth along the path, respectively. The $active$ flag represents whether or not the route is currently being used. The $route\_class$ field contains the route class information of the route. The $route\_class$

field also contains the information of relative direction between the nodes consisting of the weakest link class of the route. The $first\_class\_link$, $second\_class\_link$ and $third\_class\_link$ fields contain the intermediate links which belong to the first link class, the second link class and the third link class respectively.

3.2 Route Discovery Protocol

3.2.1 Route Discovery

The route discovery mechanism is based on request-reply operations. A *Route_Request* packet is used for the request operation from the source node and carries *<source, destination, request_id, hop_cnt, QoS_metric, route_class, int_nodes, first_class_link, second_class_link, third_class_link>* information.

• **Processing a Route_Request Packet** : A source broadcasts a *Route_Request* packet to its neighbors to find multiple disjoint paths to a destination. When a node receives a *Route_Request* packet, the following procedures are invoked.

```
1 /* If destination node of the path */
2 IF (destination of the Route_Request packet = it's own address)
3       IF (int_node of the Route_Request is disjoint from the other paths found
            earlier) AND ( route_class of the Route_Request is not '+3')
4           - Append it's own address to the int_node of the Route_Request packet.
5           - Update the QoS_metric of the Route_Request packet.
6           - Update the route_class of the Route_Request packet.
7           - Insert the link class information between the last node and itself into
              one of three class_link fields.
8           - Copy the reverse of the int_nodes of the Route_Request packet into
              the int_nodes of a Route_Reply packet.
9           - Send the Route_Reply packet to the source node along int_nodes of
              the Route_Reply packet.
10          - Insert Route information from the destination to source into it's RT.
11          - Start an acknowledgment timer for the Route_Reply packet.
12      ELSE
13          - Discard the Route_Request packet.
14 /* if intermediate node of the path */
15 ELSE
16      IF (No more bandwidth is available)
17          - Discard the Route_Request packet.
18      ELSE IF (the pair <source, request_id> of the Route_Request packet has
            been received already) AND (int_node of the Route_Request packet is
            not disjoint from the other int_node fields received earlier)
19          - Discard the Route_Request packet.
20      ELSE IF (it's own address is in the int_nodes of the Route_Request packet)
21          - Discard the Route_Request packet.
22      ELSE
23          - Append it's own address to the int_node of the Route_Request packet.
24          - Update the QoS_metric of the Route_Request packet.
25          - Update the route_class of the Route_Request packet.
26          - Insert the link class information between the last node and itself into
              one of three class_link fields.
27          - Increase the hop_cnt by one of the Route_Request Packet.
28          - Broadcast the Route_Request Packet to it's neighbor nodes.
```

In response to a *Route_Request* packet, a *Route_Reply* packet is sent from the destination node. A *Route_Reply* packet is delivered to the source node along the reverse of the path which a *Route_Request* packet came through. An acknowledgment timer for a *Route_Reply* packet needs to be used at the node which sent the packet, so that the node retransmits the packet to the neighbor node unless it receives an acknowledgment from the neighbor within the timer value. It prevents the source node from failing to find paths when the source node does not receive the *Route_Reply* packet successfully.

• **Processing a Route_Reply packet** : When a node receives the *Route_Reply* packet, the following procedure is invoked.

```
1 - Send a Route_Ack packet to the previous neighbor node.
2 /* If source node of the path */
3 IF (destination of the Route_Reply packet = it's own address)
4       IF (This route is not in it's RT)
5           - Insert the route information from the source to destination into the RT.
6       ELSE IF (This route has updated QoS_metric information)
7           - Update the route information from the source to destination into the RT.
8       ELSE IF (This route has updated route_class information)
9           - Update the route information from the source to destination into the RT.
10 /* if intermediate node of the path */
11 ELSE
12      IF (This route is not in it's RT)
13          - Insert route information from the source to destination into the RT.
14          - Insert route information from the destination to source into the RT.
15      ELSE IF (This route has updated QoS_metric information)
16          - Update the route information from the source to destination into the RT.
17      ELSE IF (This route has updated route_class information)
18          - Update the route information from the source to destination into the RT.
19      - Forward the Route_Reply packet along int_nodes of the Route_Reply packet.
20      - Start an acknowledgment timer for the Route_Reply packet.
```

