Wireless Netw (2010) 16:2065–2086
DOI 10.1007/s11276-010-0244-4

Using directionality in mobile routing

Bow-Nan Cheng · Murat Yuksel · Shivkumar Kalyanaraman

Published online: 20 March 2010

Abstract The increased usage of directional methods of communications has prompted research into leveraging directionality in every layer of the network stack. In this paper, we explore the use of directionality in layer 3 to facilitate routing in *highly mobile environments*. We introduce Mobile Orthogonal Rendezvous Routing Protocol (MORRP) for mobile ad-hoc networks (MANETs). MORRP is a lightweight, but scalable routing protocol utilizing *directional communications* (such as directional antennas or free-space-optical transceivers) to relax information requirements such as coordinate space embedding, node localization, and mobility. This relaxation is done by introducing a novel concept called the directional routing table (DRT) which maps a *set-of-IDs* to each interface *direction* to provide *probabilistic* routing information based on interface direction. We show that MORRP achieves connectivity with high probability even in highly mobile environments while maintaining only probabilistic information about destinations. Additionally, we compare MORRP with various proactive, reactive, and position-based routing protocols using single omni-directional interfaces and multiple directional interfaces and show that MORRP gains over 10–14 × additional goodput vs. traditional protocols and 15–20% additional goodput vs. traditional protocols using multiple interfaces. MORRP scales well without imposing DHT-like graph structures (eg: trees, rings, torus etc). We also show that high connectivity can be achieved *without* the need to frequently disseminate node position resulting increased scalability even in highly mobile environments.

Keywords Mobile ad hoc networks · Directional antennas · Free space optical · Routing

1 Introduction

A recent trend in wireless communications has been the desire to leverage directional forms of communications (e.g. directional smart antennas [1, 2], Free-Space-Optical transceivers [3, 4], and sector antennas) for more efficient medium reuse, increased scalability, enhanced security and potential for higher achievable bandwidth. Previous work in directional antennas focused heavily on measuring network capacity and medium reuse [2, 5]. In these works, it was shown that with proper tuning, capacity *improvements* using directional over omnidirectional antennas are dramatic—even just eight directional interfaces results in a theoretical capacity gain of 50× .

Additionally, there has been a large push in the free space optical (FSO) community to use FSO to compliment traditional RF methods [3]. FSO has several attractive characteristics like: *(1) dense spatial reuse*, *(2) low power*

A preliminary version of this paper appeared in IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS) 2008. This material is based upon work supported by the National Science Foundation under Grant Nos. 0627039, 0721452, 0721612 and 0230787. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

B.-N. Cheng (✉)
ECSE Department, Rensselaer Polytechnic Institute,
Troy, NY, USA
e-mail: bownan@gmail.com

M. Yuksel
Reno ECSE Department, University of Nevada, Reno, NV, USA

S. Kalyanaraman
IBM India Research Laboratory, Bangalore, India

usage, *(3) license-free band of operation*, and *(4) relatively high bandwidth compared to RF* but suffers from: *(1) the need for line of sight (LOS) alignment* and *(2) reduced transmission quality in adverse weather conditions*. Yuksel et al. [4] proposed several ways to mitigate these issues by tessellating low cost FSO transceivers in a spherical fashion and replacing long-haul point-to-point links with short, multi-hop transmissions.

Given the seemingly large increases in medium reuse and potential for higher bandwidth in directional forms of communications, it becomes interesting to investigate how *directionality* can be used to complement and even enhance wireless networks in all layers of the stack. There are several challenges associated with using directionality in mobile networks. Unlike omnidirectional antennas where neighbor reach depends almost exclusively on range, nodes using directional antennas need also take into account the neighbor's direction and map it to a specific interface in that direction. The problem is complicated even further as nodes closer to a source seemingly incur more dynamism (even small movements can affect perceived direction dramatically) while nodes farther away incur less change.

In this paper, we address these issues and propose utilizing directionality for a novel purpose: to facilitate layer 3 routing in *highly mobile environments* without the need for flooding either in the route dissemination or discovery phase. Most prior work on leveraging directional antennas in the routing layer focus on adapting routing protocols to simply *utilize* directional communications [6, 7]. Our work is novel in that we utilize *local* directionality as a property *to* route packets itself. To the authors' best knowledge, this is the first paper that attempts to use directionality to address issues with *high mobility*. Figure 1 illustrates this insight. Let's assume a node has eight directional antennas oriented such that they are facing different directions, providing full coverage. Traditional routing protocols take advantage of the directional antenna only in sending unicast packets. Broadcast packets such as hello packets, however, continue to be flooded out each interface to cover the full spread. By leveraging a node's local sense of direction, however, an extra degree of diversity is added.

Our protocol, Mobile Orthogonal Rendezvous Routing Protocol (MORRP) is based on two fundamental primitives:

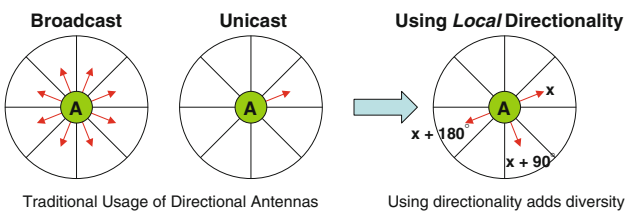


Fig. 1 Using a node's sense of direction allows for an extra degree of diversity to leverage

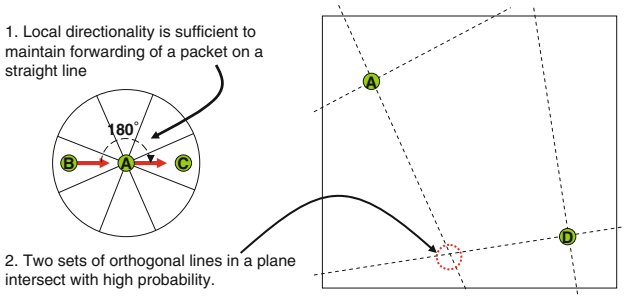


Fig. 2 By forwarding out 180 degrees from the angle of receipt, one can effectively maintain straight line paths in transmitting. In a plane, two sets of orthogonal lines intersect with a high probability

(a) local directionality is sufficient to maintain forwarding of a packet on a straight line, and (b) two sets of orthogonal lines in a plane intersect with high probability even in sparse, bounded networks. Figure 2 illustrates these primitives. Suppose that Node A receives a packet from Node B from the transceiver direction shown. By sending the packet out the transceiver 180° from the angle of receipt to Node C, Node A can effectively maintain a relatively straight line transmission path. In a plane, two sets of orthogonal lines originating at separate points have a high probability of intersect.

Cheng et al. [8] showed that in *static* wireless mesh networks, by forwarding packets to nodes intersected by a pair of orthogonal lines originating from a source and destination, one can successfully route packets to a high degree of connectivity (98%) without the need for coordinate space. Furthermore, it was shown that forwarding using this method state-scales to $O(N^{3/2})$ with the states spread evenly throughout the network, while incurring a path stretch vs. shortest path of only 1.2.

The protocol proposed in [8] relies on nodes to periodically send out announcement packets proactively in orthogonal directions with each node along the path to store the state in a routing table. When a node desires to send to a destination, it sends out a route request (RREQ) packet reactively in orthogonal directions. When the RREQ packets intersect a node that has state information about a destination, that node becomes the rendezvous node/point and a route reply (RREP) packet is sent back to the source. What results is a two-phased path from source to rendezvous node to destination. Unfortunately, the proposed protocol fails under even slight mobility because straight-line paths and rigid "destination— next-hop" routing tables are hard to maintain in the presence of node dynamism.

MORRP facilitates high mobility by abstracting the concept of rendezvous *points* to rendezvous *regions* and forwards packets *probabilistically* based on which direction a destination or rendezvous node is most likely found. These directions shift accordingly to a node's *local*

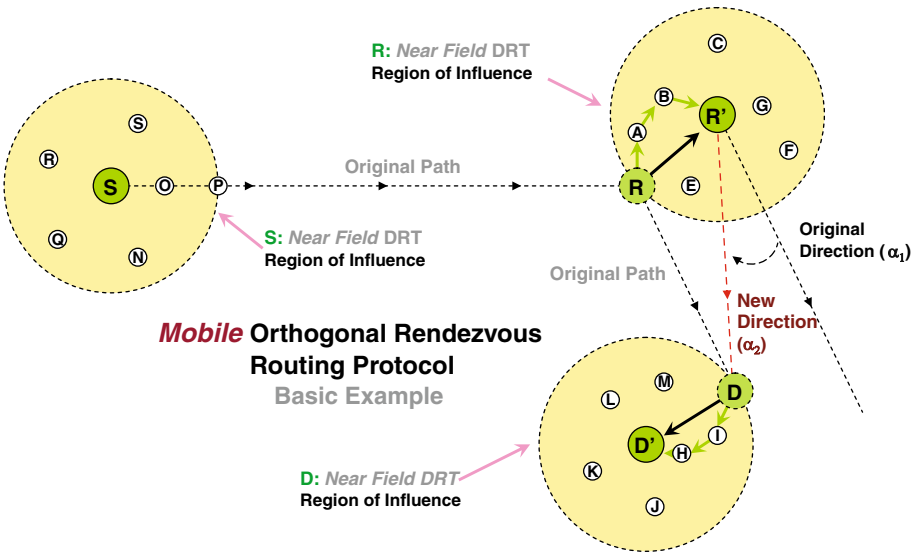


Fig. 3 Basic MORRP Example: Source sends to rendezvous node R found in region illustrated which in turn sends to destination D found in the DRT region given

velocity. For example, if a source node is moving north, a node originally *east* of the source will *seem* to be moving *south*.

Figure 3 illustrates a basic example. Suppose source S wants to send packets to destination D and through announcement and route request (RREQ) packets, the path "Original Path" is established between S and D with node R as the rendezvous node. After some time, node R has moved to R' and node D has moved to D'. With *infrequent updates* in a mobile environment, node R wishes to maintain a general direction to node D based solely on *local* information (its own mobility pattern) and adjusts its direction of sending to D from angle α_1–α_2. All nodes maintain a "field of influence" where each node knows the relative direction to all nodes in its region. The data packets S sends to D will traverse the original path, "gravitating" toward R' once it hits R's field of influence. Then, it will be sent in the modified direction of D until it hits D's field of influence and "gravitates' toward the destination.

MORRP routes packets using directionality in highly mobile environments by: (1) shifting destination node directions based on a node's *local* velocity and (2) increasing probability of finding nodes by introducing "fields of influence". All of this is done through a novel replacement to routing tables we formulate called the *directional routing table* (DRT).

The concept behind DRTs is simple: instead of maintaining *destination IDs* to *next-hop IDs*, we map a *probabilistic set-of-IDs* to each *interface direction* as shown in Fig. 4. The set-of-IDs are stored in bloom filters that are aggregated and sent to neighbors who merge them with the set-of-IDs associated with the interface of receipt. The information in the filter becomes less useful as we progress in time and space and thus we decay (remove bits) from each bloom filter before sending it to its neighbors to capture this effect. Closer nodes have "more" information because the rate at which they are being updated by the source node is higher. Because DRTs only maintain information on each interface rather than on specific routes in a network, it adds more robustness to mobility as it provides several alternative paths for reaching a destination. In short, any next-hop in a particular direction can take the packet forward. Naturally, the closer a packet gets to a destination node, the information intermediate nodes have about the location of the destination increases. For destinations too far for the source to have any information about the location, MORRP relies on route request (RREQ) packets sent in orthogonal directions to *rendezvous* with state information maintained by each node along an announcement path also disseminated in orthogonal directions. This lightweight method of information dissemination ensures low control overhead from being flooded network-wide.

Key contributions of MORRP include:

- **Using directionality to solve the issues caused by high mobility in MANETs**—Using only *local* information, any node is able to more efficiently "guess" the direction of a destination and forward probabilistically.
- **The directional routing table**—A replacement for traditional routing tables based on purely *probabilistic routing*. DRTs map a *set-of-IDs* to a specific *direction* which eliminates the need to maintain exact routing information about nodes in a network while lessening the frequency of route dissemination.
- **Routing based on probabilistic hints**—Traditional routing protocols have a hard limit on route expiration. With probabilistic routing, routing information is decayed with time and becomes less and less accurate.

