Wireless Netw (2011) 17:1095–1112
DOI 10.1007/s11276-011-0339-6

Efficient routing in MANETs using ordered walks

Stephen Dabideen · J. J. Garcia-Luna-Aceves

Published online: 5 April 2011

Abstract A new approach for routing protocols operating in MANETs is presented in which flooding is not required to establish paths from sources to destinations on demand in MANETs of moderate size. The concept of *ordered walk* is introduced as a depth-first search (DFS) that does not rely on geographical or virtual coordinate information and is much more efficient than mere random walks. The benefits of using DFS as the building block of the signaling in MANET routing protocols are exemplified by the introduction of the Ordered Walk Search Algorithm (OSA), which is used as part of the proposed Ordered Walk with Learning (OWL) protocol. OWL integrates OSA with the learning of paths from prior successful and failed attempts, and performs one or multiple concurrent ordered walks to search for destinations. Simulation experiments are used to compare the performance of OWL against that of well-known MANET routing protocols based on BFS (e.g., OLSR and AODV). The results show that OWL can achieve a performance comparable to traditional protocols that rely on some form of flooding of link states or network-wide dissemination of distance information in terms of packet delivery ratios and average end-to-end delays, while incurring up to ten times less overhead than AODV.

Keywords Routing · MANET · Depth first search · Search algorithm

S. Dabideen (✉) · J. J. Garcia-Luna-Aceves
Department of Computer Engineering, University of California, Santa Cruz 1156 High Street, Santa Cruz, CA 95064, USA
e-mail: dabideen@soe.ucsc.edu

J. J. Garcia-Luna-Aceves
e-mail: jj@soe.ucsc.edu

1 Introduction

Over the past decades, many routing protocols have been proposed using various mechanisms to build, maintain and repair paths between nodes in a network. As varied as the routing proposals for MANETs are, none of the common proactive or on-demand schemes are very efficient under all traffic-load scenarios. We argue that, at least in part, this is due to the way in which signaling packets are disseminated in these protocols. In both proactive and on-demand solutions, overhead packets can be propagated to regions where they have no significance but nonetheless consume scarce network resources. In particular, on-demand routing protocols can initiate a BFS by flooding the entire network to discover a node that may only be a few hops away. Such an inefficient use of network resources cannot be part of an efficient solution to routing in MANETs.

Section 2 summarizes prior work aimed at reducing the signaling overhead incurred in routing protocols for MANETs. As this brief survey indicates, the prior work has been aimed at reducing the number of nodes engaged in the signaling of routing algorithms based on breadth-first search (BFS), reducing the amount of BFS signaling information that must be disseminated, or establishing virtual topologies that may be maintained more efficiently. The motivation for these approaches is that BFS schemes flood the network, or at least a very large number of nodes to find a destination that is far away. If too many nodes are performing BFS simultaneously, the routing overhead can saturate the network making it impossible to deliver data packets.

Depth-first search (DFS) has been studied extensively in the past as an alternative to BFS, and many distributed algorithms for DFS have been reported [1–3]. However, DFS has not been used much to support the signaling of

routing protocols in MANETs. Apart from ordered walks, to our knowledge, the only efforts that have addressed DFS focus on random walks [4, 5], routing using location information (e.g., GPSR [6], WSR [7]) and protocols that establish and maintain virtual topologies using DFS, such as Virtual Ring Routing (VRR) [8]. Intuitively, route computations based on DFS should involve a much smaller number of nodes and consequently would incur significantly lower overhead when compared to BFS. However, a raw DFS scheme may incur much longer delays in finding the desired routes to the destination; furthermore, in the worse case, all nodes in the network may have to be involved in the routing computation. With ordered walks [9], a variety of mechanisms are added to improve the performance of the depth-first search so that the delay is comparable to BFS approaches but without incurring the extensive overhead associated with BFS.

The question regarding the existence of some optimal middle-ground between the speed of discovery in BFS and the low overhead of DFS is of great interest, because it can lead to the most efficient solution yet. In theory, the use of multiple DFS can lead to a quicker route discovery; however, each additional search necessarily increases the overhead, and as the number of searches increase, the end result becomes closer to a BFS scheme. Ordered walks [9] have been shown to be remarkably efficient and in this paper we further develop the concept and explore the consequences of using multiple ordered walks to further improve performance.