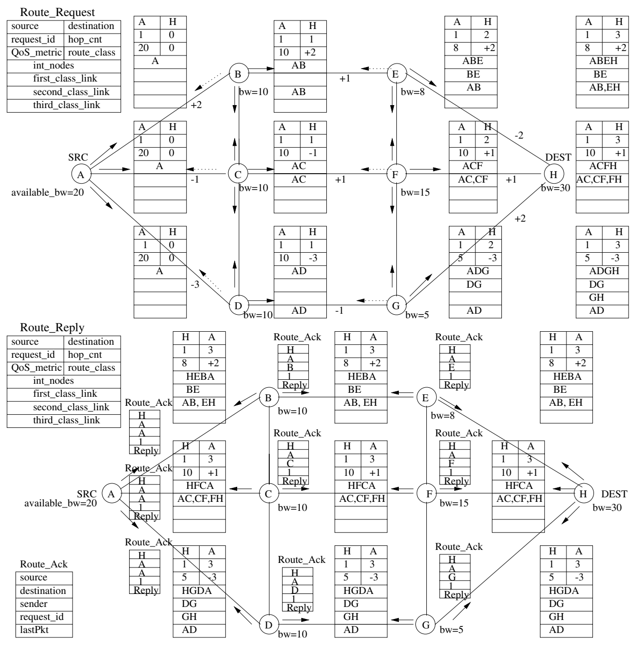

Figure 3. An example of route discovery with QoS consideration from node A to H

Figure **??** shows an example of the route discovery mechanism. First, source A broadcasts a *Route_Request* packet to the neighbors(i.e. B,C,D) to

Routing_Table at A

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	C	3	10	0	No	+1	AC,CF,FH		
A	H	B	3	8	0	No	+2	BE	AB,EH	
A	H	D	3	5	0	No	-3	DG	GH	AD

Routing_Table at H

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
H	A	F	3	10	0	No	+1	AC,CF,FH		
H	A	E	3	8	0	No	+2	BE	AB,EH	
H	A	G	3	5	0	No	-3	DG	GH	AD

Routing_Table at B

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	E	3	8	0	No				
H	A	A	3	8	0	No				

Routing_Table at E

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	H	3	8	0	No				
H	A	B	3	8	0	No				

Routing_Table at C

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	F	3	10	0	No				
H	A	A	3	10	0	No				

Routing_Table at F

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	H	3	10	0	No				
H	A	C	3	10	0	No				

Routing_Table at D

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	G	3	5	0	No				
H	A	A	3	5	0	No				

Routing_Table at G

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	H	3	5	0	No				
H	A	D	3	5	0	No				

Figure 4. An example of RT with QoS consideration at each node.

find paths to destination H with *source=A*, *destination=H*, *request_id=1*, *hop_cnt=0*, *QoS_metric=20(kbps)*, *route_class=0*, *int_nodes=A*, *first_class_link=second_class_link=third_class_link=null*. Let us assume that all the nodes in the network use the same maximum signal strengths, and that the signal strengths received at nodes C, B and D from the source A are greater than Th_1, between Th_2 and Th_1, and between S_R and Th_2 respectively. Therefore, nodes C, B and D belong to the first node class, second node class and third node class of the source A respectively, and links AC, AB and AD belong to the first link class, second link class and third link class respectively. When the node C receives the *Route_Request* packet, it appends its address to the *int_node* field (i.e AC), increases the *hop_cnt* by one, updates the *QoS_metric* with the available bandwidth at C (i.e. 10kbps), updates the *route_class* with the link class information which the intermediate link between the node A and C belongs to (i.e. the first link class with '-'), inserts the link between the node A and C into the *first_class_link* and broadcasts the packet to its neighbors(i.e. A,B,D,F). However, this packet is forwarded only to the neighbors B, D and F, because the node A drops the packet due to a loop. Solid and dotted arrow lines show the packet forwarding and dropping operations respectively. After receiving the *Route_Request* packet from the node A, nodes B and D broadcast the packet with the updated *QoS_metric* (i.e. 10 kbps), *route_class* (i.e. the second link class with '+' and third link class with '-' respectively), *second_class_link* and *third_class_link* to their neighbors. The other intermediate nodes operate in the same manner as nodes B, C and D. Finally, three *Route_Request* packets along different paths(i.e. $\{$ACFH$\}$,$\{$ABEH$\}$,$\{$ADGH$\}$) arrive at the destination H. When the destination H receives a *Route_Request* packet from the node F, it appends its address to the *int_node* field, checks whether or not the *int_node* field is disjoint with the paths found so far. If it is disjoint, it calculates the minimum available bandwidth along the *int_nodes* (i.e. $\{$ACFH$\}$), updates the *QoS_metric* and *route_class* if needed, inserts the link class information and sends a *Route_Reply* packet with updated information back to the node F. The *Route_Reply* packet arrives at the sender A along the *int_nodes* path in the *Route_Reply* packet (i.e. $\{$HFCA$\}$). Each intermediate node A, C and F confirms by acknowledging with a *Route_Ack* packet to node C, F and H respectively. The retransmission of the *Route_Reply* packet needs to be invoked in case of a packet loss. The destination H operates in the same manner for the other *Route_Request* packets. Finally, the source A receives three *Route_Reply* packets from the destination H, thus finding three disjoint paths. Figure **??** shows the RT at the nodes right after three disjoint paths have been discovered from A to H.