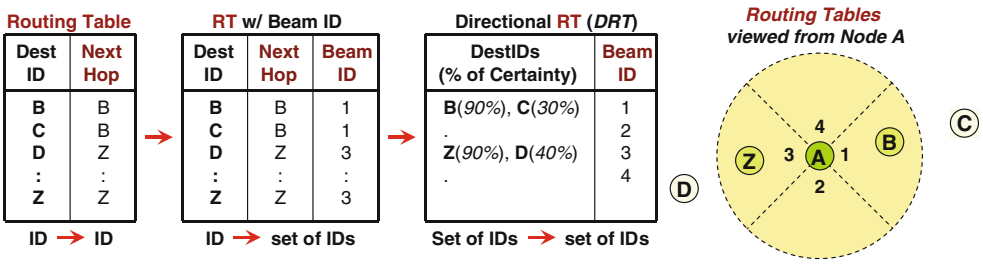


Fig. 4 Directional Routing Tables (DRTs) map a direction to a set-of-IDs stored in bloom filters

Below a certain threshold, the information becomes insignificant.

In comparing with several proactive, reactive, and position-based routing protocols, MORRP shows high data delivery (93%+), low packet overhead, and over 10-14X goodput gains vs. traditional routing protocols and 15-20% goodput gains vs. traditional routing protocols modified with multiple directional interfaces in highly mobile (30 m/s) environments. These gains come from many *key design factors*:

- **Weak state information and probabilistic routing**—MORRP does not maintain complete paths and is thus more flexible to forward packets in mobile environments.
- **Local update of weak state information**—Adjusting the "general direction" of a destination node based on one's local velocity "takes a packet forward" even with infrequent location updates.
- **Field of influence**—Enlarging the intersection area results in a greater probability of finding a path in highly mobile environments even with infrequent updates.
- **Leveraging local direction information**—Limited flooding is curtailed by using local directionality to forward in straight lines and rely on intersections of announcement and route request packets to "find" potential paths. This results in "freeing up" the medium for data. This is especially important because MORRP is a hybrid proactive/reactive protocol.

The rest of the paper is organized as follows: Sect. 2 and 3 outline the concept of MORRP including a detailed explanation of DRTs and several decaying strategies as well as how route information would be disseminated and maintained. Section 4 gives a basic numerical analysis on path intersection probability while Sect. 5 gives some simulation performance evaluations. Finally, section ref-sec:conclusion presents some thoughts on future work and concludes the paper.

2 The directional routing table

One of the underlying mechanisms behind MORRP's *probabilistic* forwarding strategy is the directional routing table (DRT), a simplified method of storing route information by leveraging directional communications methods. Unlike traditional routing tables which map *destination-IDs* to *next hop IDs*, DRTs map a *set of IDs* to a specific interface direction. In other words, all the nodes covered by the transmission sector of a specific antenna are included in the entry for that interface in the DRT. The number of entries in the DRT remains constant based on the number of interfaces and does not grow even as the number of nodes in the network grows. This is done through bloom filters.

The concept of using *bloom filters* in probabilistic routing schemes is not new. Acer et al. [9] and Kumar et al. [10] have suggested novel, decentralized, and scalable ways on how information can be disseminated in various types of networks using bloom filters. Bloom filters are space efficient probabilistic data structures that are used to test whether an element is a member of a set. Given an array of bits A (the bloom filter) initialized to all 0 and a fixed number (k) of hash functions ($h_1(.)$, ...$h_k(.)$), elements (x) are inserted into the bloom filter by evaluating the element in each hash function and mapping the resultant locations in the array to one ($h_i(x) = 1$, $i = 1, 2, ..., k$). Lookups are done in the same way in that if the positions in the bit array corresponding to the hashes of an element all equal 1, then the element is a member of the set.

Kumar et al. [10] introduced exponential decay bloom filters (EDBF), a data structure based on the traditional bloom filter concept. Instead of testing whether an element is part of a set or not (absolute information), EDBFs *count* the number of 1's in the bit array corresponding to the element hash in lookup ($\theta_x = |\{i|A[h_i(x)] = 1, i = 1, 2, ..., k\}|$). The fraction of bits set to 1 over the number of hash functions can be used to interpret the certainty of an element being in the set. Bits are "dropped" (decayed) using various strategies. In this paper, we apply the concept of EDBFs to store a probabilistic set-of-IDs corresponding to neighbor nodes a sector antenna covers in a MANET. We generalize the term to decaying bloom filter (DBF) as there are many ways to decay bloom filters.

Figure 4 outlines the structure for the DRT. In short, a *set-of-IDs* stored in a decaying bloom filter is mapped to each specific *interface direction*. To find the certainty of

reaching a node by sending out a specific interface, the DBF associated with the interface is selected and the destination node ID is sent through each hash function. By counting the number of bits set to "1" in the locations where the hashes land, the level of certainty of reaching a destination node by sending out that interface is obtained. As time goes on and without frequent updates, the level of certainty decreases. To facilitate this idea, we decrease the level of certainty by "decaying" bits in the bloom filter (i.e. changing bits in the DBF from 1 to 0). Decaying methods can be broken up into two main thrusts: *intra-node decay* which handles how bits are removed to simulate that as time goes on, there is *less certainty* about information, and *inter-node decay* which dictate how bits are removed as information is passed from node to node, simulating that nodes farther away know *less* about a node than nodes closer to the node. In the following subsections, we detail each method.

2.1 Intra-node decay

2.1.1 Time decay

Current routing strategies employ hard timeouts for routing entries, updating routing entries periodically through route dissemination or route discovery. While effective for low mobility situations, high mobility situations can cause routes to become stale quickly if the interval between route updates is not decreased. As a result, maintaining accurate routing entries network-wide poses a huge problem as it incurs a much higher overhead. MORRP attempts to mitigate this issue by decaying the likelihood a neighbor or destination is in the direction covered by a specific interface as time moves on. In stationary environments, the probability of a neighbor being in a specific region decays at a constant rate (bits from the bloom filter are removed randomly at a constant rate).

In mobile environments, we employ a different strategy to decay neighbor location probabilities. Figure 5 illustrates the basis for our formulation of a simple *time decay* heuristic in mobile scenarios. Assuming all things constant, as a node moves away from its original position, the probability of neighbors in the direction of movement should decay slower than the nodes directly opposite of the direction of movement. In short, the velocity with which each interface perceives itself to be moving at is dependent on the angle the transceiver is from the direction of movement. As we wish to split the *intra*-node decay between *time decay* and *spread decay*, we will only use half the bits in each bloom filter in our calculations.

We formulate our *time decay* heuristic as follows:

Step 1: Suppose v_x is the speed a node is moving in the "x" direction and ϕ is the angle a specific interface is from

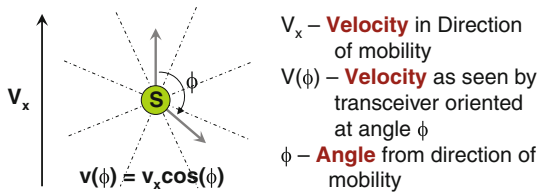


Fig. 5 Each interface has a different relative notion of how fast a specific node is traveling

the direction of movement. We define the velocity as seen by a specific transceiver v_ϕ as:

$$v(\phi) = v_x \cos(\phi) \tag{1}$$

Step 2: If we let R be the range of a transceiver, a node traveling directly *away* from a specific direction at velocity v_x would be out of the transmission region in $\frac{R}{v_x}$ seconds. As a result, we specify that *all* bits of the bloom filter in a specific interface direction must be decayed in $\frac{R}{v(\phi)}$ seconds.

Step 3: Assuming there are k bits of ones in the bloom filter for a specific interface and half of those k bits ($\frac{k}{2}$) are reserved for *time decay*, we linearly decay the number of bits in each bloom filter for each interface with respect to time and velocity. The number of bits to remove per time interval (δ_t) is:

$$\begin{aligned} \delta_t &= \frac{ktD_{tc}}{2} - \frac{ktv(\phi)}{2R} \\ \delta_t &= \frac{kt}{2}\left(D_{tc} - \frac{v_x \cos(\phi)}{R}\right) \end{aligned} \tag{2}$$

Where $k/2$ is the number of bits reserved for *time decay* ($\frac{1}{2}$ the total bits set to 1 in the bloom filter), t is the time, D_{tc} is the time decay factor in the stationary case (D_{tc} fraction of bits removed per second in the stationary case), R is the transceiver range, v_x is the velocity in a specific direction, and ϕ is the angle from the current interface to the direction of movement. These bits are removed and discarded.

2.1.2 Spread decay

In a mobile environment with directional communications, the probability a neighbor will be in a certain transmission region/sector is *stretched* over time. As time progresses, the area a neighbor is possibly located, increases. Figure 6a illustrates this concept. Suppose a neighbor announces its position to be within region 2. Without knowing what direction and velocity the neighbor is traveling at, as time progresses, there is a greater possibility that the neighbor will be in region 1 and region 3 and a lessened probability that the neighbor will be in region 2. We say that as time goes on, the "spread" for the area the neighbor is in, is increased.

In much the same way, a mobile node traversing in a certain direction will need a greater spread to cover the

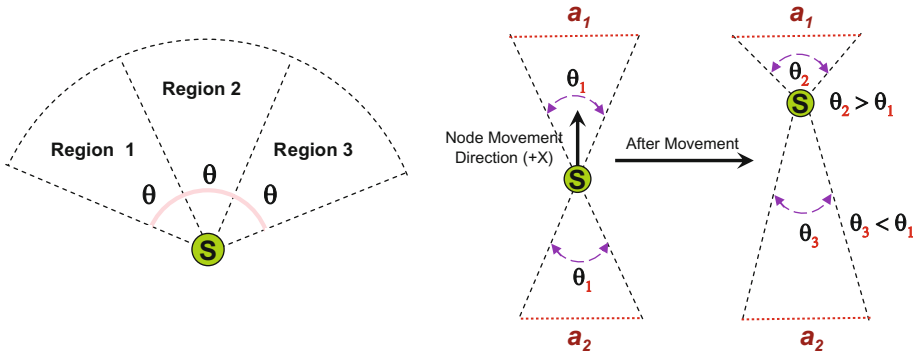


Fig. 6 Each interface/ transceiver has a specific coverage region. As a node moves in one direction, the spread overflows to regions covered by neighboring interfaces

same area in the direction it is traveling in. Figure 6b illustrates this. As a node trying to cover range θ_1 moves in the "+× " direction, it will need a greater spread, θ_2 to cover the same transmission region *in* the direction it is traveling while at the same time, a smaller spread, θ_3 to cover the same region in the direction *away* from the direction it is traveling. Each direction other than the direction the node is traveling in and the direction directly opposite has varied stretch in between these two extremes based on the angle from the direction the node is traveling.

Unlike in our *time decay* heuristic formulation, bits removed from the bloom filter are not discarded but instead, *relocated* to the surrounding directions. The inherent nature of bloom filters allows us to move bits in the DBF associated with a specific interface, to surrounding DBFs, keeping the bits set to 1 *in the same hash locations*. Due to space constraints, we do not go into details regarding spread strategies. For our simulations, we assume a simple heuristic that whatever bits were affected by the $\frac{v_x \cos(\phi)}{R}$ term in Eq. 2 are affected in the opposite way for spread decay (ie: if the bits were removed, they are spread). When there is no mobility, there is no spread decay. It is important to note the *duality* of *time* and *spread* decay: A neighbor in the direction of travel will incur *less* time decay but at the same time, *more* spread decay.

