Predictive Location-Based QoS Routing in Mobile Ad Hoc Networks

Samarth H. Shah, Klara Nahrstedt [*]

{shshah,klara}@cs.uiuc.edu

Department of Computer Science

University of Illinois at Urbana-Champaign

Urbana, IL 61801, U.S.A.

Abstract: *In mobile ad hoc networks, nodes are dynamically changing their locations. The QoS information used for routing by traditional QoS routing protocols becomes obsolete due to node mobility. To overcome this problem, a predictive QoS routing scheme is needed. In this paper, we present a location-delay prediction scheme, based on a location-resource update protocol, which assists a QoS routing protocol. Our location updates also contain resource information pertaining to the node sending the update. This resource information for all nodes in the network and the location prediction mechanism are together used in the QoS routing decisions. Our simulation results show that, with our approach, we can predict the location at a given instant in the future with a high degree of accuracy.*

1 Introduction

An ad hoc network is a collection of randomly moving wireless devices within a particular area. Unlike in cellular networks, there are no fixed base-stations to support routing and mobility management. In this paper, we add a further

[*]This research was supported by the ONR MURI grant under grant number 1-5-21394, and the NSF EIA 99-72884EQ grant under grant number 1-5-31744.

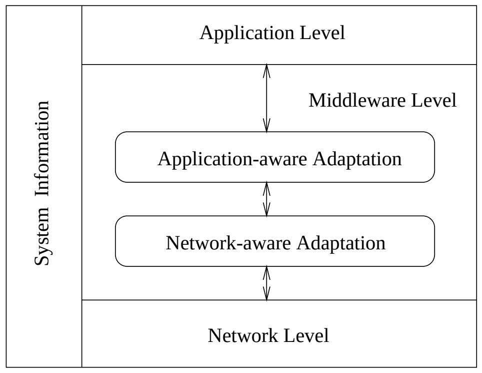


Figure 1. System Framework

restriction to the traditional definition of an ad hoc network above in that the nodes in our underlying network also vary dynamically in their resource-richness. There are dynamic changes in topology and resource availability due to the high degree of mobility of nodes in the ad hoc network. Due to these changes, the topological and routing information used by current network protocols is rendered obsolete very quickly.

Our overall goal is to build a system that can support multimedia applications such as video phones and video on demand over such a dynamic ad hoc network. A major acceptance criterion for these applications is their high quality of service (QoS). Our overall solution to this problem is a co-operative network-middleware framework. The network level obtains the state information pertaining to all the nodes in the ad hoc network through a location-resource update protocol. This information is shared by both the network level and the middleware level which lies above it. The application level lies above the middleware level. The state information obtained through network-level mechanisms assists the middleware in its goal of maximizing application QoS. The middleware also assists the network in flow-scheduling and rate-control to obtain maximum throughput. The overall system framework is shown in Figure 1.

In this paper, we present in detail our research at the network level, i.e. the state information distribution and location-and-delay prediction schemes for QoS routing. It is essential to use a predictive scheme because state information in a dynamic environment, such as an ad hoc network, does not remain current for very long. Our results show that we can accurately predict the location at a future instant and this prediction, coupled with the low-overhead resource information exchange, can be useful in performing QoS routing.

The paper is organized as follows. In section 2, we describe some of the related research vis-a-vis routing in ad hoc networks. Section 3 discusses our research at the network-level. We deal in detail with the distribution of location and resource information. Furthermore, we also present a scheme for location-and-delay prediction and describe how it is used in the QoS routing of data packets. In section 4, we evaluate the performance of our prediction scheme through extensive simulations and show the feasibility of our approach. Finally, section 5 concludes this paper.

2 Related Work

A lot of related research has been done in the areas of routing in ad hoc networks using both hop-count-based as well as location-based metrics. Some of this related work is discussed in this section.

Traditional ad hoc routing protocols can be broadly classified into two categories - *proactive* and *reactive* protocols. DSDV [5] is an example of a proactive protocol. In this protocol, the nodes in the network periodically broadcast their routing tables to other nodes in a proactive fashion. Based on these routing table updates from other nodes, a particular node in the network can calculate the next hop to each destination in the network. Then, at the time of packet forwarding, a node has to only look up the next hop entry in the routing table for the destination indicated.

DSR [3] and AODV [6] are examples of reactive or *on-demand* protocols. In these protocols, the source, at the time of sending a packet, broadcasts a *query* to obtain a route to the destination. The broadcast query propagates through the network and reaches the destination which sends a unicast reply back to the source containing the source route to the destination. Data packets are then sent along this source route.

For nodes moving with a relatively high velocity and frequently changing direction, the routing table updates (assuming a proactive protocol like DSDV) become obsolete by the time they reach the correspondent node. In a reactive ad hoc routing protocol like DSR, if the source route is broken due to one of the intermediate nodes having moved to a new location, a new source route has to be computed. Or, alternatively, a route repair mechanism has to be used. However, there is a delay between the detection of route breakage and the establishment of a new route, which could be harmful to real time transmissions.

The motivation behind using a location-based scheme rather than a hop-based scheme to solve the problem of routing is as follows: The updates of geographic location that we use in our mechanism can also become obsolete as the node moves from one location to another, just as routing table updates can. However, we can still *predict* a future location based on previous location updates. Previous routing table updates from a destination as used in DSDV, however, cannot be used to predict the routing table at some future instant. This is because the next-hop entries in the routing table become incorrect when the node migrates to a new neighborhood.