Section 4 presents the *ordered walk search algorithm* (OSA) and analyzes its potential in terms of its time complexity and the signaling involved. Instead of performing a search in a completely random manner, or assuming knowledge of the relative positions of destinations, OSA distributively constructs an approximated minimum-depth spanning tree. Then OSA searches this tree attempting to minimize the number of search messages, such that the resulting path is of reasonable length. The only requirement for this type of search is two-hop neighborhood information. We demonstrate that, given some neighborhood information, a minimum-depth spanning tree can be approximated, and that this approximation converges to the actual minimum-depth spanning tree as the size of the known neighborhood increases.

Section 5 presents the *Ordered Walk with Learning* protocol (OWL), an example of efficient routing in MANETs based on DFS without the need for location information. OWL uses OSA to establish routes on demand based on DFS. OWL can search one or multiple paths simultaneously using ordered walks. At least one of these paths is based on past location information (when available) and the others may be independent of the past locations of the destination.

In Sect. 6 we present extensive results of simulation experiments illustrating that, in terms of, end-to-end delay of data packets and the signaling overhead incurred, OWL attains better performance than popular on-demand and proactive routing protocols based on BFS. The results also show that the use of two simultaneous ordered walks increased the overhead by far more than it decreased the duration of the route discovery phase. Section 7 presents our concluding remarks.

2 Related work

Reactive routing schemes were developed to reduce routing overhead in mobile ad-hoc networks. Today, reactive (or on-demand) routing protocols have become synonymous with the flooding of route requests (RREQs) when a path needs to be established. While this approach may be the fastest solution in a network that is not bandwidth-limited, it leads to the *broadcast storm* problem as identified by Ni et. al. [10], especially in volatile routing environments. This inefficiency has been identified by many in the past, and several optimizations over this *blind* flooding have been proposed. These approaches include the use of an expanding ring search [11], the use of heuristics based on connected dominating sets to reduce the number of nodes retransmitting packets [12], the use of geographical information to direct the flooding [6] and probabilistically reducing the number of retransmissions [10]. While these schemes alleviate the broadcast-storm problem, they do not address the need for flooding which is inherent in any BFS approach.

The most common strategy to reducing the overhead of route signaling in the past has been hierarchical routing, which dates back to the design of the DARPA packet radio network (PRNET), starting with the scheme proposed by Kleinrock and Kamoun [13]. Since then, there have been many other hierarchical routing schemes [14, 15]. The limitations of hierarchical and hybrid routing schemes is that they do not address the inherent need for flooding in addition to the need to update the affiliation of nodes to clusters or zones when nodes move away from their home clusters or clusters are partitioned into two or more components due to mobility.

Broadcast backbone networks [16, 17] have been used to mitigate the cost of flooding in the network. Instead of having every node flood the network, a connected subset of nodes, which together dominate all the nodes in the network, are responsible for forwarding overhead packets. Once such an infrastructure is in place, searching the network can potentially be more efficient than having all nodes flood the search packets. The problem with this approach lies in establishing and maintaining the infrastructure, especially in the face of mobility, where the cost can be

comparable to flooding. Connected dominating sets make most sense in dense networks where there would otherwise be too much redundant broadcast of control packets but sparse networks would see little benefit from such infrastructure.

Apart from ordered walks, there have been only a few attempts to solve the problems incurred with flooding by using DFS instead of BFS. These approaches have focused on the use of *random walks* [4, 5] in which a route request starts at the source and travels along a single path, composed of consecutive random next-hop choices in the search for the destination. The limitation of such a brute-force application of DFS, as we discuss below, is that the communication complexity incurred in reaching destinations may be comparable to that of flooding, but with much longer delays. The successful application of DFS to route discovery requires more creative solutions.

Approaches that have improved on random walks in the past use location information for the routing of packets. (e.g., GPSR [6], WSR [7]). The many proposed schemes have made strong cases on performance benefits that can be attained but their limitations are that each node must know its location by some external means (e.g., GPS [6]), and that the sources need to know the locations of their target destinations. The hint-based approach [18] does not use distances to route, but instead uses a *hint* value which is based on the time of last contact with the destination of interest and the distance of propagation of this hint. In this scheme the ordering resembles the topology but there is no need to maintain strictly accurate ordering information as in distance based protocols. The hints merely reflect the probability of the node being in the right direction of the destination. A disadvantage of this approach is the need for data packets to carry the path traveled to ensure loop freedom.