2.2 Inter-node decay

The general idea behind decaying the information transferred *between* nodes is that nodes "closer" to a specific source will most likely have more accurate information about the location of the source than nodes "farther" away. Nodes that are much farther away from the source will have so little information on the source that it will be indistinguishable from "noise". Figure 7 illustrates this principle: Node A is a 1-hop neighbor of Node B. Node B aggregates its information about all its neighbors and decays this information before sending it to node C. Node C does the same thing with all its neighbors and what results is less and less accurate information about any node in a network depending on the distance that node is from the source.

2.2.1 Exponential distance decay

Updates are easily created by aggregating the DBFs associated with each interface in the DRT. We follow much of the same aggregation techniques presented in [10] in decaying bits exponentially with number of hops. Exploration of various distance decay methods are beyond the scope of this paper.

Algorithm 1 DRT updates

1: // Create local DBF (given local ID x)
2: **for all** $i \in \{ 1, ..., k \}$ **do**
3: Set bits $A[h_i(x)]$ to 1.
4: **end for**
5: // Create Update (w/ decay function $f_b(n)$)
6: // Copy all the bits from the local DBF A into update U
7: $U \leftarrow A$;
8: // Decay info received from neighbors stored in DRT
9: **for all** $i \in$ Interface Direction **do**
10: **for all** $r \in \{1, ..., m\}$ **do**
11: **if** $A_i[r] == 1$ **then**
12: $U[r] \leftarrow 1$, with prob. $f_b(n+1)/f_b(n)$;
13: **end if**
14: **end for**
15: **end for**
16: Return U;

As algorithm 1 shows, the local node ID is first hashed into a DBF U. Then, each of the DBFs associated with each interface is bitwise decayed according the decaying function $f_b(r)$ and bitwise OR-ed with U. U is then compressed using bloom filter compression [11] and broadcasted out all interfaces to all neighbors.

Upon receipt of the aggregated and decayed DBF from a neighbor, a node will take the max number of bits counted

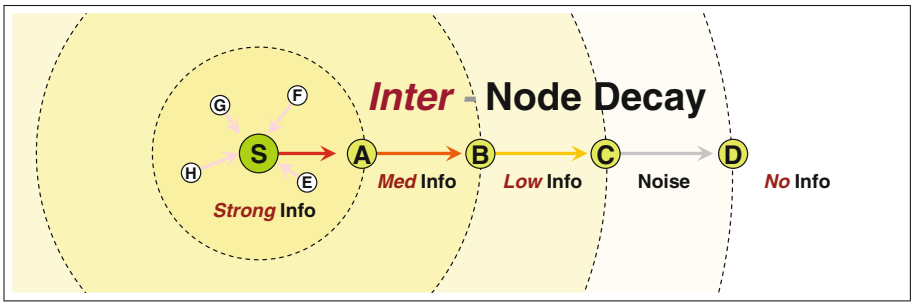


Fig. 7 Neighbor information is decayed going farther from the source

for each entry and bit-wise OR the received DBF with the DBF associated with the interface it receive the packet for the max number of bits for that entry. The reason we cannot simply bit-wise OR the entry with the received DBF is because with increased number of hash functions, probabilities will be biased toward directions with more neighbors since there is higher probability that even if neighbors have same amount of information about a specific node, the bits associated with that information will be more spread out.

Dissemination, which occurs periodically, only takes place between 1-hop neighbor nodes and requires no route/path maintenance. A common assumption in wireless routing protocols is neighbor discovery (each node knows its 1 hop neighbors) and this is usually achieved through periodically broadcasting *hello* packets to all nodes within transmission range. By piggy-backing dissemination information on these *hello* packets, we can therefore disseminate DRT information to our 1 hop neighbors without additional overhead.

2.3 Design variables and considerations

There are several factors to consider in designing routing algorithms based on DRTs. Table 1 lists several parameters that affect successful packet delivery using DRTs. Exploration of all the variables is beyond the scope of this paper, however, in Sect. 5, we examine how varying some of the constraints affect routing in MORRP.

3 Mobile orthogonal rendezvous routing protocol

MORRP relies heavily on DRTs to provide probabilistic routes from source to destination. Because information about nodes farther away tend not to need to be refreshed as often as nodes closer to a source [12], MORRP is broken into two major arenas of operation, each with a separate DRT updated at different intervals: *near field* and *far field*. The near field handles direction changes and information about 2–3 hop "neighbors" while the far field handles everything beyond the near field's "region of influence". Near field operation including information dissemination is fairly straight forward and follows what is described in Sect. 2.1. In this section, we will focus mainly on reaching nodes that are not in the immediate vicinity of the source (i.e. nodes in the far-field).

3.1 Assumptions

MORRP relaxes many of the assumptions made by position-based routing protocols (no need for location discovery and coordinate space embedding) while still providing connectivity even in *highly mobile* environments. To do so, MORRP assumes 4 givens:

Table 1 Parameters affecting successful packet delivery

Network density	Average number of neighbors
Num of interfaces (ϕ)	The number of interfacesper node
Time decay factor (D_t)	Fraction of bits in bloom filter dropped per second per time interval D_i)
Time decay interval (D_i)	The time interval to dodecaying
Dist. decay factor (D_d)	Fraction of bits in bloom filter to drop per hop
Near/far-field threshold	The number of bits found for it to be considered
(*thresh/ ff_thresh*)	A positive result in searching in NF/FF DRT
Spread ratio (s_ratio)	The ratio between bits used for spread decay and bits used for time decay
Bloom filter size (m)	The number of bits in each bloom filter
# of Hash funcs (k)	The number of hash functions

- *Neighbor to direction assignment*—Any given node will know (i) its 1-hop neighbors and (ii) the given direction/interface to send packets to reach this neighbor.
- *Local sense of direction*—Each node must have its own local perception of direction with antennas/transceivers oriented in such a way as to be able to consistently send out orthogonal directions. This can easily be done by selecting any of the transceivers as the "local North" and assigning angles to the others based on that selected transceiver. Nodes must also be capable of communicating directionally over their transceivers. This can be done by various hardware including directional and smart antennas [1], and FSO transceivers [4]. FSO transceivers are a particular interest due to their fine-grained transmit angle and ability for several dozen to be tesselated together oriented in several directions on a single node [4].
- *RREQ and RREP send/receive time is negligible*—We assume that the time required to send a RREQ and receive a RREP (if one is found) is negligible compared to node movement. In otherwords, if a path exists, a node receiving a RREQ should be able to simply record its "previous hop" and "source" so that RREP packets can retrace the route back to the source easily.
- *No rotations in nodes*—In all of our test cases, we assume that nodes do not rotate. If nodes rotate, additional algorithms and hardware (e.g. electronic compasses) are needed. Furthermore, DRT bits will need to be shifted to reflect the rotation. These considerations are therefore beyond the scope of this paper.

3.2 Near field operation

Nodes within two or three hops (depending on distance decay factor) of a specific source are considered "near-field" nodes because they have *some* information about the position of the source relative to itself. This information becomes less and less with increasing distance from the source. Near-field DRTs are maintained periodically as described in Sect. 2.1 and nodes close to a specific source should have adequate information about the position of a destination in the near-field even if they're not an immediate neighbor. Sending to a node in the near-field involves querying each entry in the DRT to return the number of bits in the DBF associated with a specific node ID. The node is said to "have information" about a specific node if the maximum returned bits is greater than a set threshold number of bits (*thresh*). The threshold number of bits is anywhere between 1 and k where k is the number of hash functions. A low threshold results in more false positives. The interface with the maximum number of bits associated with a destination node ID and above the threshold bits is then selected as the interface to send the packet and a random neighbor in that direction is chosen to be the forwarder. If there is a tie in the number of bits found for a specific node ID, one is randomly chosen. The process is repeated until the destination is reached.

Additionally, because one of the basic assumptions of MORRP is neighbor discovery in which each node knows its 1 hop neighbors and the interface associated with that interface, if a source wishes to send to its neighbor, it can do so by merely selecting the interface the neighbor resides in and send out that interface. Sending to nodes not within 1 hop from the source but within near-field operation requires querying the *near-field DRT* for a specific destination.

3.3 Far field operation

Because *near-field* DRTs are decayed between nodes at a substantial decay rate, in general, nodes past three hops from a specific source will have little to no information about the source. To forward packets to nodes where there is little to no information about position (Far-field operation), MORRP sends route request (RREQ) packets in orthogonal directions (randomly choosing a neighbor in each orthogonal direction) and when one of these RREQ packets intercepts the path of the destination's announcement packets (also sent in orthogonal directions at periodic intervals), a RREP packet is sent back to the source. MORRP stores only *weak-state* [9] at each hop and because of infrequent updates, the far-field DRT is decayed at a slower rate than the near-field DRT. The protocol itself consists of both a *proactive* and *reactive* element and the next sections will detail each element and explain the tradeoffs and design considerations associated with each part.

3.3.1 Proactive element

In order for a source and destination to agree upon a rendezvous node, pre-established "routes" from the rendezvous node to the destination must be in place. Because each node has merely a local sense of direction, making no assumption on position and orientation of other nodes in the network, it can only make forwarding decisions based on its own neighbor list. As mobility is increased however, routes become stale more quickly.

Upon a set interval, each node sends MORRP announcement packets to its neighbors in orthogonal directions. MORRP announcement packets are essentially "hello" packets that are only sent (and forwarded along) orthogonal directions and are dropped after a certain TTL. These packets are lightweight and contain nothing more than the

source node's ID. When those neighbors receive these MORRP announcement packets, it hashes the ID of the source of the packet into the far-field DRT entry corresponding to the interface/direction it received the announcement packet and stores/updates the shortest number of hops associated with this announcement sequence number to the announcement source in a "hop count" table if the sequence number of the packet is greater or the hop count is less than that recorded in the table (better or newer path). Note that this "hop count" table is not maintained in any traditional sense and only updated once we have routes. The packet is then forwarded out the interface exactly opposite in direction from the interface it received the packet. If no neighbor is found in the opposite interface to send the MORRP announcement, ORRP's multiplier angle method (MAM) is employed to attempt to maintain straight paths or forward along the perimeter as much as possible. Discussion of MAM is beyond the scope of this paper. Algorithms 2 and 3 detail the basic procedure for sending, forwarding, and receiving MORRP announcements.

Algorithm 2 Send/forward MORRP announcement

```
ForwardAnnouncementPacket(p)
// Check if we are the source - forward opposite if not
if p→ Src = ID then
    // We are the source, forward orthogonally
    // Get interface ID of local north
    j ← GetLocalNorthIntID
    α ← NumInterfaces
    // Send out orthogonal directions
    for i = 1, i ≤ 4, i++ do
        Φ ← GetRandomNeighbor(j)
        // Send to neighbor
        send(Φ)
        j ← ((j + α/4)%α)
    end for
else
    // We are forwarding - only forward opposite
    // Get received interface ID
    j ← (p → Recv_Int_Id)
    // Get opposite interface j ←((j + α/2)%α)
    Φ ← GetRandomNeighbor(j)
    // Send to Neighbor
    send(Φ)
end if
```

The entries in the far-field DRT are decayed in the same way as the near-field DRT with *intra-node* decay methods described in Sect. 2 used. In this way, even if nodes are moving, they can maintain a general sense of direction for any source they receive an announcement packet from. Time decaying methods ensure that positioning of nodes become less and less accurate with time and eventually, the information a specific node has about another node becomes negligible if not updated. Unlike the near-field DRT, however, far-field DRT is *not* shared with neighbors so *inter-node* decay is not used. This is to minimize indirection confusion.