Location-based routing schemes have been previously used in ad hoc routing in protocols such as LAR [4] and DREAM [1]. Both LAR and DREAM use a very weak prediction mechanism. These protocols do not take the direction of motion of the destination into account when attempting to predict the location at a future instant. In DREAM, update intervals depend only on the velocity of motion of the node. In our scheme, updates are generated based both on velocity and direction of motion of the node. Our protocol also differs from DREAM in terms of the mechanism used to prevent full flooding of the network by the location updates.

In [7], the locations and mobility patterns of neighboring nodes are used to compute the Link Expiration Time (LET) of the link between them. The LET can then be applied to both proactive as well as reactive protocols as the metric for each link. However, using this method, routes that are worse in terms of this new metric but better in terms of delay characteristics are not considered at all, even until their expiration time. Furthermore, the update mechanism proposed in this paper is very primitive. It does not adapt well to changes in velocity and direction because no special updates are generated to indicate this change in pattern of motion.

Also, none of the location-based routing schemes mentioned above perform QoS routing based on the resource availability at the intermediate nodes in the source to destination route. This is an important feature of our scheme.

3 Network Level Mechanisms

The network level mechanisms for mobile nodes need to assist in (1) location prediction and (2) QoS routing. To satisfy these requirements, we investigate (a) *the location-resource update protocol* which allows for location/resource information distribution, (b) *the location prediction scheme* to estimate new location at a future instant, based on information regarding previous locations and previous end-to-end delays and (c) *QoS routing protocol* to route multimedia data in an efficient and high quality fashion.

3.1 Update Protocol

The update protocol is crucial for distribution of geographical location and resource information. We consider resources such as battery power, queuing space, processor speed, transmission range, etc. We assume in this paper that all clocks in the mobile ad hoc nodes are synchronized, and hence all update information is properly ordered. We also assume that each node can obtain its geographic location using GPS or some similar mechanism. In our protocol, we

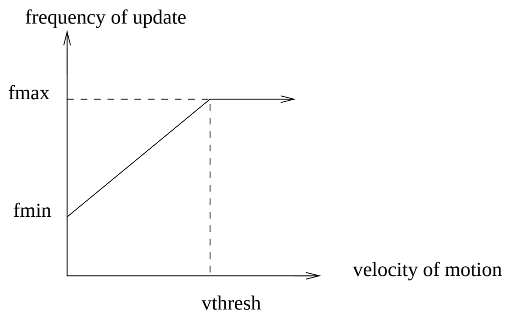


Figure 2. Variation of update frequency of type 1 update with velocity of the node

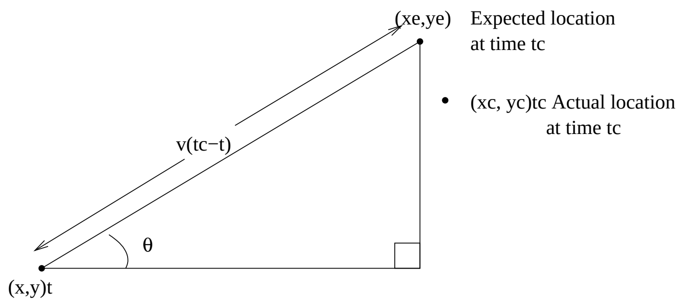

(xe,ye) Expected location at time tc
(xc, yc)tc Actual location at time tc
v(tc−t)
θ
(x,y)t

Figure 3. Check at time t_c whether type 2 update should be generated

use two types of updates:

1. *Type 1 update:* A type 1 update is generated periodically. It can be generated with a constant frequency, i.e. the *time* between successive type 1 updates remains constant. Alternatively, the frequency of the type 1 update can vary linearly between a maximum (f_{max}) and a minimum (f_{min}) threshold, with the velocity v of the node. Consequently, the *distance* travelled between successive type 1 updates remains constant. This function is illustrated in Figure 2.

2. *Type 2 update:* A type 2 update is generated when there is a considerable change in the node's velocity or direction of motion. From its recent history (i.e. from recent updates), the mobile node can calculate an expected location that it should be in at a particular instant. The node then periodically checks if it has deviated a distance greater than δ from this expected location. If it has deviated more than a distance δ from its expected location, a type 2 update is generated.

   Suppose that the periodic check for a particular node is scheduled at time t_c. Then, the node finds whether it has deviated more than a distance δ from its expected location (x_e, y_e) at t_c. Further, let us suppose that its own most recent update was generated at time t, $t = t_c - \triangle t$, where $\triangle t$ is some time interval. Let us assume this update was generated at point (x, y), reporting a velocity v, and direction such that the node moves at an anti-clockwise angle θ to the horizontal. Also, let us assume the current location of the node at the time of checking t_c is (x_c, y_c). Let the velocity at the time of checking v_c remain unchanged since the last update, i.e. $v_c = v$. This situation is illustrated in Figure 3. Then, expected location (x_e, y_e) is given by the equations:

$$x_e = x + v \cdot (t_e - t) \cdot cos\theta \tag{1}$$

$$y_e = y + v \cdot (t_e - t) \cdot sin\theta \tag{2}$$

Now, if $[(x_e - x_c)^2 + (y_e - y_c)^2]^{1/2} > \delta$, then a type 2 update due to significant change in the pattern of motion is generated at the time of checking, i.e. t_c. Care is taken to see that δ is large enough to prevent the reporting of minor perturbations in direction.

Alternatively, if there is a significant change in the velocity at t_c also a type 2 update due to significant change in the pattern of motion is generated. In our simulations, we define significant change in velocity as an increase or decrease of $\geq$ 1 m/s. Thus, a type 2 update could also be generated if $|v_c - v| \geq$ 1 m/s. If there is no significant change in either velocity or direction, then no type 2 update is generated.