3.2.2 Route Selection and Resource Reservation

When the source has multiple routes, a route with stronger *route_class* takes precedence over other routes. When the source has multiple routes of same route class, '-' takes precedence over '+'. This is because the route with stronger route class, where intermediate nodes are getting closer, is likely to result in a longer-lived connection and path. However, it excludes the selection of any paths with $route\_class$ as '+3', because the paths will be unavailable soon. From the example of Figure **??**, the source A would select the first path for 10kbps and second path for 5kbps with bandwidth constraint of 15kbps.

Bandwidth reservation is needed for data flow on the selected paths. A *QoS_Reserve* packet is used for bandwidth reservation from the source to destination along the selected path. When an intermediate node receives the packet, ADQR interacts with the resource management protocol to perform bandwidth reservation. In response to the *QoS_Reserve* packet, the intermediate node acknowledges with a *QoS_Ack* packet to the previous neighbor node and forwards the *QoS_Reserve* packet to the next neighbor node. A retransmission timer for the *QoS_Reserve* packet needs to be applied, so that each node retransmits the packet to the next neighbor node after timeout value expires unless it receives an acknowledgment from the next neighbor node. Right after getting an acknowledgment, each node sends the data to the next neighbor node.

3.3 Route Maintenance Protocol

Once the paths have been found, they need to be monitored and updated continuously, because frequent node movement and channel impairments result in path breakage. We propose a fast route maintenance scheme which is called *two-phase monitored rerouting*. Two distinct route maintenance phases, namely *Pre-Rerouting* and *Rerouting* phases are defined. In the *Pre-Rerouting* phase, it performs the operations to prepare for rerouting. In the *Rerouting* phase, it performs the operations of rerouting itself. Signal strength is used to determine the phases.

In order to determine the *Pre-Rerouting* phase and perform the maintenance operations during the phase, the threshold Th_1 is used in conjunction with *Route_Prepare* packet. In order to determine the *Rerouting* phase and perform the maintenance operations during the phase, the threshold Th_2 is used in conjunction with *Route_Reroute* packet. *Route_Update* packet is used in conjunction with both thresholds during both phases. Each packet contains <*source, destination, sender, request_id, link, link_class*> information.

Another set of four packets are used for QoS management; *QoS_Reserve*, *QoS_Release*, *QoS_Update* and *QoS_Ack* packets. Their operation is combined with route maintenance and will be discussed in this section.

3.3.1 The First-Link-Class

When the received signal strength from a neighbor becomes stronger and reaches above Th_1 (i.e. the class of the intermediate link has been changed from the second link class to the first link class), and the link to the neighbor is on the path list in it's RT, the node sends a *Route_Update* packet to the source node of the path. When the source node receives the *Route_Update* packet, it updates the link class information of the path in RT. The source node moves the $link$ field of the *Route_Update* packet from the $second\_class\_link$ into the $first\_class\_link$ of the path in it's RT. The source node also updates the $route\_class$ of the path in it's RT, if the $link\_class$ of the *Route_Update* packet is the weakest intermediate link class of the path. Updating the $route\_class$ field means that the route class of the path has been changed to the first route class, and also means that the intermediate nodes are getting closer.