Algorithm 3 Receive MORRP announcement

```
RecvAnnouncementPacket(p)
1: p_src ←(p → Src)
2: p_int ← (p → Recv_Int_Id)
3: dbf ← GetDBFfromFarFieldDRTInterface(p_int)
4: // Hash Announcement source (p_src) into Far-Field DRT associated
   with received interface
5: for all i ∈ { 1, ..., k } do
6:     Set bits dbf[h_i(p_src)] to 1.
7: end for
8: // Get entry from hop-count table, if missing, create one
9: hc ← GetHCEntry(p_src)
10: if hc = null then
11:     // There's no entry back to announcement source, create one
12:     hc ← CreateHCEntry(p_src)
13: end if
14: // Update hop count entry if its a new announcement or if hop
    count smaller
15: if (hc_seqnum < p_seqnum) OR (hc_seqnum = p_seqnum AND
    hc_hops < p_hops)then
16:     hc ← UpdateHCEntry(p)
17: end if
18: if p_hops ≥TTL then
19:     drop(p)
20: else
21:     ForwardAnnouncementPacket(p)
22: end if
```

3.3.2 Reactive element

In order to build the path from source to rendezvous node, an on-demand, reactive element to MORRP is necessary. When a node wishes to send packets to an destination that is not within its immediate neighbor table or near-field DRT, it creates an entry in a simple *destination-rendezvous node* table and sends out a route request packet (RREQ) in all four of its orthogonal directions. Due to the fact that far-field DRTs only track nodes that send MORRP announcements or RREQ packets along the line, the *destination-rendezvous* table keeps track of which rendezvous nodes to forward to for a specific destination. Until a RREP

is found, this entry is considered unusable. Algorithm 4 outlines how MORRP RREQ packets are sent and forwarded.

When a neighbor node receives this RREQ packet, it hashes the node ID of the source into its far-field DRT and forwards the packet in the opposite direction utilizing MAM. Because one of the assumptions we made is that RREQ and RREP send and receive times are negligible compared to node movement, we need to add a short-timeout reverse path to the source so RREP packets can be sent back quickly. A simple *destination-nexthop* routing table with fast entry expiry times is used for this reverse-route back to the source. Algorithm 5 shows how MORRP RREQ packets are processed upon receipt.

Algorithm 4 Send/forward MORRP route request

```
ForwardRREQPacket(p)
// Check if we are the source - forward opposite if not
if p→ Src = ID then
    // We are the source, forward orthogonally
    // Get interface ID of local north
    j ← GetLocalNorthIntID
    α ← NumInterfaces
    // Send out orthogonal directions
    for i = 1, i ≤ 4, i + + do
        Φ ← GetRandomNeighbor(j)
        // Send to neighbor
        send(Φ)
        j ← ((j + α/4)%α)
    end for
    // Create an entry in the Destination-Rendezvous table
    dre ← CreateDREntry(p_src)
else
    // We are forwarding - only forward opposite
    // Get received interface ID
    j ← (p → Recv_Int_Id)
    // Get opposite interface j ←((j + α/2)%α)
    Φ ← GetRandomNeighbor(j)
    // Send to Neighbor
    send(Φ)
end if
```

In a 2-D Euclidian plane, by sending a RREQ packet in all 4 of its orthogonal directions, it is highly likely to encounter a node that has a path to the destination. When a node with a path to the destination (destination is either in neighbor table or destination ID is above threshold in near or far-field DRTs) receives the RREQ, it sends a RREP packet back the way the RREQ came. Because each node along the path stored a reverse route to the source and we assume that nodes have not moved much in the process of the RREQ being sent, it is able to forward the RREP back efficiently. Finally, when the source receives the RREP, it hashes the rendezvous node's ID into its far-field DRT and updates the *destination-rendezvous* table with the rendezvous node for a specific destination and "activates" that entry.

Algorithms 7 and 8 detail the send, forward, and receive process for MORRP RREP packets.

3.3.3 Data delivery

For data delivery, if the packet is at the source, first the neighbor list and near-field DRT is queried for the destination. If destination is not found in these two tables, then the far-field DRT is checked to see if the number of bits associated with the destination hash is above the threshold. If destination is still not found in the far-field DRT, then the destination-rendezvous table is queried to see if there is a rendezvous node we need to send to. If it is found, then the far-field DRT is queried for the rendezvous node ID. If after all these steps the destination is unreachable, then a RREQ is sent out in orthogonal directions.

Algorithm 5 Receive MORRP route request

```
RecvRREQPacket(p)
p_src ←(p → Src)
p_search_id ← (p → Search_ID)
p_int ← (p → Recv_Int_Id)
dbf ← GetDBFfromFarFieldDRTInterface(p_int)
// Hash RREQ source (p_src) into Far-Field DRT associated with received interface
for all i ∈ { 1, ..., k } do
    Set bits dbf[h_i(p_src)] to 1.
end for
// Create an entry for the reverse route for RREP
rt ← GetRTEntry(p_src)
if rt = null then
    // There's no entry back to RREQ source, create one
    rt ← CreateRTEntry(p_src)
end if
// Update reverser route entry if its a new RREQ or if hop count smaller
if (rt_seqnum < p_seqnum) OR (rt_seqnum = p_seqnum AND rt_hops < p_hops) then
    rt ← UpdateRTEntry(p)
end if
// Get entry from hop-count table, if missing, create one
hc ← GetHCEntry(p_src)
if hc = null then
    // There's no entry back to RREQ source, create one
```

Table e continued

Algorithm 5 Receive MORRP route request

```
23:     hc ← CreateHCEntry(p_src)
24: end if
25: // Update hop count entry if its a new RREQ or if hop count smaller
26: if (hc_seqnum < p_seqnum) OR (hc_seqnum = p_seqnum AND hc_hops < p_hops) then
27:     hc ← UpdateHCEntry(p)
28: end if
29: // Look to see if we can send out a Route Reply
30: RREQCheckRoute(p)
```

Algorithm 6 Search destination route check

```
RREQCheckRoute(p)
1: if p_search_id = ID then
2:     // I'm what the source is looking for
3:     num_hops ← 0
4:     SendRREPPacket(p, num_hops)
5: else if p_search_id ∈ NeighborList then
6:     // RREQ search ID is my 1 hop neighbor:
7:     num_hops ← 1
8:     SendRREPPacket(p, num_hops)
9: else if p_search_id ∈ NearFieldDRT then
10:    // RREQ search ID is my Near Field DRT:
11:    num_hops ← 2
12:    SendRREPPacket(p, num_hops)
13: else if p_search_id ∈ FarFieldDRT then
14:    // RREQ search ID is my Far Field DRT:
15:    hc ← GetHCEntry(p_src)
16:    SendRREPPacket(p, hc_hops)
17: else
18:    if p_hops ≥ TTL then
19:        drop(p)
20:    else
21:        ForwardRREQPacket(p)
22:    end if
23: end if
```

Algorithm 7 Send/forward MORRP route reply

```
ForwardRREPPacket(p)
1: // Search for path back to RREQ source
2: rt ← GetRTEntry(p_dest)
3: // Send to next hop
4: send(rt_nexthop)
```

Algorithm 8 Receive MORRP route reply

```
1: // Take care of reverse route to rendezvous node
2: p_src ←(p → Src)
3: p_search_id ← (p → Search_ID)
4: p_int ← (p → Recv_Int_Id)
5: dbf ← GetDBFfromFarFieldDRTInterface(p_int)
6: // Hash RREQ source (p_src) into Far-Field DRT associated with received interface
7: for all i ∈ { 1, ..., k } do
8:     Set bits dbf[h_i(p_src)] to 1.
9: end for
10: // Create an entry for the reverse route to rendezvous node
11: rt ← GetRTEntry(p_src)
12: if rt = null then
13:     // There's no entry back to RREQ source, create one
14:     rt ← CreateRTEntry(p_src)
15: end if
16: // Update reverse route entry if its a new RREP or if hop count smaller
17: if (rt_seqnum < p_seqnum) OR (rt_seqnum = p_seqnum AND rt_hops < p_hops) then
18:     rt ← UpdateRTEntry(p)
19: end if
20: // Get entry from hop-count table, if missing, create one
21: hc ← GetHCEntry(p_src)
22: if hc = null then
23:     // There's no entry back to RREQ source, create one
24:     hc ← CreateHCEntry(p_src)
25: end if
26: // Update hop count entry if its a new RREQ or if hop count smaller
27: if (hc_seqnum < p_seqnum) OR (hc_seqnum = p_seqnum AND hc_hops < p_hops) then
28:     hc ← UpdateHCEntry(p)
29: end if
30: // Process RREP Packet
31: if p_dest = ID then
32:     // We are the source of the RREQ/Dest of RREP
33:     // Find rendezvous table entry
34:     rne ← GetRTEntry(p_src)
35:     rne_rend_node ← p_src
36:     // Send all buffered packets for this destination
37:     SendBufferedData()
38: else
39:     // We are NOT the source of the RREQ/Dest of RREP
40:     ForwardRREPPacket(p)
41: end if
42: SendBufferedData()
```

For forwarding packets, a similar approach is taken in that first the neighbor list and near-field DRT is checked for the rendezvous node if its present in the packet header and if not, the destination node. If it is not found in either, the far-field DRT is checked. If it is not found in any of the tables, the packet is simply forwarded to the opposite direction of receipt (the antenna exactly 180° from the receiving antenna). Algorithms 9–12 depict this process.

Algorithm 9 MORRP data delivery

```
ForwardData(p)
1: if p_dest ∈ NeighborList then
2:     // Destination is my 1 hop neighbor:
3:     SendData(p, next_hop)
4: else if p_dest ∈ NearFieldDRT then
5:     // Destination is my Near Field DRT:
6:     j ← GetInterfaceIDfromNFDRT()
7:     Φ ← GetRandomNeighbor(j)
8:     p_rend_node ← null
9:     SendData(p, Φ)
10: else
11:     if p_src = ID then
12:         // I'm the source
13:         ForwardDataSrc(p)
14:     else
15:         // I'm just a forwarder
16:         if p_rend_node≠NULL then
17:             // data packet has rendezvous node state set in packet
    header
18:             ForwardDataWithRendezvous(p)
19:         else
20:             // data packet has no rendezvous node
21:             ForwardDataNoRendezvous(p)
22:         end if
23:     end if
24: end if
```

4 Numerical analysis

In ORRP, a path is established when a RREQ and announcement packet intersect at a rendezvous node. The probability of intersection depends on a point and determines reachability. With MORRP, because nodes are constantly moving, the probability that a RREQ will intercept a node that originally contained announcement information about a destination becomes increasingly slim with time. We say that the information is "diffused" over space with time. It is therefore interesting to gain insight on the probability of even finding a rendezvous node in a mobile environment by sending out RREQ and announcement packets in orthogonal directions.

Figure 9 gives an illustration of our analysis. Details are left out due to space constraints. In short, assuming a source S wanting to send to a destination D, if the transmission radiuses of the nodes are the green/smaller bands, we say that with a set mobility speed, the maximum an announcement packet path can deviate from the line is represented by the grey/larger bands. The intersection formed by the smaller green bands (area B) represent the area of nodes that would have have received the RREQ *and* the announcement packets. Additionally, the intersection formed by the larger grey band and green band originating from the source represents the area of nodes where nodes originally along the announcement path would have traveled (area A).

Algorithm 10 Data delivery—packet source

```
ForwardDataSrc(p)
1: if p_dest ∈ FarFieldDRT then
2:     // Destination is my Far Field DRT:
3:     j ← GetInterfaceIDfromFFDRT()
4:     Φ ← GetRandomNeighbor(j)
5:     p_rend_node ← null
6:     SendData(p, Φ)
7: else if p_dest ∋ DestRendTable then
8:     // Destination not in Dest-Rendezvous Table
9:     BufferData(p)
10:     ForwardRREQPacket(p)
11: else
12:     dre_rend_node ← GetRendNode(p_dest)
13:     if dre_rend_node ≠ null then
14:         if dre_rend_node ∈ NeighborList then
15:             // Rendezvous node is my 1 hop neighbor:
16:             SendData(p, dre_rend_node)
17:         else if dre_rend_node ∈ NearFieldDRT then
18:             // Rendezvous node is my Near Field DRT:
19:             j ← GetInterfaceIDfromNFDRT()
20:             Φ ← GetRandomNeighbor(j)
21:             p_rend_node ← dre_rend_node
22:             SendData(p, Φ)
23:         else if dre_rend_node ∈ FarFieldDRT then
24:             // Rendezvous node is my Far Field DRT:
25:             j ← GetInterfaceIDfromFFDRT()
26:             Φ ← GetRandomNeighbor(j)
27:             p_rend_node ← dre_rend_node
28:             SendData(p, Φ)
29:         else
30:             // Stale route
31:             BufferData(p)
```

Table j continued

Algorithm 10 Data delivery—packet source

```
32:          ForwardRREQPacket(p)
33:       end if
34:    else
35:       // Destination and Rendezvous definitely not known
36:       BufferData(p)
37:       ForwardRREQPacket(p)
38:    end if
39: end if
```

Using Matlab, we iterate through all possible nodes in a network and all possible source and destination orientations and generate the intersection parallelograms. Then, we count the number of nodes within area A and divide by area B to get the probability that sending a RREQ will intercept an announcement given a set mobility speed, transmission radius, and time after announcement sent. In our numerical analysis, we tried to mimic our NS simulations so we normalized the 250 m transmission radius to 1m and corresponding mobility speeds. For our paper, we only considered square topologies.