A different approach to routing is through the establishment of a virtual topology as in the Virtual Ring Routing (VRR) protocol [8]. In this protocol each node is assigned an identifier independent of its location in the network and maintains paths to a subset of nodes in the network closest to its identifier, creating a virtual ring in which nodes maintain paths to their virtual neighbors proactively regardless of path lengths. The advantage of this approach is that nodes do not need to maintain, or even discover on-demand, paths to every other node in the network but only to a subset of nodes (i.e. their virtual neighbors). The virtual topology may have no semblance to the physical topology and routing based on the virtual topology can result in a logarithmic stretch [19] with the use of short-cuts. Although these protocols do not gather complete topology information, establishing and proactively maintaining the virtual ring in a highly mobile environment can incur unnecessary overhead, as we shall discuss in the Sect. 5.9.

In previous work [9], we have presented the use of a single ordered walk to search for destination nodes. As long as the network is not too large, ordered walks can be an efficient alternative to flooding. However, the delay in route discovery using this approach can be intolerable in very large networks. Searching a single path at a time can be a slow process, especially if the path searched is based on past location information that is no longer valid due to network changes or mobility. The use of multiple simultaneous ordered walks, including some walks that do not depend on past location information, can lead to faster route discovery without the need to flood or incur nearly as much overhead as flooding. Furthermore, the use of multiple search paths leads to an increased rate of learning in the network.

3 Motivation for using DFS in route signaling

BFS has remained the most popular choice for on-demand route discovery in MANETs because, at least in principle, a well designed scheme based on BFS is the fastest approach to establishing the desired routes requiring on average $O(log_k\ N)$ time, where k is the average network connectivity. However, the price paid for this search speed is the signaling overhead incurred, which is $O(N)$ in a network with broadcast links. In practice, given that MANETs are bandwidth limited, this large communication complexity means that BFS may not succeed at establishing routes quickly, because signaling packets may suffer long queuing delays or even losses due to multiple access interference. In proactive routing protocols, the time complexity for establishing routes is $O(1)$ as routes are readily available, but this comes at the signaling complexity $O(N^3)$ as every node broadcasts its topology table (of size $O(N^2)$) in the worst case. We discuss this further in the next section.

DFS may be a viable option to avoid the problems introduced by the broadcast storm associated with flooding. However, applying a DFS strategy to on-demand routing means that route requests are propagated from one node to a single neighbor and thus travel a single path from the source to the destination. Accordingly, using DFS in a graph in which each step is completely random means that the time required to find a path to a given destination is $O(N)$, which is the case of the destination being the last node searched, and the average complexity would be $O(N/2)$ for both time and number of messages. Hence, compared to BFS, DFS offers only a constant factor improvement but with a very large penalty in the time complexity it incurs. Clearly, DFS schemes based on random walks make sense only in networks where bandwidth is at a premium and delays in finding paths are not important, which *may* be the case in some sensor networks with static topologies.

We advocate a new approach to DFS applied to on-demand routing in MANETs that takes advantage of two important characteristics of MANETs. Firstly, some local topology information is readily available to nodes due to the broadcast nature of radio links. In particular, a node can hear over time about the presence of neighboring nodes, and even the presence of its neighbors' neighbors. Secondly, MANETs are not completely random and, more importantly, source-destination dialogues follow patterns of interest, which means that sources will be able to find destinations more effectively , over time, than by random walking if their searches choose "children" in the DFS tree based on this prior knowledge that incorporates information gathered from past searches, some of which may have even failed. Accordingly, our approach to using DFS in on-demand routing is based on the concept of "ordered walks," which we describe in the next section, and complement this DFS approach with learning gained from prior walks.

4 Ordered walk: a hybrid distributed search algorithm

With the ordered-walk search algorithm (OSA), we aim to take advantage of the smaller time complexity of BFS and combine it with the low communication complexity of DFS to further improve the efficiency of the search through the use of known topology (i.e., path) information. The basic idea is to approximate the construction of a minimum-depth spanning tree rooted at the source (as in BFS) and then perform DFS on this tree. If there is information about the past location of the destination, then this can be used to guide the search. We call this an *informed search*, and while such information can help the search, it is not necessary for the OSA.

4.1 Efficient uninformed searches

It is often the case in the establishment of new paths that no information about the destination is available at the node making a search decision. To make the search efficient, nodes can take advantage of topology information they have about their neighborhood. In particular, the number of nodes covered (in the known neighborhood) needs to be maximized while the number of nodes relaying the query needs to be minimized by the choice of next hop in the search. To achieve this, successive nodes in the search should have as few neighbors in common as possible. This is possible if two-hop neighborhood topology information is known by performing set comparisons between the current search node and its neighbors. This would usually favor choosing nodes physically far apart, rather than close together, and consequently results in shorter paths than a random walk, where a near node and a far node have the same probability of being the successor in the search. While set comparisons can be computationally intensive, it should not be an issue with modern technology.