The frequency of the periodic check for significant change in the pattern of motion is obviously greater than that of the minimum threshold f_{min} of the type 1 update. Thus, any update that is not a periodic update, but is generated by the significant change of some resource/property of the node, is categorized as a type 2 update.

Update packets contain timestamps, current geometric co-ordinates, direction of motion, velocity and also resource information pertaining to the node that is used in QoS routing. Updates also contain a *motion stability parameter*, which is also used in QoS routing. The motion stability parameter is a special QoS parameter, represented by a single bit, that indicates whether the update has been generated by a routine periodic timeout or due to the node's changed pattern of motion (i.e. when $[(x_e - x_c)^2 + (y_e - y_c)^2]^{1/2} > \delta$ or $|v_c - v| \geq 1$ m/s). In other words, it indicates whether the update is a type 1 or a type 2 update.

Conceptually, the motion stability parameter indicates whether the velocity and direction of motion of the node are constant, and hence predictable, or are dynamically varying. If the velocity and direction are dynamically varying, they are hard to predict accurately. Hence, such nodes with dynamically varying patterns of motion should not be used as intermediate nodes for connections requiring low delay and low delay jitter. Using such nodes with unstable motion patterns as intermediate nodes results in frequent re-routing, which increases delay and delay jitter.

The update packet also has a field for the mobile node to indicate whether it is moving in a piecewise-linear pattern (i.e. with angular velocity zero) or angular pattern (i.e. with angular velocity greater than zero).

Updates from a node in the network are propagated to other nodes by broadcast flooding, similar to DSDV [5] and DREAM [1]. In section 3.5, we discuss a method to optimize the flooding of updates.

3.2 Predictions

When a packet arrives at a node a to be routed to a particular destination b, a has to follow a two step process to forward the packet along.

The first step is to predict the geographic location of the destination b as well as the candidate next hop nodes, *at the instant when this packet will reach the respective nodes*. Hence, this step involves a location as well as propagation delay prediction. The location prediction is used to determine the geographical location of some node (either an intermediate node or the destination b) at a particular instant of time t_p in the future when the packet reaches it. The propagation delay is used to estimate the value of t_p used in the above location prediction. These predictions are performed based on previous updates of the respective nodes.

The second step is to perform QoS routing based on the information, determined in the first step, as well as the resource-availability information for the candidate next-hop nodes.

Location Prediction: For now, we assume that a node moves in a piecewise linear pattern. In other words, we assume that between successive update points, the node has moved in a straight line. For a piecewise linear motion pattern and update packets that do not contain direction information, two previous updates are sufficient to predict a future location of the mobile node in the plane.

Let (x_1, y_1) at t_1 and (x_2, y_2) at t_2 $(t_2 > t_1)$ be the latest two updates respectively from a destination node b to a particular correspondent node a. Let the second update also indicate v to be the velocity of b at (x_2, y_2). Assume that

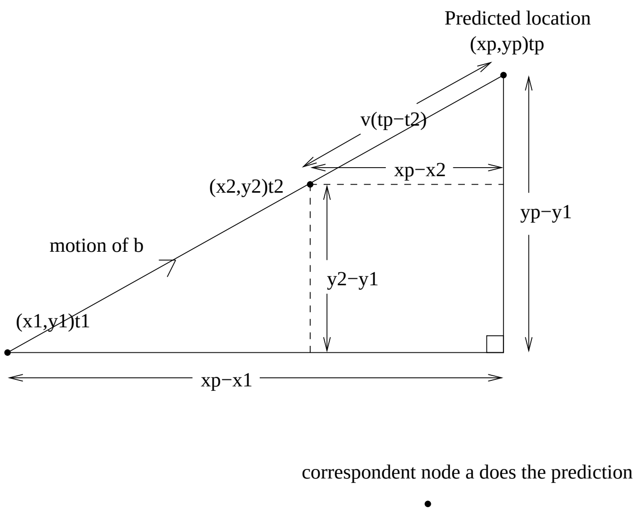


Figure 4. Prediction of future location at a given time in the future using last 2 updates

a wishes to predict the location (x_p, y_p) of b at some instant t_p in the future. (We have to first determine the time t_p for which this location is being predicted and this is where delay prediction comes into play.) This situation is depicted in Figure 4. The value of t_p is set by a to current time + predicted delay for the packet to reach b from a.

From Figure 4, using similarity of triangles, we get:

$$\frac{y_2 - y_1}{y_p - y_1} = \frac{x_2 - x_1}{x_p - x_1} \tag{3}$$

Solving for y_p from the above equation,

$$y_p = y_1 + \frac{(x_p - x_1)(y_2 - y_1)}{x_2 - x_1} \tag{4}$$

Using the above equation, a can calculate y_p if it knows x_p, which in turn can be calculated as follows. Using similarity of triangles again, we get:

$$y_p - y_2 = \frac{(y_2 - y_1)(x_p - x_2)}{x_2 - x_1} \tag{5}$$

Also, using Pythagoras' theorem,

$$(x_p - x_2)^2 + (y_p - y_2)^2 = v^2(t_p - t_2)^2 \tag{6}$$

Substituting for $y_p - y_2$ from Equation 5 in the above and solving for x_p, we get:

$$x_p = x_2 + \frac{v(t_p - t_2)(x_2 - x_1)}{[(x_2 - x_1)^2 + (y_2 - y_1)^2]^{1/2}} \tag{7}$$

In the case of updates that include information about the direction of motion of the nodes, only 1 previous update is required to predict a future location. The calculation of the predicted location at a correspondent node a is then exactly the same as the periodic calculation of the expected location (x_e, y_e) at the node b itself. This calculation of (x_e, y_e) was described in the previous subsection.