3.3.2 The Second-Link-Class - Pre-Rerouting Phase

• In case of active route

When the received signal strength from a neighbor becomes weaker and falls below Th_1, but is larger than Th_2 (i.e. the class of the intermediate link has been changed from the first link class to the second link class), and the link to the neighbor is on the active path list in RT, the node sends a *Route_Prepare* packet to the source node of the path. The purpose of the *Route_Prepare* packet is to have the source node find alternate paths in advance before the current path becomes unavailable. When the source receives the *Route_Prepare* packet, it scans the RT to find out whether or not there are other non-active paths that meet the QoS constraints to the destination. It excludes any paths with $route\_class$ as '+3'. If there are, the source node does not do anything further, except updating the route information of the path if needed. The source node moves the $link$ of the *Route_Prepare* packet from the $first\_class\_link$ into the $second\_class\_link$ of the path in RT. The source node also updates the $route\_class$ of the path, if the $link\_class$ of the *Route_Prepare* packet is the weakest intermediate link class of the path. Updating the $route\_class$ field means that the route class of the path has been changed from the first route class to the second route class, and also means that the intermediate nodes are getting away from each other. If there are no other existing paths to the destination in the RT, the source node initiates a route discovery mechanism to find alternate paths to the destination. The source continues to transfer data over the current path, because the path is not broken yet. When an intermediate node receives the *Route_Prepare* packet, it forwards the packet to the source node. A timer for the *Route_Prepare* packet needs to be used at the sender node, so that it retransmits the packet to the source node after timeout value expires. It prevents the source node from failing to find alternate paths when the source node does not receive the *Route_Prepare* packet successfully. The sender node continuously retransmits the *Route_Prepare* packet to the source node every time the timeout value expires, until the signal strength from a neighbor falls below Th_2. When the source node receives the redundant packets, it just ignores them if it already has alternate non-active paths to the destination.

However, when the received signal strength from a neighbor becomes stronger and reaches above Th_2, but is below Th_1 (i.e. the intermediate link class has been changed from the third link class to the second link class), and the link to the neighbor is on the active path list in RT, the node sends a *Route_Update* packet to the source node of the path. The purpose of the *Route_Update* packet is to have the source node update the route information correctly. When the source node receives the *Route_Update* packet, it moves the $link$ field of the *Route_Update* packet from the $third\_class\_link$ into the $second\_class\_link$ of the path in RT. The source node also updates the $route\_class$ of the path, if the $link\_class$ of the *Route_Update* packet is the weakest intermediate link class of the path. Updating the $route\_class$ field means that the route class of the path has been changed from the third route class to the second class, and also means that the intermediate nodes are getting closer.

• In case of non-active route

When the received signal strength from a neighbor reaches between Th_2 and Th_1 (i.e. the class of the intermediate link belongs to the second link class), and the link to the neighbor is on the non-active path list in RT, the node sends a *Route_Update* packet to the source node of the path. When the source receives the *Route_Update* packet, it updates the link class information of the path in it's RT. The source also updates the $route\_class$ field of the path, if the $link\_class$ of the *Route_Update* packet is the weakest intermediate link class of the path. Updating the $route\_class$ field means that the route class of the path has been changed.

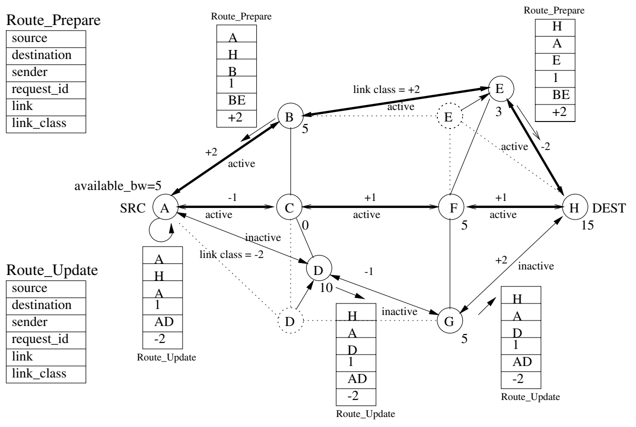

Figure 5. An example of *Route_Prepare* and *Route_Update* packets operation

In the example shown in Figure **??**, the source A uses two paths (i.e. {ACFH},{ABEH}) to the destination H. When the node B detects that the signal strength from the node E reaches a value less than Th_1, but is larger than Th_2, as the node E moves, it sends a *Route_Prepare* packet to the source A with *source=A*, *destination=H*, *sender=B*, *request_id=1*, *link=BE* and *link_class=+2*. The source A does not take any action, because there is an available non-active path (i.e. {ADGH}) that meets the bandwidth constraint (i.e. 5kbps). If there are no other paths, the source A would initiate the route discovery mechanism to find alternate paths. The source A just updates the second route information in RT as shown in Figure **??**. The node E also sends a *Route_Prepare* packet to the node H. When the source A detects that the signal strength from the node D reaches a value larger than Th_2, but is less than Th_1, as the node D moves, it sends a *Route_Update* packet to the source A with *source=A*, *destination=H*, *sender=A*, *request_id=1*, *link=AD* and *link_class=-2*. The source A just updates the third route information in RT as shown in Figure **??**. The node D also sends a *Route_Update* packet to the node H.