At times it would be desirable to direct the search in a physical direction without knowing the relative positions of the nodes. A notion of direction can also be attained using two-hop neighborhood information. To guide the search away from a node, the neighbors of that node should not be allowed to be a successor in the search tree. To guide the search along the current trajectory, the neighbors of all previously searched nodes should be pruned from the search tree. This can be seen in Fig. 1. Once a node eliminates the one hop neighbors of the previous nodes in the search, its remaining neighbors will follow a trajectory away from the origin of the search.

4.2 Approximating a minimum-depth spanning tree

Given complete topology information, any node can construct a minimum-depth spanning tree in a connected network. The source becomes the root (depth 0) for the search. Nodes connected to the source are then at depth 1, their neighbors not yet included are at depth 2, and this process continues until all nodes n hops away from the source are at depth n in the tree. Performing the search on a minimum-depth spanning tree results in paths with a shorter expected length than those obtained with a random walk search.

Maintaining complete topology information amounts to proactive routing and is especially difficult in mobile networks. Fortunately, such a tree can be approximated and constructed distributively requiring as little as 2-hop neighborhood information with the approximation becoming closer to the actual minimum depth spanning tree as the width of the neighborhood increases.

The first step is establishing the n-hop neighborhood. This is be done by having nodes advertise distances to all other nodes within $(n - 1)$ hops. A node receiving such an advertisement would store the distance to every node in its n-hop neighborhood from each of its neighbors in a routing table. If n is equal to the diameter of the network, then each node maintains distances from each of its neighbors to every other node in the network which is proactive routing. By maintaining only the n-hop topology we reduce the complexity of the signaling.

At each step in the search, say at depth x, a node must choose children that are at a greater depth $(x + 1)$ in the search tree than the node itself. If the current depth is less than n (the radius of the known neighborhood) then the children can be chosen so as to guarantee that they are at a greater depth in the actual minimum depth spanning tree. If the current depth of the search tree is greater than n, then the exact depth of the neighbors with respect to the root cannot be ascertained and an approximation must be made.