Our mechanism below can be easily extended to nodes having non-zero angular velocity, i.e. nodes not having piecewise linear mobility pattern. Instead of 2 previous updates, 3 are needed in this case [1]. If we apply our 2 update based prediction method to a node with non-zero angular velocity, then its motion curve will be approximated as a series of piecewise linear line segments.

Delay Prediction: In our current approach, we assume that the end-to-end delay for a data packet from a to b will be the same as the delay experienced by the latest update from b to a. Our results show that end-to-end delays for packets travelling between a and b usually remain more or less similar for no more than 0.5 seconds. We found that the difference in end-to-end delay, experienced by two packets travelling more than 0.5 seconds apart from a, even to a neighbor b, is likely to be $\geq$ 20 ms. In other words, we can say that the distribution of the end-to-end delays between a and b has a "memory" of no more than 0.5 seconds. The reason is that the dynamic nature of the ad hoc network causes some change in topology or in traffic characteristics during this interval of approximately 0.5 seconds that completely alters the end-to-end delays between a and b by more than 20 ms.

[1] Due to space constraints, the mathematical derivation of the prediction equations for this special case has not been included in this paper.

Therefore, the delay experienced by an update arriving at a from b, more than 0.5 seconds before the current time t, is usually not approximately equal to the end-to-end delay experienced by a packet leaving a for b at time t. Consequently, we do not use older updates in our delay prediction, only the most recent one. We just predict that the end-to-end delay *to* node b will be equal to the delay experienced by the most recent update that has arrived *from* b. The usage of this mechanism will be further justified in subsection 3.4, when we explain in-band updates.

Although this section describes the prediction mechanism only with respect to a destination b, the correspondent node a performs location prediction at the time of routing for the destination as well as intermediate nodes. The delay prediction for each node is based on the end-to-end propagation delay for that node from a.

3.3 QoS Routing

The location-and-delay prediction is extensively used in the QoS routing decisions. The presented QoS routing algorithm belongs to the source routing category where each node a has information about the whole topology of the network. It can thus compute the route from itself to any other node, using the information it has, and can include this source route in the packet to be routed.

The state information about the network stored at a node a consists of two tables - the *update table*, shown in figure 5, and the *route table*. The update table contains information pertaining to every node that a receives updates from, i.e. every node in the network. When an update packet arrives at a node a from a node b, the information it contains is used to update this table. The update table stores the node ID of b, the time the update packet was sent, the time it was received, the geometric co-ordinates of b as contained in the update packet, the speed of b as contained in the update packet, the resource parameters of b and, optionally, the direction of motion of b.

In the update table, the entry for some node k also contains a *proximity list* which is a list of all nodes lying within a distance of 1.5 $\times$ transmission range (called *proximity distance*) of k. The proximity list is used at the time of QoS route computation. During route computation, the source a is required to compute the neighbors of the intermediate node k (i.e. nodes lying within k's transmission range) that can be used as next-hops from k. However, if we only maintain a neighbors list for k (i.e. list of nodes lying within k's transmission range) rather than a proximity list, then nodes that were outside k's transmission range at the time of their respective last updates, but have since moved into it, will not be considered. We choose 1.5 $\times$ transmission range as the proximity distance because we assume that a node,

node ID	update number	send time	recv time	location		speed	proximity list	resource parameters			direction (optional)
				x	y			power	range	stability	
1	1										
	2										
2	1										
	2										
...											

Figure 5. Update table at a to store location and resource information about other nodes in the network, obtained from their updates

which is not even within distance 1.5 $\times$ transmission range of k at update reception time t, would not have moved within the transmission range at route computation time $t + \delta t$. Thus, based on an updating node b's location update, node a inserts it into the proximity list of all the nodes k in its update table that it lies within 1.5 $\times$ transmission range of, at the time of reception of the update.

To make the location-and-delay predictions, a needs to store the last 2 (for piecewise linear motion) or 3 (for angular motion) updates of b in its update table. The node a receives updates from all nodes in the network, i.e. all possible destinations b. Thus, the update table at a contains the whole view of the network for a's reference.

The route table at a node a contains all the routes originating at a, i.e. all the connections for which a is the source. When an update is received at a node a, it checks if any of the routes in its route table is broken or is about to be broken. A route, in our protocol, could be broken either when neighboring nodes move out of each other's transmission ranges or, alternatively, if the resource availability at some intermediate node changes and this results in the node being unable to support the QoS requirements of the connection. In either case, route re-computation should be initiated. Due to the location prediction based on the updates, we can predict if neighboring nodes on a route are about to move out of each other's transmission range. Thus, route re-computation could be initiated *before* the route actually breaks.

In our simulations, the update packet contains percentage of battery power remaining, transmission range and the motion stability parameter as the resource parameters. Other resource information such as CPU speed could also be added to the update packet. In addition, the end-to-end delay can be calculated from the send and receive time of the update packet. Thus a obtains a resource tuple $<$ power, range, stability, delay $>$ for each node in the network, which it uses in making forwarding decisions.