Table 2 shows our results for probability of announcement/RREQ rendezvous for various mobility speeds after waiting 1 and 4 s after announcement packets were sent. As expected, the results showed decreasing, yet high, intersect probability with higher mobility and longer wait time. This is because information becomes more dispersed over time and higher mobility. Our analysis gives only a partial view of reach probability as actual data will still need to hit the rendezvous region and destination region for successful packet delivery in mobile environments. And although not complete in describing the whole protocol, it gives a high-order view of the overall intersect behavior and shows that even with high mobility, the probability of finding a rendezvous point is relatively high. In the actual protocol, not all nodes require this far-field operation because some are close enough to the source to utilize the near-field DRT. Additionally, RREQs are sent upon need and can be anywhere between the announcement interval and node mobility velocity is not constant throughout the network. All these factors merit additional simulations to fully understand the inner-workings of the protocol which we describe in the following section.

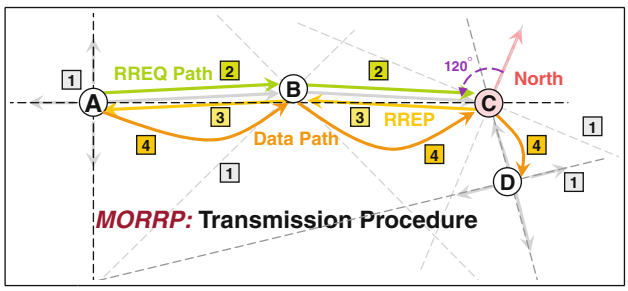


Fig. 8 1: MORRP Announcements used to generate rendezvous node-to-destination paths 2–3: MORRP RREQ and RREP Packets to generate source-to-rendezvous node paths 4: Data path after route generation

Algorithm 11 Data delivery—forward with rendezvous node

```
ForwardDataWithRendezvous(p)
1: if p_rend_node = ID then
2:    // We are the rendezvous node. Check if dest is in far field DRT
3:    // (Whether dest is in neighbor list and near field DRT already checked)
4:    p_rend_node ← null
5:    if p_dest ∈ FarFieldDRT then
6:       j ← GetInterfaceIDfromFFDRT()
7:       Φ ← GetRandomNeighbor(j)
8:       SendData(p, Φ)
9:    end if
10: else
11:    if p_dest ∈ FarFieldDRT then
12:       // Destination in far field DRT
13:       j ← GetInterfaceIDfromFFDRT()
14:       Φ ← GetRandomNeighbor(j)
15:       SendData(p, Φ)
16:    else if p_rend_node ∈ NeighborList then
17:       // Rendezvous node is my 1 hop neighbor:
18:       SendData(p, next_hop)
19:    else if p_rend_node ∈ NearFieldDRT then
20:       // Rendezvous node is my Near Field DRT:
21:       j ← GetInterfaceIDfromNFDRT()
22:       Φ ← GetRandomNeighbor(j)
23:       SendData(p, Φ)
24:    else if p_rend_node ∈ FarFieldDRT then
25:       // Destination is my Far Field DRT:
26:       j ← GetInterfaceIDfromFFDRT()
27:       Φ ← GetRandomNeighbor(j)
28:       SendData(p, Φ)
29:    else
30:       // Just keep forward in opposite direction
31:       α ← NumInterfaces
32:       j ← (p → Recv_Int_Id)   j ← ((j + α/2)%α)
33:       Φ ← GetRandomNeighbor(j)
34:       SendData(p, Φ)
35:    end if
36: end if
```

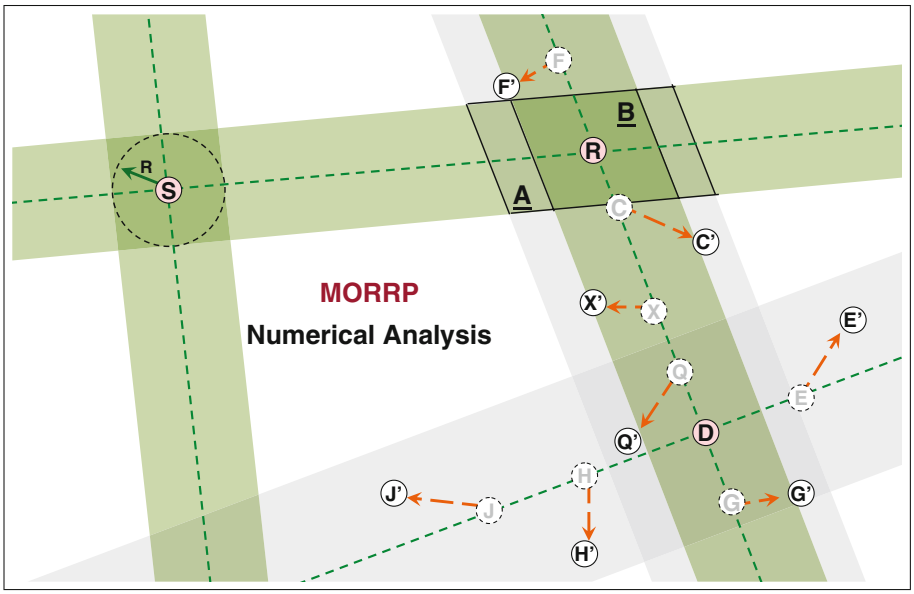


Fig. 9 MORRP reachability numerical analysis calculation

Table 2 Comparison of Probability of Rendezvous vs. Velocity

Mobility speeds	10 m/s (%)	20 m/s (%)	30 m/s (%)
After 1 s	98.5	96.9	94.3
After 4 s	91.9	81.9	74.6

Algorithm 12 Data delivery—forward without rendezvous node

```
ForwardDataNoRendezvous(p)
1: // (Whether dest is in neighbor list and near field DRT already checked)
2: if p_dest ∈ FarFieldDRT then
3:     // Destination in far field DRT
4:     j ← GetInterfaceIDfromFFDRT()
5:     Φ ← GetRandomNeighbor(j)
6:     SendData(p, Φ)
7: else
8:     // Just keep forward in opposite direction
9:     α ← NumInterfaces
10:    j ← (p → Recv_Int_Id)   j ← ((j + α/2)%α)
11:    Φ ← GetRandomNeighbor(j)
12:    SendData(p, Φ)
13: end if
```

5 Performance evaluation

In this section, we provide performance evaluations of MORRP under various parameters and against several proactive, reactive, and position-based routing protocols with one omni-directional interface and several directional interfaces. The simulations were performed using Network Simulator [13], with nodes using the standard IEEE 802.11 MAC with the antenna range set to 250 m (NS2 default). Each node moves using the random waypoint mobility model with a node pause time of 5 s in a 1,300 m × 1,300 m area.

The performance metrics we evaluated are *packet delivery ratio*, *control packet overhead*, *average path length*, *aggregate network goodput*, *end to end latency*, and *far-field vs. near-field DRT usage*. We examine these metrics under conditions of varying *node mobility speeds*, *decay factors*, *transmission rates*, and *network densities*. All simulations were averaged over 3 runs of 5 different random topologies (total 15 trials). Table 3 outlines our default simulation parameters.

MORRP and ORRP were configured using n interfaces (divisible by 4) with each interface having a beam-width of $360/n°$. The interfaces/antennas were modeled simply as a sector antenna with each of the n interfaces oriented in such a way to provide omnidirectional circular coverage with no overlap (no side lobes or interference). Free-space propagation model was used unless otherwise noted. Some tests were performed later with two-ray ground propagation and the results were not much different. Announcement packets were sent every 4 s and announcement and RREQ packet TTLs were set to 10 hops. We choose 30 hash functions and a bloom filter size of 16000 bits for simulations with MORRP to ensure minimum overlap of bits with 100 or so nodes and employ no bloom filter compression. The exploration of optimal hash function sizes to ensure minimal bit collisions are beyond the scope of the paper and more information can be found in [11].

For *reactive* routing protocols like DSR and AODV which require no periodic updates, the standard NS2

Table 3 Default Simulation Parameters

Parameter	Values
Trans. radius/# interfaces	250 m/eight directional interfaces
Topology boundaries	1,300 m × 1,300 m
# of nodes/simulation time	100/170s
Announcement interval	4.0s
Mobility (m/s)	RWP Model between 0 and 30 m/s
Distance decay factor (D_d)	0.7 (fraction of bits dropped per hop)
Time decay factor (D_t)	0.3 (fraction of bits dropped per sec)
Time decay interval (D_i)	0.5s
# of BF hash funcs/BF size	30/16,000 bits
NF threshold/FF threshold	6/6 bits
Spread decay ratio (s_{ratio})	0.5

defaults were used. GPSR with GLS as the location service utilized defaults as well. For OLSR, topology control update interval set to 4 s to match ORRP and MORRP announcement intervals. For all simulations, a hello interval of 2 s was used and MAC layer feedback employed for all the routing protocols. A potential future extension is MORRP with routing metrics and link layer feedback. Traffic patterns varied for each test and are described in each subsection. Implementations and defaults for GPSR/GLS and OLSR can be found at [14] and [15] respectively.

In order to explore whether MORRP and ORRP gains were merely from capacity gains with directional antennas and multiple interfaces or *actual* design improvements, we modified AODV and OLSR implementations to support multiple directional interfaces in the same way as MORRP and ORRP. Since AODV and OLSR rely on omni-directional broadcast to disseminate information, by sending out all interface directions, one can simulate the behavior of AODV and OLSR broadcasts. Transmitting data packets, however, require only one interface to be active at a time and thus frees the medium and other interfaces for other nodes to use.

5.1 Evaluation of MORRP parameters in mobile environments

5.1.1 Effect of time and distance decay factors

Sections 2.1 and 2.2 point out that knowing how many bits of the bloom filter to "decay" (i.e. drop) per time interval and per hop will in many ways determine the key metric in mobile environments: reachability. In this section we evaluate how the *time* and *distance* decay factors (what fraction of bits are dropped) affect reachability and far field DRT usage. The smaller the decay factor, the *less* fraction of the bits are decayed per time interval for *time decay* and per hop for *distance decay*. In our simulations, we fixed the default values given in Table 3 while varying the decay factors from 0.1 to 1.0. Figures 10 and 11 show our results for 100 nodes (average of 10 1-hop neighbors per node) network density.

As expected, for various degrees of mobility, decreasing the time decay factor (dropping *less* bits per time interval) results in lower reach probability due to misinformation and bit accumulation. On the opposite spectrum, having too high of a time decay factor, thereby dropping a high number of bits per time interval also leads to less reach probability. This can be explained by the *far-field* DRT usage graph. As the time decay factor is low, the majority of data packets will be utilizing the far-field DRT to find a path because the near-field DRT coverage region will decay rather quickly. Because far-field DRT information is updated *less* frequently, too much reliance on it can yield inaccurate results. An optimal decay factor must be selected, therefore, to ensure high delivery success and a fair usage of both far-field and near-field DRTs.