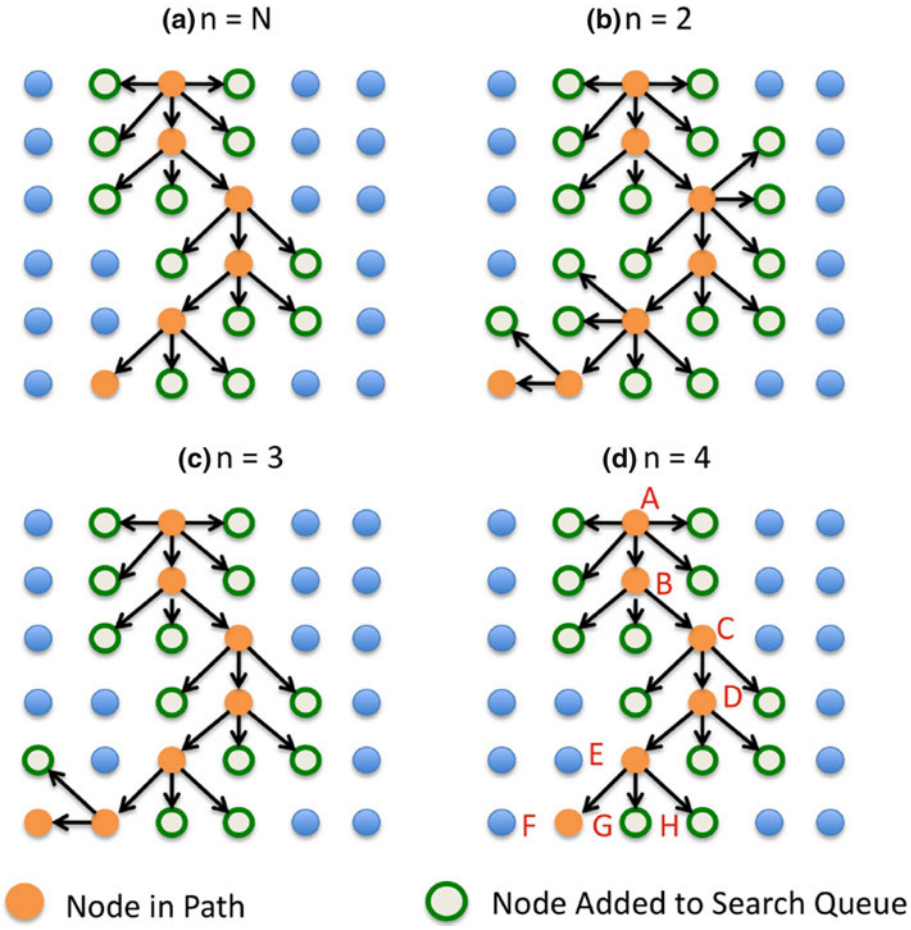

Fig. 1 DFS pruning in OSA

We make the approximations based on the node closest to the root of the tree within the known neighborhood. Let p be the node exactly $(n-1)$ hops before x in the search path. Then, if y is a child of x in the search tree then y must be exactly n hops from p. If y is less than n hops from p then y is not at a greater depth than x in the actual minimum depth tree and we can prune all such nodes. Node y is at most 1 hop further from p than x, therefore y must be exactly n hops from p. The larger the value of n, the more accurate the tree becomes. If $n = 2$, the reference point will be the previous hop in the search path and can prune all the 1 hop neighbors of this node. If $n = 3$, the reference point is the node 2 hops before in the search path and we can preclude all nodes that are either 1 or 2 hops from this node from being at depth $(x + 1)$. If $n = 4$, then the reference point is 3 hops up the path of the search and we can preclude all neighbors which are 1,2 or 3 hops from this node. In this process of eliminating nodes from the search tree we need the distance from each 1-hop neighbor to the reference point which was designed to be in the known neighborhood.

Figure 1 shows four executions of OSA. In the first case, complete topology information is known and in the subsequent cases, only 2-hop, 3-hop or 4-hop information is known. With 2-hop neighborhood information, we only prune the 1-hop neighbors of the previous node in the search as seen in Fig. 1(b). As we increase the size of the known neighborhood more nodes are pruned from the search tree and in this case, $n = 4$ results in the same path as the case where global topology is known, but this will not hold for larger networks. With known global topology, we are guaranteed the shortest path as can be seen in Fig. 1(a). With an approximated tree the path can be longer than the shortest path as evident in Fig. 1(b, c). Consider Fig. 1(d) where the search starts at A and the 4-hop neighborhood is known. A is the root (at depth 0) and must add nodes at depth 1 (1 hop from A) to the search queue and this corresponds to all of $A's$ neighbors. Of its neighbors if B is determined to be the best, B must now find nodes at depth 2. Since $n > 2$, B simply checks which of its neighbors are 2 hops from A and adds those nodes to the search queue. There are 3 such nodes and suppose C is the best node. Now C is at depth 2 and must determine which of its neighbors are at depth 3. Once again $n > 3$, and C can add its neighbors which are exactly 3 hops from A to the search queue. Once the search reaches E, which is at depth 4, we must start

making approximations. E has neighbors which do not have A in their 4 hop neighborhood so it cannot use A as a point of reference. Instead it will use B. E will add *only* its neighbors which are 4 hops from B to its search queue. Only 3 of its neighbors satisfy this criteria. F will then examine its neighbors and add those which are 4 hops from C to the search queue. There are no such nodes so F will be pruned from the tree and E will choose a different neighbor. If neither G nor H are the destination, the search will return to node D which will chose its next best neighbor, which has new neighbors to explore and the depth first search will proceed in this manner.

4.3 The ordered walk search algorithm

The pseudocode for the *ordered-walk search algorithm* (OSA) is provided in Algorithm 1. It takes as input a graph G, a source node s, and the destination node d that is being searched, and returns a path from s to d, should one exist. It also assumes that any node knows its n-hop neighborhood in the network, where $n \geq 1$. The larger the value of n, the more effective the algorithm would be as the search will terminate once it arrives in the n-hop neighborhood of the destination. However, maintaining accurate topology information beyond one-hop becomes increasingly difficult and there is a trade-off between the complexity and the precision of the algorithm.

OSA proceeds as DFS, using the approximated minimum-depth spanning tree. At each step in the ordered walk, a node must choose its successor in the search tree. An informed search would be prioritized if the relevant topology information were available, but if this is not the case, an uninformed choice is made along the current trajectory of the search.