When a connection is required from a to the destination b, the middleware level of the node a specifies some QoS

```
**global** $stack$;
$/*\ We\ assume\ function\ compute\_cl\ which\ performs\ location - delay\ prediction\ */$
$/*\ and\ admission\ control\ for\ each\ node\ in\ the\ source's\ proximity\ list\ to\ */$
$/*\ obtain\ a\ list\ of\ candidate\ next - hops\ */$
**proc** $\mathit{QoS\_routing}(src, dest, < qos\_requirements >) \equiv$
  $valid\_routes := \{\}$;
  $cand\_list := \mathit{compute\_cl}(src, < qos\_requirements >)$;
  **if** $cand\_list\ != \{\}$
    **then if** $dest \in cand\_list$
          **then** $Directly\ forward\ packet\ to\ destination$;
          **else foreach** $c \in cand\_list$ **do**
                $stack := \{\}$;
                $\mathit{push}(stack, c)$;
                $\mathit{find\_route}(c, dest, < qos\_requirements >)$
              **od**
        **fi**
  **fi**
  **if** $valid\_routes = \{\}$
    **then** $No\ route,\ reject\ connection$;
    **else** $Output\ shortest - distance\ route\ from\ valid\_routes$;  **fi.**
**proc** $\mathit{find\_route}(start, dest, < qos\_requirements >) \equiv$
  $cand\_list := \mathit{compute\_cl}(start, < qos\_requirements >)$;
  **if** $cand\_list\ != \{\}$
    **then foreach** $n \in cand\_list$ **do**
          **if** $n \notin stack$  $/*\ to\ eliminate\ loops\ */$
            **then** $\mathit{push}(stack, n)$;
                **if** $n = dest$
                  **then** $valid\_routes := valid\_routes \cup stack$;
                  **else** $\mathit{find\_route}(n, dest, < qos\_requirements >)$;  **fi**
                $\mathit{pop}(stack)$;
          **fi**
        **od**
  **fi.**
```

Figure 6. Algorithm to find QoS route

requirements to the network level for this connection. The QoS requirements are represented in the form of a tuple $<$ estimated duration of connection, maximum delay, maximum delay jitter $>$. The connection duration can be mapped to a minimum required battery power at the intermediate nodes. The longer the connection duration, the higher the remaining battery power required at the intermediate nodes. Similarly, the maximum delay jitter allowable for the connection can be mapped onto the motion stability factor. If larger values of jitter are tolerable, then unstable nodes can be used as intermediate nodes to support the connection. The maximum delay QoS requirement can be mapped onto the end-to-end delays observed for the updates from b to a. A node that cannot satisfy the QoS requirements is not used in the routing. Thus, given the resource availability at the nodes and the QoS requirements of the connection, “admission control” can be performed.

When a connection request arrives at the network level of a, it first runs a location-delay prediction on each node in its proximity list and thus obtains a list of its neighbors at the current time. Then, it finds out which of these neighbors have the resources - maximum delay, required stability and battery power - to satisfy the QoS requirements of the connection. (We call these neighbors that satisfy the QoS requirements as “candidates”.) It then performs a depth-first search for the destination starting at each of these candidate neighbors to find all candidate routes. At each step of the depth-first search, only the neighbors with resource levels satisfying the QoS requirements of the connection are considered. Also at each step, a node checks if it can hear link layer transmissions from these candidate neighbors. This is just to ensure that even though the neighbor is close enough geographically, there is no physical impediment in between causing the signal to fade.

Thus, route discovery and admission control proceed hand-in-hand at every hop. From the resulting candidate routes, the geographically shortest one is chosen as the route to forward the packet along. The packet is then forwarded along this route to the destination. The source route is included in the packet. If the route breaks at any point, route repair can be initiated from the node where the route has broken.

After connection establishment, packets are forwarded along the computed source route until the end of the connection or until the route is re-computed due to breakage or in anticipation of breakage.

The QoS routing algorithm is illustrated in Figure 6. We assume there exists a function $compute\_cl$ that computes, in two steps, the candidate list of a particular node given the QoS requirements and its proximity list. The function $compute\_cl$ first performs location-and-delay prediction for each node in the proximity list of the $start$ node to figure out if it is a neighbor. Then, among all the neighbors, it checks which nodes have resources to support the connection. The resulting list of nodes is the candidate list. $valid\_routes$ is a list of routes containing all the valid routes returned after the depth-first search. The procedure $find\_route$ is called for each candidate neighbor of the source. It performs the depth-first search for the destination. When the destination is found, the route stored in the variable $stack$ is added to the $valid\_routes$ list. The main procedure $QoS\_routing$ rejects the connection if no valid routes are found. Otherwise, the connection is accepted and the list $valid\_routes$ is then examined to find the route that is shortest in terms of geographic distance from source to destination. As admission control has been done at the time of determining valid routes, all routes contained in $valid\_routes$ satisfy the QoS requirements of the connection.

The algorithm outputs the best loop-free path from source to destination if at all one exists.

3.4 Protocol Enhancements

The basic update protocol and the location prediction mechanism can be further enhanced to provide more accurate location prediction. We will present the following enhancements: (1) direction information included in the update, (2) in-band updates piggy-backed on data packets and (3) double type 2 updates.

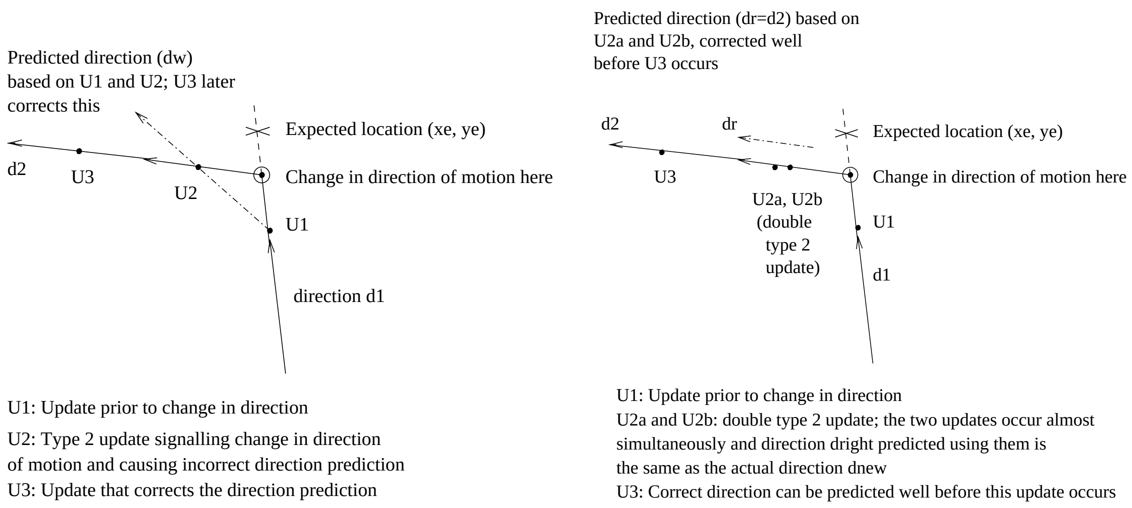
Notice that there is a clear tradeoff between the enhancements to the update protocol to achieve finer prediction accuracy and the extra overhead on the underlying ad hoc network.