3.3.3 The Third-Link-Class - Rerouting Phase

• In case of active route
When the received signal strength from a neighbor becomes

Routing_Table at A

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	C	3	0	10	Yes	+1	AC,CF,FH		
A	H	B	3	3	5	Yes	+2		AB,EH,BE	
A	H	D	3	5	0	No	+2	DG	AD, GH	

Routing_Table at H

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
H	A	F	3	0	10	Yes	+1	AC,CF,FH		
H	A	E	3	3	5	Yes	+2		AB,EH,BE	
H	A	G	3	5	0	No	+2	DG	AD,GH	

Figure 6. An Example of RT at node A and H, after *Route_Prepare* packet operation

weaker and falls below Th_2, but is still larger than S_R (i.e. the class of the intermediate link has been changed from the second link class to the third link class), and the link to the neighbor is part of the active path in RT, the node sends a *Route_Reroute* packet to the source node of this path. The purpose of the *Route_Reroute* packet is to have the source node change the path to another one in advance of the path becoming unavailable. When the source node receives the *Route_Reroute* packet, it checks whether or not there are other inactive paths that meet the bandwidth constraint to the destination, that no other *Route_Reroute* packets have been received, and that the available paths do not belong to the third route class with '+'. If there are such paths, it changes the path to the path that is likely to be most longer lived from the set of available paths, makes the old path inactive and updates other route information. If not, it continues to transfer data on the old path while it initiates a route discovery mechanism to find alternate paths. Traffic is not diverted on those paths which have received a *Route_Reroute* packet already or which belong to the third route class, because they are likely to become unavailable soon. After the source finds alternate paths, it performs rerouting. When an intermediate node receives the *Route_Reroute* packet, it forwards the packet to the source node. Once again, a retransmission timer for the *Route_Reroute* packet needs to be applied, so that the sender node retransmits the packet to the source node after the timeout value expires. It prevents the source node from failing to reroute traffic when the source node does not receive the *Route_Reroute* packet successfully. The sender node continuously retransmits the *Route_Reroute* packet to the source node every time the timeout value expires, until rerouting is performed. When the source node receives the redundant packets, it just ignores them if it has already rerouted traffic to the destination.

The rerouting involves bandwidth reservation and release on the newly selected paths and old paths respectively. The source node sends a *QoS_Reserve* packet along the newly selected paths to reserve the required bandwidth before transmitting data. The source node also sends a *QoS_Release* packet along the old paths to release the bandwidths which have been reserved previously. In response to these packets, the intermediate node acknowledges with a *QoS_Ack* packet to the previous neighbor node. An acknowledgment timer of the *QoS_Reserve* and *QoS_Reserve* packets needs to be applied at each node which sends the packets, so that the node retransmits the packets unless it receives an acknowledgment within the timer value.

• In case of non-active route

When the received signal strength from a neighbor reaches between S_R and Th_2 (i.e. the class of the intermediate link belongs to the third link class), and the link to the neighbor is on the non-active path list in RT, the node sends a *Route_Update* packet to the source node of the path. When the source receives the *Route_Update* packet, it updates the link class information of the path in RT. The source node also updates the $route\_class$ field of the path, if the $link\_class$ of the *Route_Update* is the weakest intermediate link class of the path. Updating the $route\_class$ field means that the route class of the path has been changed.