OSA uses a function N_1(x) that returns the set of 1-hop neighbors of x, and therefore possible children in the tree. If we are at a depth less than n in the search tree then we can use the root as a point of reference (line 15) to determine which of $x's$ neighbors are actually at a greater depth and prune those which are not (lines 20, 21). If, however, we are at a depth greater than n, we do not know the distance from the neighboring nodes to the root of the tree and we must use a heuristic and the point of reference becomes the node $(n - 1)$ hops higher in the search tree (line17) and use this to prune some, but not necessarily all, of the nodes which are not at a greater depth. If $n = 2$ then our point of reference is always the previous hop in the search path and we prune only its one hop neighbors.

OSA also uses the SORT() function, which takes as input a set of nodes and places them in ascending order of distance to d, such that the nodes closest to d are searched first. The notion of proximity to the destination is based on past known locations of the destination or if the destination lies in the known neighborhood. If neither of these are applicable, the neighbor with the least common neighborhood is use to sort the nodes. Nodes that are viewed as equidistant are placed in random order.

Algorithm 1 OSA(G, s, d)

1: $P_0 \leftarrow s$
2: $depth \leftarrow 0$
3: $T \leftarrow N_1(s)$
4: $Q \leftarrow SORT(T)$
5: $n \leftarrow$ *number of hops in known neighborhood*
6: **while** $Q \neq \phi$ **do**
7: $x \leftarrow$ first member of Q
8: $depth \leftarrow (depth + 1)$
9: $P_{depth} \leftarrow x$
10: **if** $x = d$ **then**
11: **return** (P_{depth}, $P_{depth\text{-}1}$… P_0)
12: **end if**
13: $T \leftarrow N_1(x)$
14: **if** ($depth < n$) **then**
15: $p \leftarrow P_0$
16: **else**
17: $p \leftarrow P_{(depth-n)}$
18: **end if**
19: **for all** $x_i \in T$ **do**
20: **if** *distance* $(x, p) \geq$ *distance* (x_i, p) **then**
21: $T \leftarrow T - x_i$
22: **end if**
23: **end for**
24: $SORT(T)$
25: Add T to front of Q in order
26: **end while**
27: **return** ϕ

4.4 OSA correctness

The following theorems show that OSA obtains finite paths from sources to destinations, if they exist, using only local information regarding a node's neighborhood and path information accumulated over time.

Theorem 1 *OSA terminates for any finite connected graph with N nodes.*

Proof Once a node is visited in the search, it is added to P. For any node, $N_0(x) = x$, and thus any node in the path is never added to the queue Q, because it is never added to T in line 18 of Algorithm 1. Given that the number of nodes is finite and each node can only be searched at most once, the algorithm terminates when d is found or when there are no more nodes in the queue Q, which happens in at most N steps.

Theorem 2 *OSA terminates for any finite disconnected graph with N nodes.*

Proof The search is be confined to the component of the graph from which it originated. The search does not leave this component of the graph, else the component would have to be connected to another component. All nodes within this component are connected and this component can be treated as a finite connected graph. Hence, by Theorem 1, it follows that the search terminates on this subgraph.

Theorem 3 *For a finite graph G, OSA returns a path from the source s to destination d, if one exists.*

Proof It can be seen from Algorithm 1 that a node is not added to the search tree T at a particular instant if and only if it is either already in the search tree or in the known neighborhood N_k (x) of a node already in the search tree.

If s and d are in different components of the graph, the theorem follows. Assume for contradiction that s and d are in the same connected component of the graph G and OSA terminates without returning a path from s to d. Then it must be the case that d was never added to the search tree T. If there was a path, it would have been found when d was searched. If d was never added to the search tree, then none of its neighbors could have been added to T, because then d would have been added when its first neighbor was searched. This same argument can be applied to the neighbors of d and can be repeated until s is reached which must be the case because they are in the same connected component. Accordingly, it must be true that the connected component containing d cannot contain any member of the search tree T and therefore cannot contain s, which contradicts the initial hypothesis. Therefore, the theorem is true.

4.5 OSA complexity

For the sake of discussion, consider a network G with node degree k. If a path from a source s to the destination d is known, then the complexity of OSA is $O(log_k\ (N))$. This corresponds to the depth of a k-ary tree, and is in fact the best upper bound on the run time of any search algorithm; however, it is not really a search if a path is already known. It is well-known that the complexity of a centralized DFS algorithm is $O(V + E)$, which in this case is reduced to $O(N + k*N)$, which is $O(N)$. If the n-hop neighborhood is known, then the search will terminate in at least n fewer steps. Therefore, the time complexity will be reduced to $O((N - n) + k*(N - n))$. For a small n, such as $n = 2$ the time complexity is still $O(N)$ if there is no information to guide the search. If however, n is large, the time complexity is reduced and becomes $O(1)$ when $n = N$, which corresponds to proactive routing.