Direction Information Included in Update: In our basic mechanism, the update packet does not include direction information as this can be easily determined using the latest 2 updates. However, if the mobile node has changed its direction in the intervening time between the latest 2 updates, then the direction obtained from the latest 2 updates is inaccurate. On the other hand, if the exact direction of motion is included in the update itself, the correspondent node can always obtain the correct direction of motion of the destination node at the time of prediction. The down side is that this increases the size of the update packet. In our solution, we used 8 extra bytes to encode the direction information into the update packet.

In-band Updates Piggy-backed on Data Packets: Another enhancement to the basic update protocols involves the piggy-backing of in-band updates on data packets and acknowledgements between mobile end hosts. This increases the frequency of updates and hence increases the accuracy of both the location prediction and the end-to-end delay prediction. It increases the accuracy of the latter prediction because end-to-end delays are correlated over a period of 0.5s. If there is a data packet with update information piggy-backed on it arriving within 0.5s of the previous update, then the next delay prediction, based on this newer update, is likely to be very accurate.

The down side, once again, is that this increases overhead on the network because data packets are now much larger since they have update information piggy-backed on them. The overhead can be controlled by piggy-backing updates on selected data packets.

Double Type 2 Updates: The use of double type 2 updates is best illustrated with an example. In our basic update protocol (updates without direction information), for piecewise linear motion, we use 2 previous updates for prediction. Now, consider a node that experiences a significant change in direction of motion. As it moves a distance


Predicted direction (dw)
based on U1 and U2; U3 later
corrects this
Expected location (xe, ye)
d2
U3
U2
Change in direction of motion here
U1
direction d1
U1: Update prior to change in direction
U2: Type 2 update signalling change in direction
of motion and causing incorrect direction prediction
U3: Update that corrects the direction prediction
Predicted direction (dr=d2) based on
U2a and U2b, corrected well
before U3 occurs
d2
dr
Expected location (xe, ye)
U3
Change in direction of motion here
U2a, U2b
(double
type 2
update)
U1
d1
U1: Update prior to change in direction
U2a and U2b: double type 2 update; the two updates occur almost
simultaneously and direction dright predicted using them is
the same as the actual direction dnew
U3: Correct direction can be predicted well before this update occurs

(a) Error in determining direction of motion when using default prediction scheme

(b) Remedied situation using double type 2 updates

Figure 7. Double type 2 updates

δ away from its expected path, it generates a type 2 update indicating its current position along its new path. This situation is illustrated in Figure 7(a). In the figure, the old direction is marked d_1 and the new direction is marked d_2. The type 2 update is marked as U_2.

However, predictions are based on the last 2 updates. Hence, the obsolete update previous to the recent type 2 update, marked in Figure 7(a) as U_1, also plays a part in the prediction. This skews the direction prediction. A wrong direction of motion of the node is thus predicted and this is illustrated in Figure 7(a) as d_w. A correspondent node continues to predict this wrong direction based on the last two updates it has received, U_1 and U_2, until the *next* update, U_3 corrects this. U_3 *confirms* the changed direction and both of the latest 2 updates used in prediction, U_2 and U_3 are now current and not obsolete.

By using double type 2 updates [2], we artificially make obsolete U_1 right after the change in direction, well before U_3. The two type 2 updates constituting the double, U_{2a} and U_{2b}, are now used as the latest 2 updates in the predictions and the obsolete U_1 update has no role to play. This remedied situation is illustrated in Figure 7(b). The figure shows U_{2a} and U_{2b} occuring almost simultaneously. We assume that they also reach the correspondent node almost simultaneously. The two most recent updates that the correspondent node now has are therefore U_{2a} and U_{2b}. The

[2]The protocol is statically configured to either use double type 2 updates or single type 2 updates. There is no dynamic choice between the two at runtime.

direction predicted using these two updates is d_r, which is the same as d_2, the actual direction. A wrong direction is predicted only for the negligibly short time interval between the reception of U_{2a} and the reception of U_{2b}. The increase in number of updates is what leads to the increase in overhead in this case.

3.5 Optimization of the Update Protocol

The update protocol in this paper involves flooding of location and resource information pertaining to a node to all the other nodes in the network. Ordinarily, such a full flooding of the network involves a very large overhead. However, with schemes such as the MPR scheme [2], the overhead associated with flooding can be considerably reduced. In the MPR scheme, certain nodes are elected as multi-point relays (MPRs) for their neighborhoods. Nodes that are not MPRs receive and process the flooded message from their neighborhood MPRs, but do not re-broadcast it. Only the designated MPRs re-broadcast the flooded message. Thus, overhead is reduced because there are fewer copies of the message in the network as compared to the number there would be if full flooding was done.