Our results in Fig. 10 show that there is a gradual increase in reach probability when the distance decay factor goes from 0.1 to 0.6 and then plateau's out. The low reach probability when the distance decay is lower results

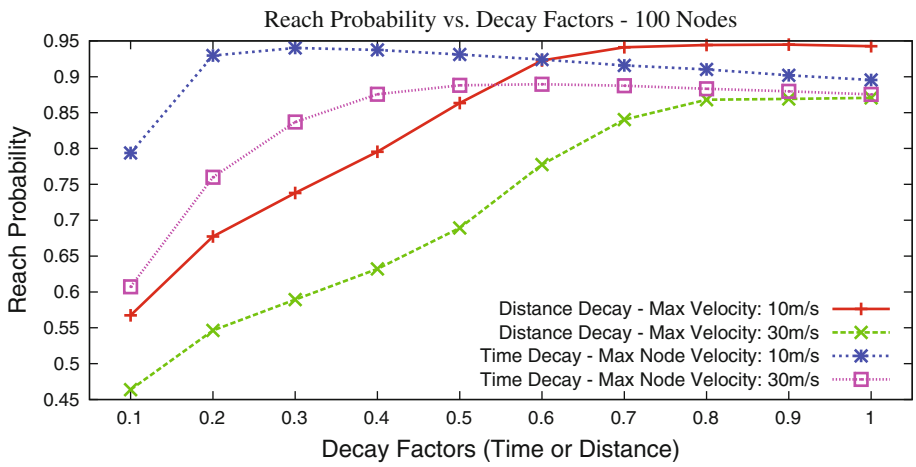


Fig. 10 As time decay factor increases, reach probability drops due to confusing paths as old information is not decayed adequately. As distance decay factor increases (more bits dropped per hop), a plateau in reach occurs

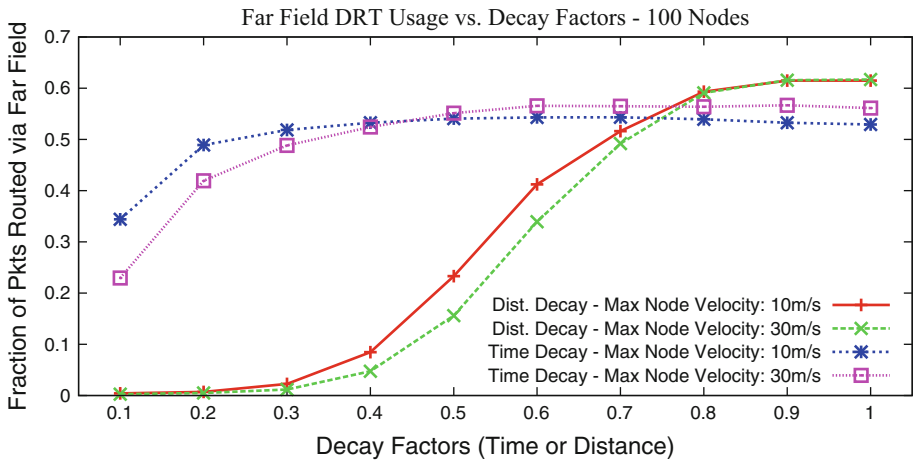


Fig. 11 Increased dependence on Far-Field DRT happens when more bits are dropped every time interval and distance

from saturation of bits to multiple interfaces resulting in confusing paths chosen. While the reach probability plateau's at a distance decay factor of 0.6, the far-field dependence graph in Fig. 11 shows that there is still a gradual shift from using near-field DRT to route information at the source to far-field DRT dependence.

5.1.2 Effect of threshold

Another interesting knob to adjust is the near-field and far-field bit threshold. As the threshold increases, the amount of information a node has about neighbors farther away decreases. It is expected that as the threshold for the near-field increases to greater than or equal to the number of hash functions, each node will only have information about itself. From our results, it was shown that when the threshold is low, there is a lot more confusion about path selection because of bit collisions and slow time decay. This results in poor path choices, low reachability, and greater dependence on the far-field DRT. As threshold increases, it approaches a point where each node has information about itself and its 1 hop neighbors (since 1 hop neighbors do not decay their node own ID hash when sending their DRT) resulting in high reachability. Furthermore, it was seen that reach probability peaks at roughly 6 bits which is roughly 20% of the number of hash functions used.

5.2 Comparison of MORRP against AODV, GPSR/GLS, OLSR and ORRP

In this subsection, we evaluate MORRP against reactive protocols like AODV [16], proactive protocols like OLSR [17], and position-based protocols like GPSR/GLS [18, 19] in terms of reach probability, average path length, control overhead, delivery success, aggregate network goodput, and end-to-end delay under conditions of varying mobility speeds, data rates, and network densities. Table 4 shows a comparison of the classification of ad-hoc routing protocols measured, highlighting the need for node localization and key differences. Default parameters for MORRP given in Table 3 were used in all scenarios unless otherwise stated. We focus heavily on *reachability/delivery success* in all these scenarios because in mobile adhoc networks, reachability comes primary over throughput, latency, etc. The reason is because our results show that for high mobility, even limited-flooding protocols like AODV and OLSR simply cannot deliver the majority of the packets (low reachability).

5.2.1 Effect of increased velocity

In this subsection, we evaluate the effect of increasing velocity on traditional routing protocols like AODV, GPSR/GLS, and OLSR and compare it to MORRP, ORRP, and multi-interfaced versions of AODV and OLSR. Our initial simulations involve relatively light load (1000 random 5 connections). While protocols like GPSR/GLS provide high reach under light load, as the load increases to 10,000 connections, we see a significant drop in reachability. Figure 5.2.1 shows our results in comparing MORRP to traditional routing protocols with one omni-directional antenna under varying node and topology sizes.

It is clear that in conditions of high mobility with few connections, MORRP with atleast 8 interfaces provides high reach probability (93% for $1,300 \times 1,300\ m^2$

Table 4 Comparison of Adhoc Routing Protocols

Type	Requires localization	No localization
Proactive	–	OLSR, ORRP, MORRP
Reactive	GPSR/GLS	DSR, AODV, ORRP, MORRP

networks and 87% for $2000 \times 2,000\ m^2$ networks) even under conditions of infrequent announcements sent (4 intervals). As maximum velocity increases, AODV and OLSR fail because of stale routes. With high mobility, it becomes increasingly hard to maintain end-to-end routes without increasing state dissemination rate or route requests. Both options lead to network congestion.

GPSR with GLS performs rather well in all cases, providing high reach even with increased mobility. Simply using GPSR with GLS (or a version modified to support directional antennas), however, runs into several issues: (1) End-to-end packet latency with GPSR with GLS is fairly bad, reaching close to 3–4 s per packet (See Fig. 5.2.1) and (2) GPSR/GLS requires node localization which we currently assume is a given. Position information is often obtained through node localization protocols which incur additional overheads and function poorly in sparse network environments or additional hardware like GPS which require "sky" access.

To see where the multiple interfaces shine, we increase the load from 1,000 to 10,000 connections, each for 5 s. As Fig. 5.2.1 shows, under increased load, protocols that utilize omnidirectional antennas saturate the medium and fail to successfully deliver packets. It becomes important, therefore, to understand how much of the gains from Fig. 5.2.1 results from more efficient medium reuse due to directional interfaces vs. the gains coming from MORRP protocol design itself. We compare modified versions of AODV and OLSR to support multiple directional antennas with ORRP and MORRP with the results shown in Fig. 5.2.1. The modified versions of AODV and OLSR still broadcast (ie: send out all interfaces) when performing route requests or dissemination due to the protocol design and as such, we expect to see better performance with MORRP.

AODV with eight directional interfaces shows significant improvement in delivery success vs. the traditional AODV due to the directional interfaces causing less interference in data delivery. The primary gains come from utilizing multiple interfaces as AODV is a reactive protocol and sends out route request packets on-demand. OLSR, on the other hand, saw only gains vs. the single interface OLSR only under low mobility. When the mobility increases to 30 m/s, these gains from capacity almost fade. This is due to OLSR's proactive nature. The periodic rate of link-state exchange simply cannot keep up with mobility speed and as a result, most packets are not successfully transmitted due to stale information rather than medium saturation. ORRP delivery success drops with increased mobility because it cannot maintain straight line next hop paths without constant updates. MORRP performs consistently well, delivering over 93% of the packets even in highly mobile environments.

5.2.2 *Effect of increased network density*

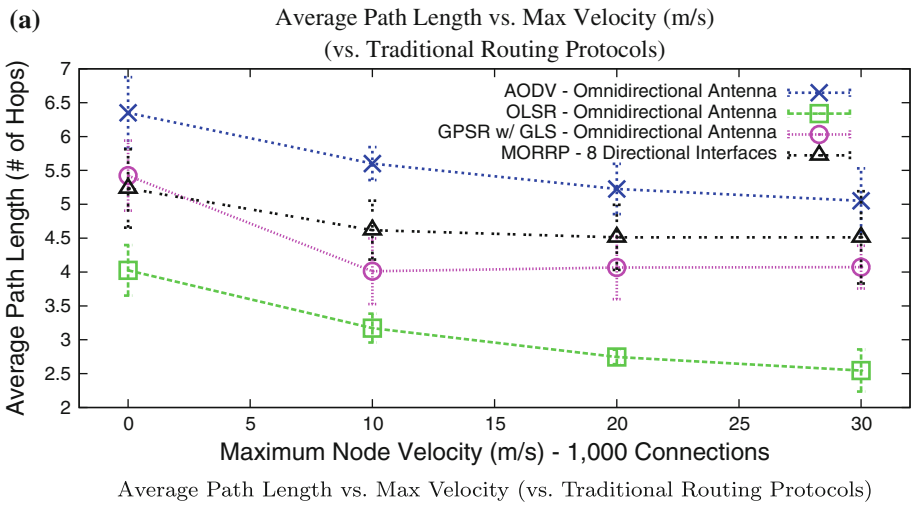
In this subsection, we evaluate how increasing network density affects *reachability* and *amount of control packet bytes* networkwide. The reason why we focus on control packet bytes rather than control packets is simply because MORRP sends bloom filters to its immediate neighbors. Although bloom filters are relatively small in size, the incurred overhead is larger than traditional packets. Figures 14 and 15 shows our results varying number of nodes from 50 to 300 with each node having a maximum velocity of 30 m/s. 2,500 random source and destination pairs are chosen and 512KB CBR packets sent for 20 s at a rate of 2 Kbps. For fair comparison, we only evaluate MORRP against ORRP and the modified versions of AODV and OLSR to support multiple directional interfaces.

It can be seen that as the number of nodes increases, AODV with multiple interfaces start dropping in reach due to its broadcast nature. OLSR fails because of stale routes due to high mobility. As the density increases, however, OLSR performs seemingly better because closer nodes are more within better reach. ORRP fails to deliver packets because in highly mobile environments, straight line paths are hard to maintain. MORRP delivers roughly 90% of the packets successfully. It seems odd that Fig. 14 shows that increasing network density doesn't affect delivery success in a significant way. The reason is that with a transmission radius of 250 m and a 1,300 m $\times$ 1,300 m topology, the average number of neighbors for a 300 node scenario is about 34. Coupled with the fact that each node has 8 interfaces, it averages out to each interface transmitting to about 4 neighbors. This kind of load is fairly light.