OSA has a proactive and an on-demand component and the total overhead in the network would be a consequence of the searches plus the overhead to maintain n-hop neighborhood information. To maintain the topology information of the n-hop neighborhood, each node must advertise all nodes within $(n - 1)$ hops. The size of such a packet in a network with average degree k is $O(k^{n-1})$. Therefore, the total message complexity of OSA is $O((N - n) + k*(N - n) + k^{n-1})$. As n approaches N the protocol becomes a proactive routing protocol and no search is required but the overhead increases exponentially. Unlike proactive protocols, if only two-hop neighborhood information is maintained, hello messages are kept small because only immediate neighbors are advertised, as opposed to complete topology information. With small n, the time and message complexity of DFS are maintained. The accuracy of the ordering determines how far the actual search deviates from this worst-case value and how close it approaches the optimal value of $O(log_k(N))$. The message complexity of un-optimized link state is $O(N^3)$ [20] while that of link state using multipoint relays (such as OLSR) is $O(N(Log(N)^2))$ [20]. There have been other optimizations to reduce the overhead such as fish-eye [14] routing and hazy-sighted link state [15] but the complexity will be worse than $O(n)$ as long as each node must broadcast more than its 1-hop neighborhood.

We note that the search space in OSA is reduced by a factor of k for every step that is known to be correct. This is illustrated in Fig. 2, which shows OSA being executed on a binary tree. The direction of the arrows shows known paths. It is clear that, with every step taken towards the destination, the number of nodes in the search tree is halved. For pure DFS or pure BFS only one node is removed from the search tree in any give step. In real MANETs, the situation is not as simple as in Fig. 2, because nodes would form a mesh instead of a tree and the nodal degree is likely to be variable and greater than two. However, the reduction in the search space at each step is still significant. The experimental results we obtained show that OWL incurs significantly less overhead, which indicates that far fewer nodes needed to be searched before the destination was discovered.

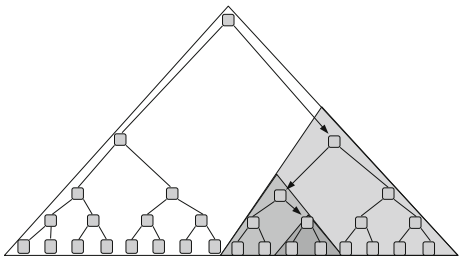

Fig. 2 An ordered walk example

5 OWL: DFS with learning

OSA relies on topology information to enhance the performance of the depth-first search. To demonstrate the effectiveness of OSA, we present the *ordered walk with learning* (OWL) routing protocol, which uses DFS to establish and repair paths from the source to the destination with minimal signaling overhead and fast convergence. OWL is an on-demand routing protocol, which means that paths are established only when needed. However, it uses a proactive component to maintain up-to-date 2-hop neighborhood information i.e. value of n in OSA is set to 2 in OWL.

Larger values of n will increase efficiency of the algorithm as the approximation to the minimum depth spanning tree will be more accurate and once the search enters the n-hop neighbor, a path to the destination will be found. When n is greater than or equal to the diameter of the network, the protocol becomes completely proactive and inherits the advantages and limitations of this approach. The size of the overhead packets to maintain n-hop topology information grows exponentially with n as each node will periodically broadcast packets with all nodes within (n-1) hops and their corresponding distances. Furthermore, we use set comparisons to determine the best next hop and this too becomes costly as the set of known nodes increases. The heuristics developed for proactive routing such as multipoint relays or hazy-sightedness can be applied to reduce the overhead. Nonetheless there is a tradeoff between the total overhead and the degree of accuracy in the approximation of the minimum depth spanning tree. Our main motivation for using a value of n=2 is for simplicity. Nodes only send their 1-hop neighbors in the hello and comparisons involve a few nodes. We aim to show that although the use of such limited information will not result in a very good approximation, the benefit is still enough to yield good results. In static networks, or networks with few topology changes a large value of n would have most benefit since little overhead is needed to maintain the ordering. In more mobile networks, the topology will change often and this incurs overhead which increases exponentially with n and therefore a smaller value of n is desired.