4 Performance Results

We performed extensive simulations using the ns-2 network simulator to evaluate the performance of our prediction scheme, the overhead involved in the updates and the throughput of the QoS routing scheme. Our simulations were conducted using a network topology consisting of 40 nodes spread over a 1100m $\times$ 1100m area. We propose to make our protocol scalable to larger networks through the use of hierarchy and clustering. We can limit the update flood within the cluster and have only cluster-border nodes propagate their updates to neighboring clusters. The velocity of each node in our simulations varied dynamically between 0 and 15m/s. We used the random waypoint mobility model provided by the mobility extensions to the ns-2 simulator. We ran our simulations over a time interval of 150 seconds. To simulate background traffic, we started 10 TCP connections between various nodes.

4.1 Evaluation of Update and Prediction Mechanisms

We use a *prediction error* metric to gauge the accuracy of our prediction mechanism. This prediction error is defined as the percentage of predictions that were off the actual location by a particular distance. We measure this prediction error under various conditions such as location-only prediction, location-and-delay prediction, constant time type 1

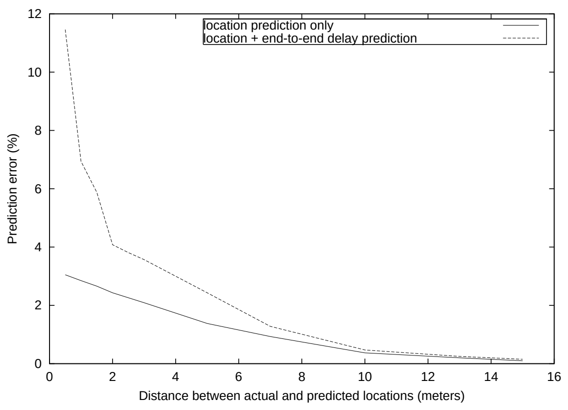
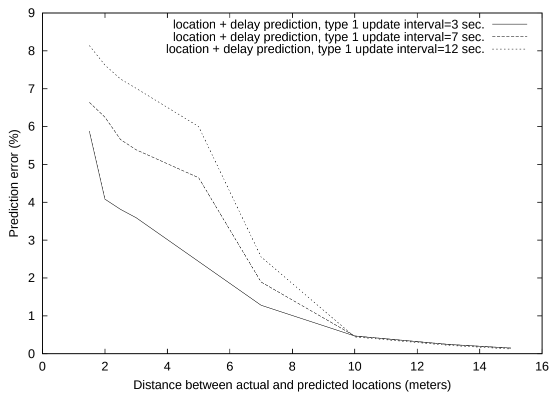
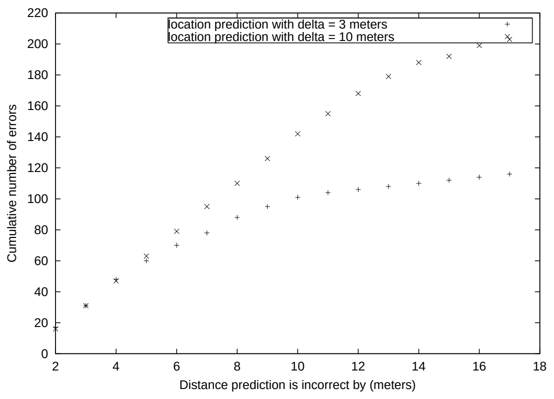


Figure 8. Accuracy of location + delay predictions as compared to location only predictions



(a) Accuracy of location + delay predictions for various constant inter-update time intervals



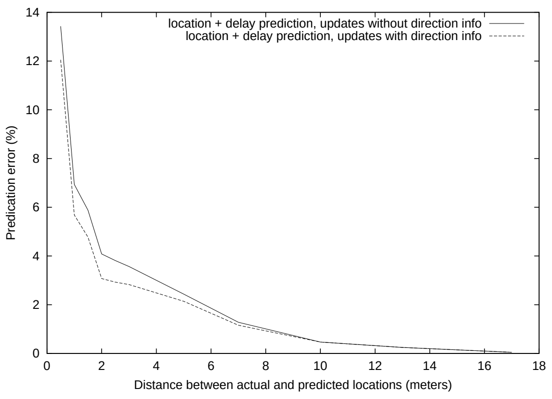
(b) Cumulative number of errors for various error distances

Figure 9. Evaluation of Update Protocol for various Type 1 inter-update intervals and various Type 2 error distances

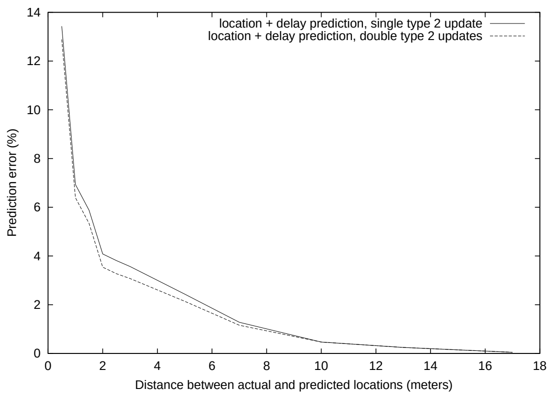
update intervals and for the enhanced versions of the update protocol.

Figure 8 shows the accuracy of location-only predictions as compared to location-and-delay predictions. For location-only predictions, we find the difference between actual and predicted location of the receiver at the time when the packet is transmitted from the sender, rather than at the time it is received. This means that we do not have to take into account the end-to-end propagation delay of the packet from sender to receiver. For location-and-delay predictions, the inaccuracy in the delay prediction adds to the inaccuracy of the location prediction, but our predictions are still accurate within 1 meter in more than 90% of the cases.