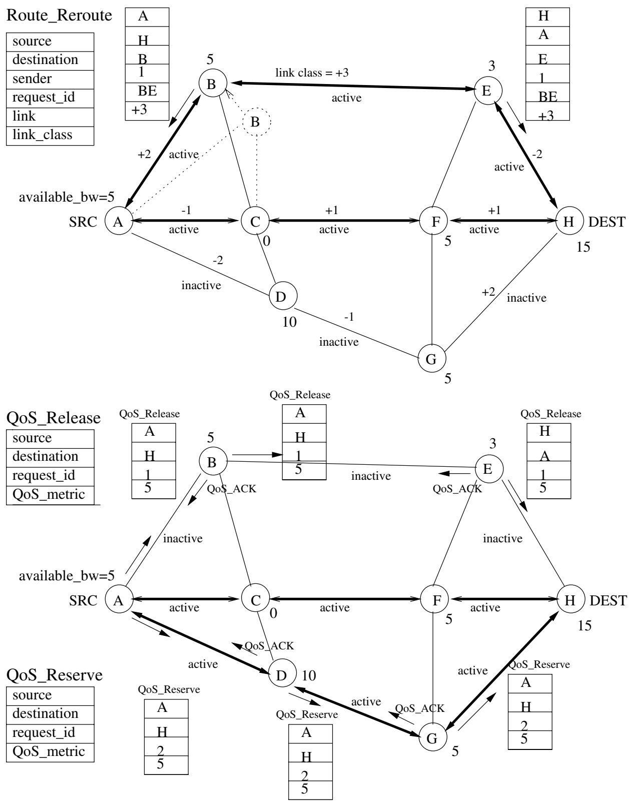

Figure 7. An example of *Route_Reroute* **packet operation**

In the example shown in Figure **??**, when the node B determines that the signal strength from the node E has reached a value between S_R and Th_2 due to its motion, it sends a *Route_Reroute* packet to the source A with *source=A*, *destination=H*, *sender=B*, *request_id=1*, *link=BE* and *link_class=+3*. The node E also sends a *Route_Reroute* packet to the node H. After receiving the packet, the source A changes the path from {ABEH} to {ADGH} (i.e. it uses {ADGH},{ACFH}) and sends a *QoS_Reserve* packet along the new path (i.e {ADGH}) to reserve the required bandwidth (i.e. 5kbps). source A also sends a *QoS_Release* packet along the old path (i.e {ABEH}) to release the bandwidth that was reserved (i.e. 5kbps) and makes it inactive. Each intermediate node which received the *QoS_Reserve* and *QoS_Release* packets confirms by acknowledging with a *QoS_Ack* packet. The retransmission of the *QoS_Reserve* or *QoS_Release* packets needs to be invoked in case of not receiving the acknowledgment within the timer value. The RT at node A and H needs to be updated as shown in Figure **??**.

Routing_Table at A

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
A	H	C	3	0	10	Yes	+1	AC,CF,FH		
A	H	B	3	8	0	No	+3		AB,EH	BE
A	H	D	3	0	5	Yes	+2	DG	AD, GH	

Routing_Table at H

Source	Dest	Next Hop	Hop Count	Available Bandwidth (kbps)	Reseved Bandwidth (kbps)	Active	route class	first class link	second class link	third class link
H	A	F	3	0	10	Yes	+1	AC,CF,FH		
H	A	E	3	8	0	No	+3		AB,EH	BE
H	A	G	3	0	5	Yes	+2	DG	AD,GH	

Figure 8. An example of RT at nodes A and H, after *Route_Reroute* **packet operation**

3.3.4 Reporting Broken Paths

When a neighbor is not reachable any more, and the link to the neighbor is part of paths in RT, the node sends a *Route_Error* packet to the source node of these paths. Each intermediate node including the source node which is part of the paths deletes the route information from their RT and releases any bandwidth reserved on the path appropriately. In response to the *Route_Error* packet, the intermediate node acknowledges with a *Route_Ack* packet to the previous neighbor node and forwards the *Route_Error* packet to the next neighbor node. An acknowledgment timer for the *Route_Error* packet needs to be applied, so that each node retransmits the packet to the next neighbor node after timeout value expires unless it receives an acknowledgment from the next neighbor node.

3.3.5 QoS Information Update

A node can be a part of non-active paths and also a part of other active paths at the same time. When the node reserves network resources for other active paths, it sends a *QoS_Update* packet to the source node of each non-active path. Each intermediate node including the source node which is part of the non-active paths updates the network resource information in its RT.

4 Simulation Results

The ADQR protocol was simulated using GLOMOSIM. A network consisting of 30 mobile nodes over a fixed size $500m \times 500m$ terrain was considered. The maximum transmission power range was assumed to be $100m$ between nodes. Nodes were initially placed randomly within the fixed-size physical terrain. They move to a randomly selected destination within the terrain with a speed of MOVING-SPEED(meter/sec). After reaching the destination, they stay there for 3sec time period before they move again. The source and destination are selected randomly, and they also move and stop continuously in random directions during the whole simulation.

A two-state Markov model [**?**] is used to represent the error behavior of slowly fading channels in wireless networks. The Markov chain model consists of two states, Good(G) and Bad(B). In the good state G, the probability of error (P_{eg}) is 0. In the bad state B, the probability of error (P_{eb}) is 0.5. P represents the transition probability from state G to B, whereas p is the transition probability from state B to G. The probabilities to remain in each state are $Q = 1 - P,\ q = 1 - p$ respectively. It can be shown that $P = (p * (ber - P_{eg}))\ /\ (P_{eb} - ber)$, where ber is the channel bit error rate. In our simulation, p is assumed to be 0.25.