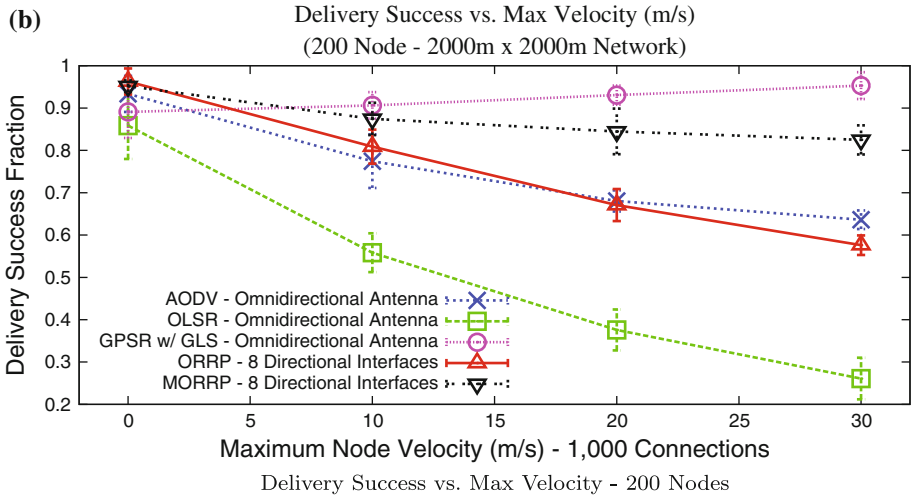
It is interesting to note that MORRP seems to send out *less* control packets than ORRP despite it needing to periodically send DRT update messages to all neighbors. The reason for this is simple: In ORRP, RREQ packets travel in a line and a RREP is generated only when this packet intersects with a path generated by an ORRP announcement packet. With MORRP, however, RREQ packets stop being forwarded once it intersects with a destination's "field". Because these "fields" are two or three hops large, MORRP RREQ packets traverse less hops than ORRP RREQ packets. OLSR grows rapidly with network size because more nodes are periodically sending out link-state information. AODV grows despite the constant number of connections due to more nodes in the network forwarding RREQ and RREP packets.

5.2.3 *Effect of increased data rate*

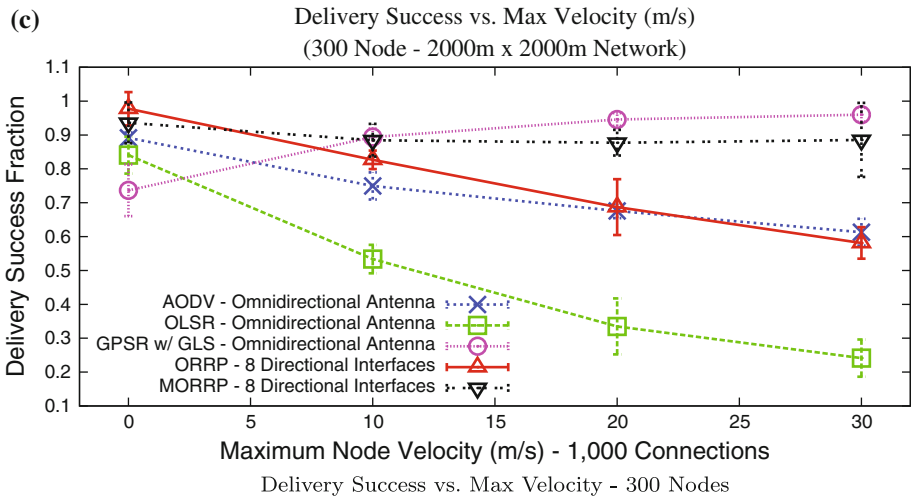
Although in mobile environments, high reachability naturally leads to high aggregate network goodput, it is important to quantify these gains. In this subsection, we



Average Path Length vs. Max Velocity (vs. Traditional Routing Protocols)



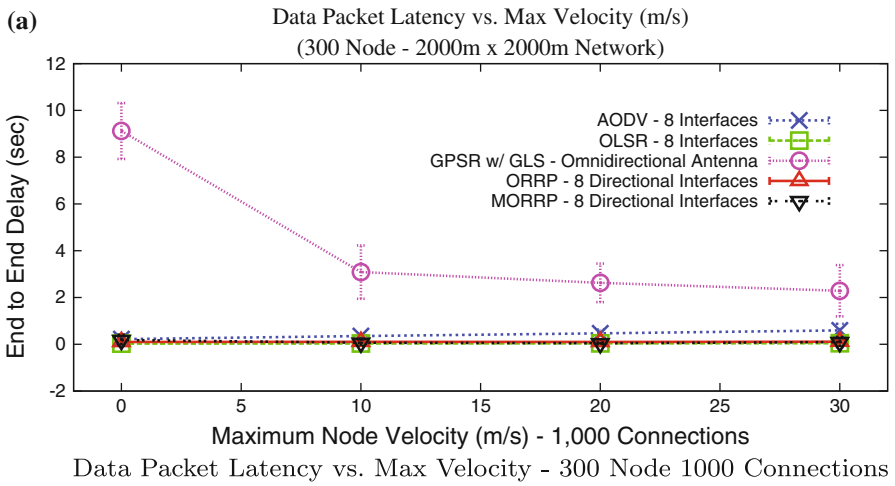
Delivery Success vs. Max Velocity - 200 Nodes



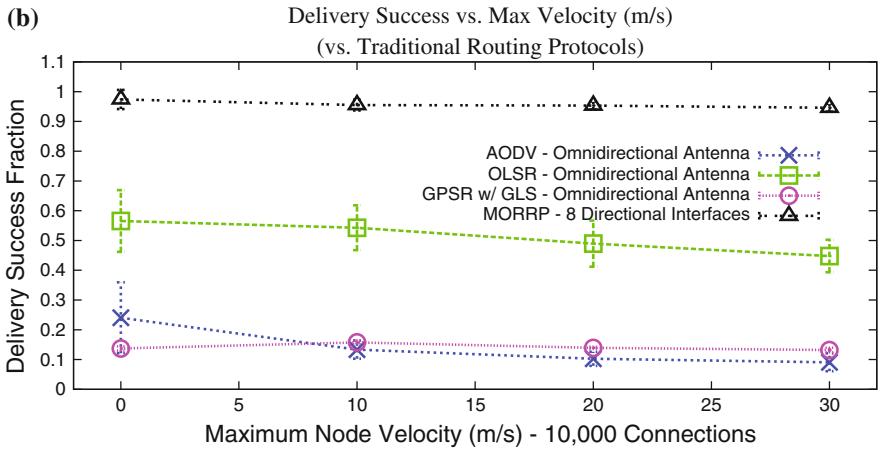
Delivery Success vs. Max Velocity - 300 Nodes

Fig. 12 MORRP yields above 93% reachability (data delivery success) even in highly mobile environments for medium-sized networks (1,300 m × 1,300 m) and about 87% reachability for large-sized networks (2,000 m × 2,000 m) with moderate density

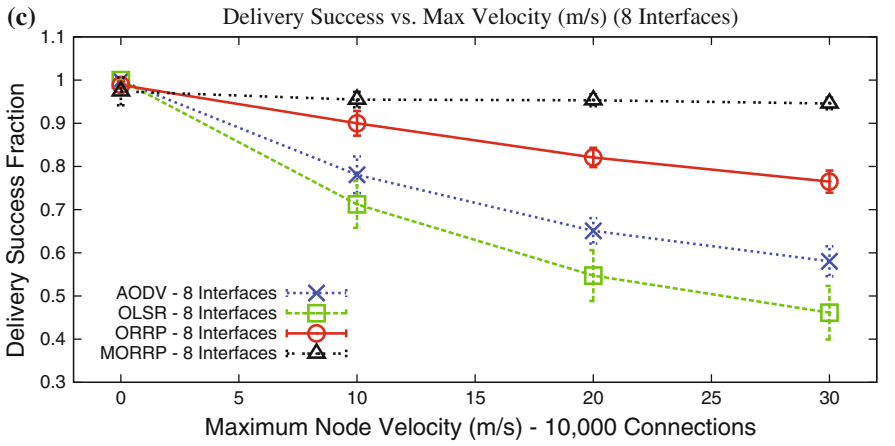
evaluate the effect of increased data rate on network goodput. To do so, we make all-to-all connections simultaneously network-wide and send packets at a set data rate for 20 s. By slowly increasing the rate, we can measure the amount of data that actually gets sent. We expect the capacity constraints will be mostly dependent on medium usage. All nodes are moving at a uniformly distributed velocity with a max of 30 m/s.



Data Packet Latency vs. Max Velocity - 300 Node 1000 Connections



Delivery Success vs. Max Velocity (vs. Traditional Protocols) - 10,000 Connections



Delivery Success vs. Max Velocity - 10,000 Connections

Fig. 13 GPSR w/ GLS has much higher end-to-end latency than all the other routing protocols including MORRP. Under load, all other routing protocols with omnidirectional antennas fail. MORRP performs much better than traditional routing protocols modified with directional antennas

We first compare MORRP to AODV, OLSR, and GPSR/GLS to highlight the gains from simply moving from omnidirectional antennas to directional antennas. Figure 16 shows our results. As expected, MORRP with 8 interfaces achieves much higher goodput than all the other protocols (roughly 10–14 × more than OLSR the closest competitor).

Comparing to AODV and OLSR with eight directional interfaces and ORRP, MORRP still performs almost 15–20% better than OLSR and ORRP. Again, ORRP fails

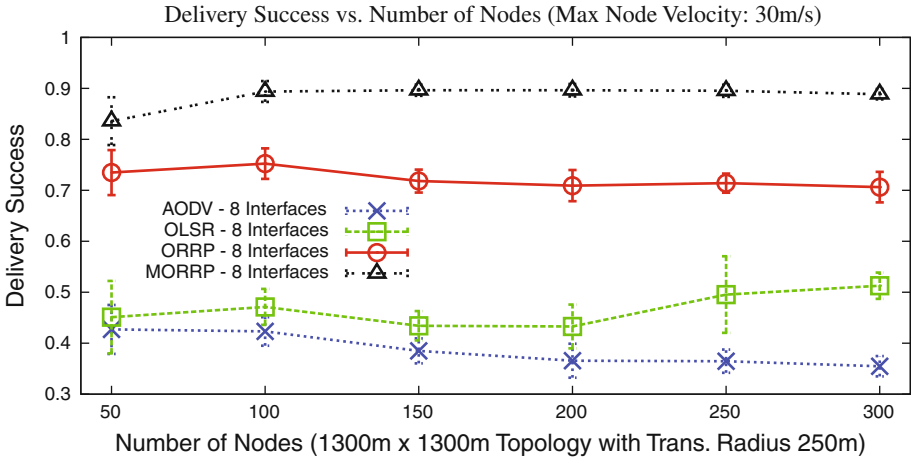


Fig. 14 MORRP scales well under increasing node density, providing over 90% delivery success in most cases

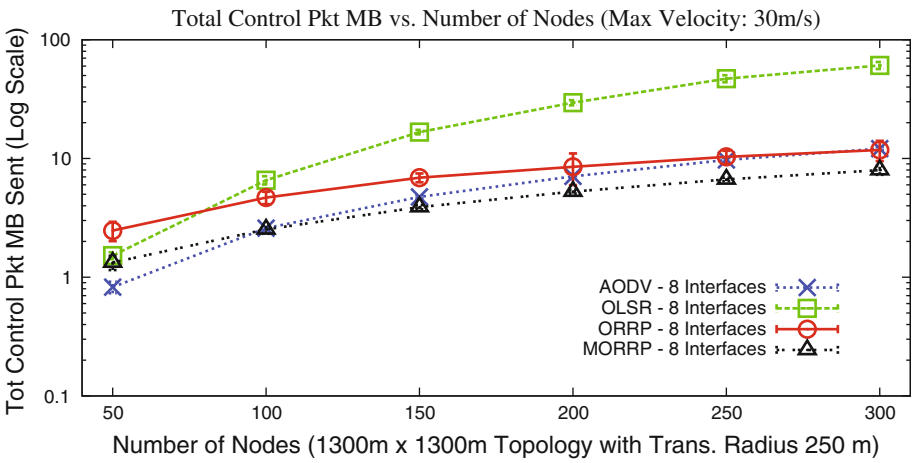


Fig. 15 MORRP incurs low packet overhead even in high mobility

because it was never designed for mobility and maintenance of straight-line paths becomes difficult in highly mobile environments. The gains from MORRP come from protocol design. Much like the majority of previous work in using directional interfaces in layer 3 routing [2, 7], the modified versions of OLSR and AODV simply *adapt* the protocol to support directionality rather than leveraging the *inherent properties* of directionality to route. Whereas OLSR and AODV even with multiple directional interfaces simply "broadcast" out all intervals for topology control dissemination or route discovery, MORRP utilizes local directionality to disseminate packets along lines to limit flooding. Therefore, it is understandable to see large gains with MORRP over OLSR and AODV with multiple interfaces.