Figure 9(a) shows the accuracy of location-and-delay predictions obtained for various constant values of inter-update intervals for type 1 updates. As expected, when the update interval increases, the inaccuracy in the predictions



(a) Improvement in prediction accuracy over default scheme due to direction information included in updates



(b) Improvement in prediction accuracy over default scheme due to use of double type 2 updates

Figure 10. Evaluation of Protocol Enhancements

increases. The curves merge for prediction errors of larger than 10 meters. This is because these errors are caused by sudden change in the pattern of motion. The changes occur at a much smaller time-scale than the inter-update intervals for the type 1 update. These residual errors occur before the type 2 update indicating the changed pattern of motion can propagate to the correspondent node.

Figure 9(b) shows the location-only prediction error, represented by the distance between actual and predicted location, for various values of δ. We predicted the location at regular intervals of 40ms over our entire simulation run of 150 seconds. We measured, for each prediction, how far our predicted location was from the actual location. The prediction error metric here is the cumulative number of errors rather than the percentage of errors so that we can show clearly how the choice of δ affects the distribution of errors over various error distances.

For $\delta = 3$ meters, there were 116 errors of ≥ 1.0 meters, out of which 17 were between 1.0 and 2.0 meters, 31 were between 1.0 and 3.0 meters, and so on. This is represented by the curve corresponding to $\delta = 3$ in figure 9(b). We find that the number of errors is uniformly distributed for error distances less than δ for both curves. (Curves rise linearly approximately upto the value of δ.) The number of errors for error distances *greater* than δ are considerably fewer because our type 2 updates filter out most of these errors. This underlines the value of our type 2 updates. Some residual errors, however, remain because of the delay in detecting the deviation of δ from the expected location and the delay in propagating the type 2 update.

Figure 10(a) illustrates the improvement obtained in prediction accuracy by using direction information. Figure

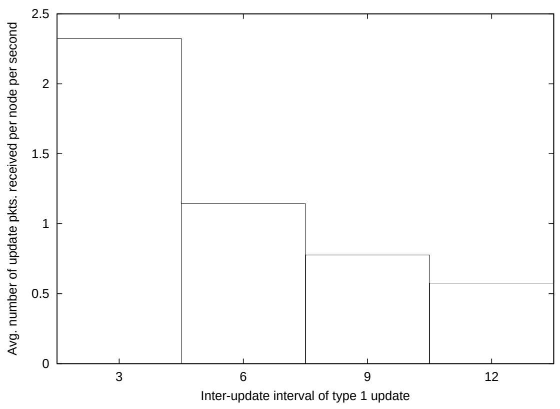


Figure 11. Update overhead for various type 1 inter-update intervals

10(b) illustrates the improvement obtained in prediction accuracy using double type 2 updates. The results show that by using these protocol enhancements, the prediction accuracy only improves marginally over the default scheme. However, there is an increase in the update overhead.

Figure 11 shows the update overhead, without the use of any optimizations, measured in terms of the average number of update packets received per node in the network per second, for various type 1 inter-update intervals. δ is kept constant at 3 meters for this experiment.

4.2 Evaluation of QoS Routing Protocol

The simulation parameters (number of nodes, node velocity, background traffic, etc.) used in the evaluation of the QoS routing protocol ware identical to that used in the evaluation of the update mechanism. Using a type 1 inter-update interval of 3 seconds and with δ=3 meters, we found that route re-computation needed to be performed at an average of every 0.53 seconds. Using TCP traffic, we also obtained an average throughput (ratio of number of packets successfully forwarded to destination divided by total number of packets transmitted) of 69.10%. The remaining packets or their acknowledgements were dropped and the packets thus had to be re-transmitted. The average throughput measured in terms of packets delivered successfully to the destination per second is 141.91 pkts/sec.

5 Conclusion

In this paper, we have presented a new QoS routing protocol based on the prediction of location of nodes in an ad hoc network. Our predictive solution overcomes the problem caused when routing information used in existing ad hoc routing protocols becomes obsolete due to node migration. Both the location prediction scheme, the performance of which we have evaluated using simulations, and QoS routing protocol, which we are currently simulating, depend on a low-cost location-resource update protocol. Our simulations demonstrate a high degree of accuracy in location prediction, which can further be improved using the protocol enhancements suggested in the paper, at the cost of increased update overhead. Our QoS routing algorithm is robust and provides loop-free paths from source to destination.

References

[1] S. Basagni, I. Chlamtac, V. Syrotiuk and B. Woodward. A Distance Routing Effect Algorithm for Mobility (DREAM). *Proc. of ACM MobiComm'98*, October, 1998.

[2] P. Jacquet, P. Muhlethaler and A. Qayyum. Optimized Link State Routing Protocol. *ftp://ftp.ietf.org/internet-drafts/draft-ietf-manet-olsr-00.txt*, 1998.

[3] D. Johnson and D. Maltz. Dynamic source routing in ad hoc wireless networks. *Mobile Computing, chapter 5, pages 153–181. Kluwer Academic Publishers*, 1996.

[4] Y. Ko and N. Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. *Proc. of ACM MobiCom'98*, October, 1998.

[5] C. Perkins and P. Bhagvat. Highly dynamic destination-sequenced distance vector routing for mobile computers. *Proc. of ACM SIGCOMM '94*, October, 1994.

[6] C. Perkins, and E. Royer. Ad-hoc On-Demand Distance Vector Routing. *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, pp. 90-100*, February, 1999.

[7] W. Su, S.-J. Lee and M. Gerla. Mobility Prediction in Wireless Networks. *Proc. of IEEE MILCOM 2000*, October, 2000.