ADQR is expected to receive a beaconing control packet every 200ms, to update the signal strength received from each neighbor node. ADQR considers a link to a neighbor to be broken, if it does not receive a beacon packet from the neighbor more than 5 consecutive times. An acknowledgment (*Route_ACK* or *QoS_ACK* packet) is required in response to *Route_Reply*, *Route_Error*, *QoS_Reserve*, *QoS_Release* and *QoS_Update* packets. An acknowledgment timer value for the acknowledgment is set equal to 10ms, so that ADQR retransmits these packets in case an acknowledgment is not received within 10ms. A retransmission timer is involved for *Route_Prepare* and *Route_Reroute* packets, and a value is set equal to a node staying stationary for a time period, i.e. 3sec in this simulation.

The second threshold Th_2 is the actual point at which rerouting is actually done. When a node is moving with velocity $X(meter)/S(sec)$, and the time for a packet to traverse from end to end in the network is T, β should be greater than or equal to $(4 \times T \times X)/S$ in the worst case in order to reroute packets without data flow disruptions, i.e. $\beta \geq$ (Time to notify the source node about the need for rerouting + Time for the source to complete the route discovery task + Time for the last packet to traverse from the source to destination, before rerouting occurs) $\times\ (X/S) = (T + 2T + T) \times (X/S)$. β is set equal to $(4 \times T \times X)/S$. T is set equal to 200ms. $X(meter)/S(sec)$ is the node moving speed which ranged from 0m/s to 10m/s in this simulation. Th_1 is the point at which rerouting preparation begins. α is set at $2 \times \beta$.

• QoS goodput

The simulation is to analyze how routing protocols support end-to-end QoS in ad-hoc networks. To analyze the end-to-end QoS support, a QoS-aware application needs to be simulated, where it requires the certain amount of bandwidth as a QoS requirement in this work. To simulate the application, a Constant Bit Rate (CBR) application which generates real time data traffic from source to destination at a constant rate is used. CBR runs over UDP transport layer protocol which does not employ retransmission schemes for lost data packets. Packets are dropped at the intermediate nodes, whenever they get corrupted due to channel fading. The link bandwidth is 2Mbps in the simulations. Network congestion is simulated at some of the nodes, which results in low available bandwidths at the nodes. Heavy incoming data traffic into the nodes which are congested results in dropped packets at the nodes.

QoS goodput is defined as the ratio of actual end-to-end QoS achieved to the required end-to-end QoS, where end-to-end QoS is represented by bandwidth (in terms of bits per second) for data transfer. Figures **??** and **??** present the QoS goodput of the ADQR protocol. These results represent how node mobility and link bit error rates affect the end-to-end QoS in ad-hoc networks. ADQR does not support 100% of required end-to-end QoS as nodes move around. ADQR tries to reroute before a routing path is broken. However, all the feasible paths may be broken in the middle of data transmission due to the mobility of the intermediate nodes. This results in QoS disruptions as shown in the results. It is assumed that a QoS-aware application has the functionality to negotiate the end-to-end QoS requirement in case of QoS disruptions. However, this application is not implemented in this simulation in order to focus how much a QoS routing procotol itself (ADQR) can support the required end-to-end QoS. ADQR supports almost 100% of QoS requirements when node mobility is extremely low, and also supports more than 90% of QoS requirements when moderate data packet size is used under low link error rate conditions.

5 Summary and Conclusions

In this paper, a new QoS routing algorithm, ADQR, is proposed for mobile ad-hoc networks. ADQR uses signal strength to make routing decisions. In the route discovery phase, ADQR finds multiple disjoint paths with network resource information using retransmission scheme. A feasible path that is more likely longer-lived is selected for data transfer. In the route maintenance phase, ADQR effectively keeps monitoring network topology changes and performs

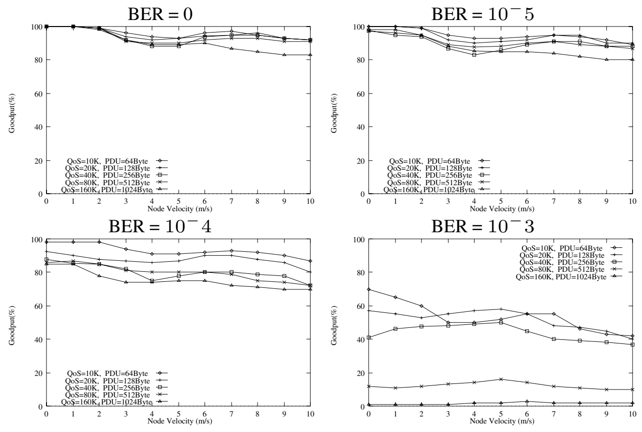

Figure 9. ADQR QoS Goodput for different bit error rates

rerouting before the paths become unavailable. With these route discovery and maintenance mechanisms operating together, ADQR significantly improves routing performance and supports end-to-end QoS in ad-hoc networks.