5.2.4 Summary of performance evaluation

Below we summarize our findings in evaluating MORRP:

- MORRP yields above 93% reachability even in highly mobile environments for medium-sized networks and 89% reach for large-sized networks with medium density.
- Routing using MORRP accounts for an almost 10-14x higher aggregate goodput compared to AODV, OLSR and GPSR/GLS. These gains come primarily through more efficient reuse of the medium under heavy load.
- MORRP yields 15–20% higher aggregate goodput compared to modified versions of AODV and OLSR for eight directional interfaces and also ORRP. These gains come by using directionality constructively and scalably to overcome problems inherent with directionality.
- End to end packet latency is very low under MORRP compared to AODV, OLSR, and GPSR/GLS because of more efficient medium reuse.
- As node density increases, AODV, OLSR and GPSR/GLS data delivery success drops significantly due to network saturation but does not affect MORRP much.
- MORRP sends *less* control packets than ORRP and much less than AODV, and OLSR in highly mobile situations.

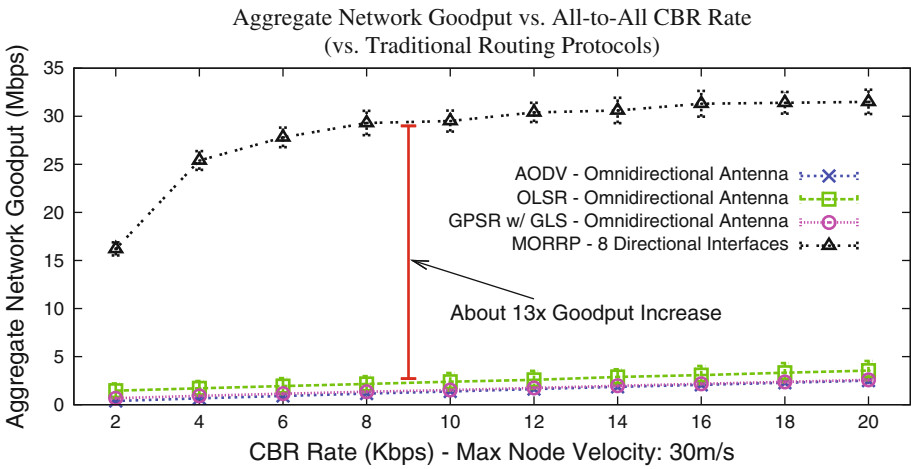


Fig. 16 MORRP achieves 10–14× more aggregate goodput compared to routing protocols with omnidirectional antennas due to efficient medium reuse

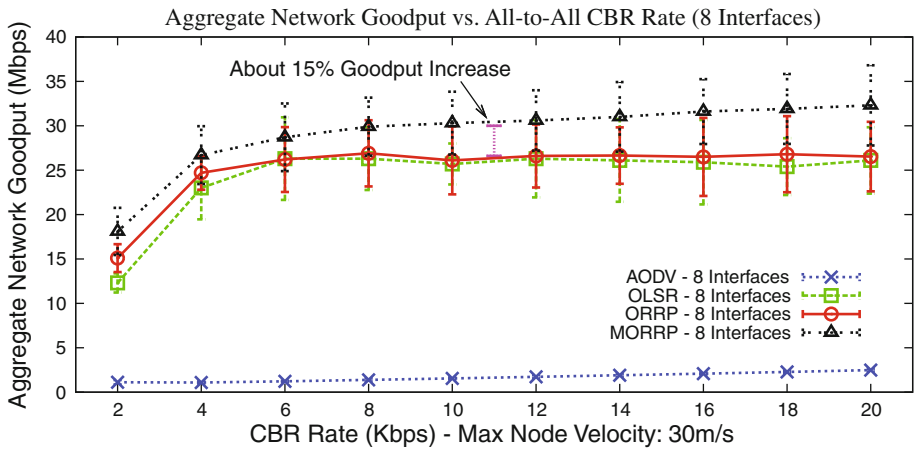


Fig. 17 MORRP achieves 15–20% more aggregate goodput over protocols with eight directional interfaces because of use of directionality to limit flooding

6 Conclusion

In this paper, we presented Mobile Orthogonal Rendezvous Routing Protocol (MORRP), an *unstructured*, *probabilistic*, and *highly mobility tolerant* forwarding paradigm based on directional communication methods and rendezvous abstractions. By utilizing directional routing tables (DRTs), a novel replacement for traditional routing tables, information about nodes in a specific region and nodes along a straight line path is maintained probabilistically. DRTs map interface *directions* to a probabilistic *set-of-IDs* which are decayed and spread locally within a node based on time and *local* node velocity and decayed by number of hops from the source. DRTs provide *regions* where a node can be found in the near-field case and *directions* to send in the far-field case.

When a destination is outside of the near-field region, MORRP relies on taking intersections of orthogonal lines originating from source and destination and forwarding packets from the source to rendezvous nodes which in turn hand them over to the destination providing simplified routing. We have outlined several "knobs" associated with MORRP and evaluated *distance decay factor*, *time decay factor*, and *near-field and far-field threshold* under conditions of varying mobility. It can be seen that spread decay affects networks that are sufficiently dense and has very little affect on sparse networks. Additionally, we compared MORRP against DSR, AODV, OLSR, and GPSR/GLS ORRP under varying conditions of mobility and node densities and found that MORRP provides higher reach probability, average path selection, and has much lower control packet overhead. In short, MORRP provides high connectivity even in highly mobile, dense, and unstructured environments.

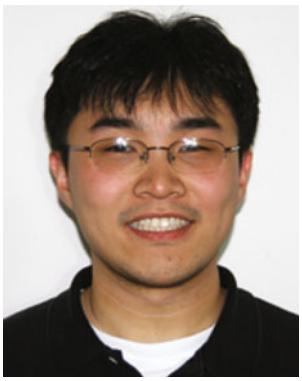
While we have only considered the base case of MORRP in square topologies with random waypoint mobility, there are several directions for future work. First, it would be interesting to see how MORRP fits into hybrid routing environments with networks having a mixture of nodes with omnidirectional and directional communications. It would be interesting to see how to incorporate routing metrics into MORRP and DRTs to provide for even better path selection and obstacle avoidance. Another area of consideration is a more detailed evaluation of MORRP under various topologies and traffic patterns. To explicitly quantify MORRP's performance as a routing protocol, we used CBR traffic without end-to-end congestion control in

our evaluation of MORRP. However, it would be interesting to observe interaction between MORRP and TCP traffic with end-to-end congestion control.

References

1. Choudhury, R. R. & Vaidya, N. (2003). Impact of directional antennas on ad hoc routing, Proceedings of the Eighth International Conference on Personal Wireless Communication (PWC), Venice, September 2003.
2. Ramanathan, R. (2001). On the performance of ad hoc networks using beamforming antennas, Proceedings of ACM MOBIHOC, October 2001.
3. Britz, D., & Miller, R. (2007). Mesh free space optical system: A method to Improve Broadband Neighborhood Area Network backhaul, Proceedings of IEEE LANMAN, Princeton, NJ, June 2007.
4. Yuksel, M., Akella, J., Kalyanaraman, S., & Dutta, P. (2007) Free-space-optical mobile ad-hoc networks: Auto-configurable building blocks, To appear in ACM/Springer Wireless Networks.
5. Yi, S., Pei, Y., & Kalyanaraman, S. (2003). On the capacity improvement of ad hoc wireless networks using directional antennas, Proceedings of ACM MOBIHOC, pp. 108–116, Annapolis, MD, June 2003.
6. Nasipuri, A., Mandava, J., Manchala, H., & Hiromoto, R. (2000). On demand routing using directional antennas in mobile ad hoc networks. Proceedings of IEEE WCNC.
7. Choudhury, R. R., & Vaidya, N. (2004). Performance of ad hoc routing using directional antennas Journal of Ad Hoc Networks—Elsevier Publishers, November 2004.
8. Cheng, B., Yuksel, M., & Kalyanaraman, S. (2009). Orthgonal rendezvous routing protocol for wireless mesh networks, To appear in IEEE/ACM Transactions on Networking (ToN), June 2009.
9. Acer, U., Kalyanaraman, S., & Abouzeid, A. (2007). Weak state routing for large scale dynamic networks. Proceedings of MOBICOM 2007, Sept 2007.
10. Kumar, A., Xu, J., & Zegura, E. (2005). Efficient and scalable query routing for unstructured peer-to-peer networks, in *IEEE Infocom*.
11. Mitzenmacher M. (2002). Compressed bloom filters. *IEEE/ACM Transactions on Networking, 10*(5), 604–612.
12. BBN Technologies. Hazy Sighted Link State (HSLS) Routing: A scalable link state algorithm, http://www.cuwireless.net/downloads/HSLS.pd.
13. The Network Simulator. ns-2. http://www.isi.edu/nsnam/n.
14. HLS patch for ns-2.29. http://www.cn.uni-duesseldorf.de/staff/kiess/software/hls-ns2-patc.
15. OLSR-UM Implementation for ns-2.29. http://www.masimum.dif.um.es/?Software:UM-OLS.
16. Perkins, C. & Royer, E. (1999). Ad hoc on-demand distance vector routing. Proceedings of the 2nd IEEE Workshop on Mobile Computing Systems and Applications, New Orleans, LA, February 1999, pp. 90–100.
17. Optimized Link State Routing Protocol RFC. http://www.ietf.org/rfc/rfc3626.txt.
18. Karp, B. & Kung, H. T. (2000). GPSR: Greedy perimeter stateless routing for wireless networks, Proceedings of MOBICOM 2000.
19. Li, J., Jannotti, J., De Couto, D., Karger, D., & Morris, R. (2000). *A scalable location service for geographic ad hoc routing*. Boston MA: ACM Mobicom, pp. 120–130.
20. Gossain, H., Joshi, T., De Morais Cordeiro, C., Agrawal, & Dharma P. (2006). DRP: An efficient directional routing protocol for mobile ad hoc networks. *IEEE Trans. Parallel and Distributed Systems, 17*(12), 1439–1451.
21. Chebrolu, K., Raman, B., & Sen, S. (2006). Long-distance 802.11b links: Performance measurements and experience. Proceedings of MOBICOM Los Angeles, USA.
22. Johnson, D., Maltz, D. A., & Broch, J. (2001) DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad Hoc Networking*, Chapter 5, Addison-Wesley, pp. 139-172

Author Biographies



Bow-Nan Cheng is a member of the Technical Staff at MIT Lincoln Laboratory. He received M.S. and Ph.D. degrees in Computer Systems Engineering at Rensselaer Polytechnic Institute in Troy in 2005 and 2008 respectively. He holds a B.S. degree in Electrical Engineering from the University of Illinois at Urbana-Champaign. His research interests include wireless routing using directional communications methods in hybrid free-space-optic (FSO), RF disruption/delay tolerant, and airborne wireless networks. He is a member of IEEE and ACM.

Murat Yuksel is an Assistant Professor at the University of Nevada - Reno. He received M.S. and Ph.D. degrees in Computer Science from Rensselaer Polytechnic Institute in 1999 and 2002 respectively. He holds a B.S. degree in Computer Engineering from Ege University, Izmir, Turkey. His research is on various networking issues such as wireless routing, free-space-optical mobile ad-hoc networks (FSO-MANET), network modeling and economics, protocol design, peer-to-peer, and performance analysis. He is a member of IEEE, ACM, and Sigma Xi.

Shivkumar Kalyanaraman is with IBM India Research Laboratory, Bangalore, India. He is a Professor (on leave) at the Department of Electrical, Computer and Systems Engineering at Rensselaer Polytechnic Institute in Troy, NY. He holds a Computer Science B.Tech degree from IIT, Madras, India, and M.S., Ph.D. degrees from Ohio State University. His research interests include various traffic management topics and protocols for emerging tetherless networks. He is a senior member of IEEE and a senior member of ACM.