References

[1] Y Hwang, H Lee and P Varshney An Adaptive Routing Protocol for Ad-hoc Networks using Multiple Disjoint Path *VTC*, May, 2001

[2] S. Basagni, I. Chlamtac, V.R. Syrotiuk and B.A. Woodward A Distance Routing Effect Algorithm for Mobility (DREAM) *MOBICOM*, October, 1998

[3] J. Raju and J.J. Garcia-Luna-Aceves A New Approach to On-demand Loop-Free Multipath Routing *Computer Communications Review*, October, 1999

[4] S. Vutukury and J.J. Garcia-Luna-Aceves MDVA: A Distance-vector Multipath Routing Protocol *INFOCOM*, May, 2001

[5] J. Wu An Extended Dynamic Source Routing Scheme in Ad hoc Wireless Networks *HICSS*, January, 2002

[6] H. Nasipuri and S. Das On-Demand Multipath Routing for Mobile Ad Hoc Networks *ICCCN*, October, 1999

[7] X. Lin and I. Stojmenovic Location Based Localized Alternate, Disjoint, Multi-path and Component Routing Algorithms for Wireless Networks *Mobile AdHoc Networking and Computing*, October, 2001

[8] S. Chen and K. Nahrstedt Distributed Quality-of-Service Routing in Ad Hoc Networks *JSAC*, August, 1999

[9] C. Lin and M. Gerla Real-time Support in Multihop Wireless Networks *Wireless Networks*, March, 1999

[10] D. Sidhu, R. Nair and S. Abdallah Finding Disjoint Paths in Networks *SIGCOMM*, September, 1991

[11] E. Gilbert The Capacity of a Burst-Noise Channel *The Bell System Technical Journal*, September, 1960

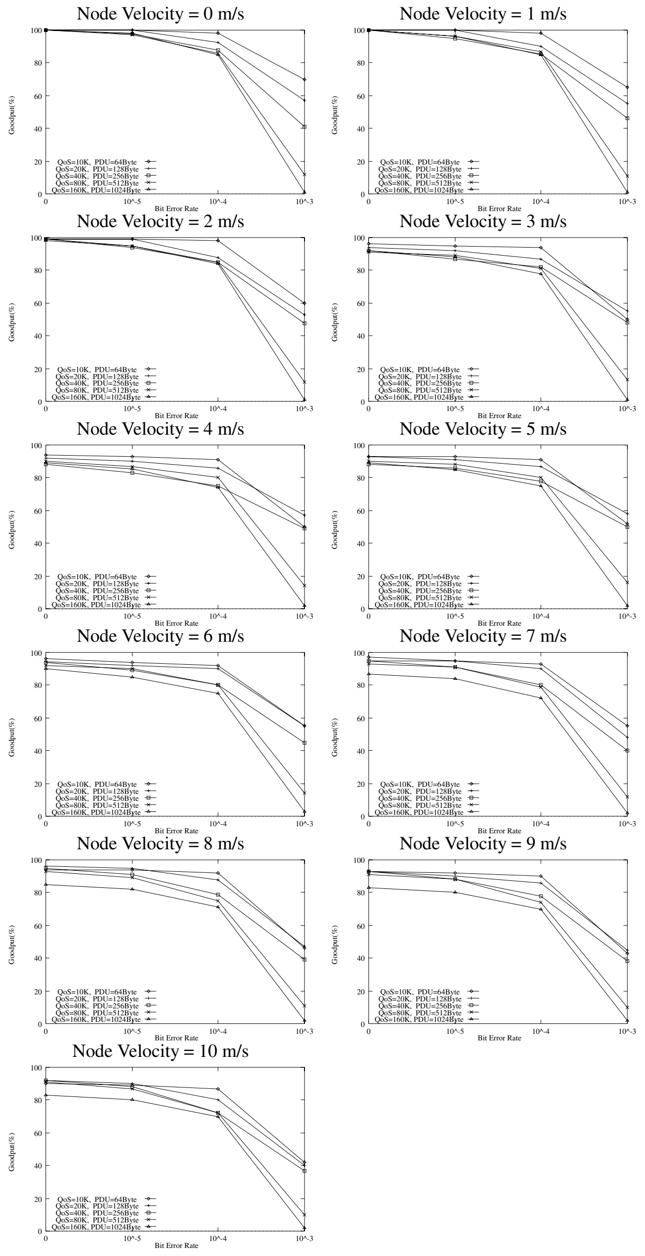

Figure 10. ADQR QoS Goodput for different node velocities