5.1 Routing information

OWL relies on two-hop neighborhood information gathered using periodic Hello packets. Hello packets contain the one-hop neighbors of a node and are sent periodically with a frequency depending on the mobility of the nodes. The default hello packet interval in OWL is initially set to 30 seconds and is then additively increased and decreased based on the number of changes in the neighborhood. When a node needs to send a hello, it first determines the number of new neighbors and the number of nodes that are no longer neighbors. The sum of these two is then subtracted from the current hello interval with a minimum value being restricted to ten seconds. If there is no change in the neighborhood, the hello interval is incremented by five seconds, with the maximum interval being restricted to one minute. We do not allow the hello interval to become too large because it would be difficult and slow to detect changes in the neighborhood if they did occur.

In the hello packets, each neighbor is paired with a one bit flag indicating bi-directional connectivity. If a node receives a hello from its neighbor and it is not listed as a neighbor despite having broadcasted hello packets, then this may be an indication of a unidirectional link. This bit is set to 1 only if the neighbor has advertised the node in its hello packets.

OWL uses two structures to store the information necessary for route discovery and routing of data packets. The first is a neighbor table that stores each of the node's neighbors and keeps a list of each of their one-hop neighbors (as advertised in Hellos). This allows nodes to have two-hop neighborhood information for use when performing ordered walks.

The second structure is a destination table. It stores the latest sequence number and corresponding distance to each known destination. The information in the destination table is updated using path information carried in route requests and route replies. Each node also maintains a *preferred neighbor* to each active destination it knows. This preferred neighbor is the first neighbor from which the node receives the latest route reply for the given destination, and therefore the perceived closest neighbor to that destination. Associated with each preferred neighbor is the number of times that neighbor has been used as the preferred neighbor in a DFS since the last successful search (we call this variable PNUse).

5.2 Route establishment and maintenance

Route requests (RREQs), route replies (RREPs), route errors (RERRs) and hello messages are the basic building blocks of OWL. These packets are used in the same way as many on-demand routing protocols.

5.2.1 Route requests

When a source needs to discover a path to the destination, the source node initiates a RREQ with a new sequence number that serves as a time-stamp. The time-to-live (TTL) for the request is initially set to a value of ten whenever the source node has no idea of the distance to the destination. Once the source discovers the destination, it acquires a

distance to it. On successive searches, the source sets the TTL to the last known distance plus two. This restricts the maximum length of the path discovered. This past information is used to set the TTL for at most two unsuccessful searches. After which the sources uses the default value of ten as the TTL. For networks with larger diameters an appropriate TTL value should be used.

Upon sending a RREQ, the source node sets a timer proportional to the TTL of the route request. If no route reply is received by the time this timer expires, another ordered walk is initiated to search for the destination.

Upon receiving a RREQ, a node updates its routing table using the path information carried in the RREQ. Then the node must either respond with a RREP or forward the RREQ to a neighbor as determined by OSA. The precise manner in which RREQs are handled is stated in Algorithm 2. According to the algorithm, any known path to the destination would be used to forward the RREQ. If the selected neighbor is no longer valid (detected by failed transmissions of the RREQ), the neighbor table is updated and a different neighbor is selected. If there is no valid neighbor, a route error (RERR) is sent to the source of the RREQ.

5.2.2 Route replies

RREPs can only be issued by the destination and this is done upon receipt of a RREQ with a new sequence number. In a mobile environment, the topology information available to any node has limited lifetime. With this in mind, OWL verifies the existence of known paths through the propagation of RREQs all the way to the destination rather than allowing intermediate nodes to respond with RREPs, which may be based on topology information that is no longer valid.

Nodes listen in promiscuous mode for RREPs from their neighbors. Upon overhearing a RREP for which the node is not designated as the next hop, nodes simply record the next hop in the path and do not retransmit the packet.

Nodes store the distance and sequence number carried in the RREQs and RREPs in their routing table. This distance information is useful in ordering the nodes for future searches in the event of link failures.

This ordering information can be used for up to two future searches (as determined by the previously defined variable PNUse), after which it expires and is removed from the routing table. This value can be changed depending on the mobility of the network. Whenever a node receives a RREP with a higher sequence number, it refreshes PNUse, so the preferred neighbor can be used two more times, regardless of prior use.

An ordered walk fails if the TTL expires or a leaf node is reached. A RERR is sent back to the source in both cases, and the source then initiates an ordered walk in a different direction, with the notion of relative direction being derived from two-hop neighborhood information as perviously discussed. This would result in a different branch of the approximated minimum-depth spanning tree being searched in a DFS manner and increases the probability of discovering the destination because it maximizes the number of new nodes being searched.

In a mobile environment, link failures are inevitable and if the path being used no longer exists, the last node before the point of path failure makes up to two attempts to repair the path using neighbors with a known route to the destination. If no such neighbor exists, or the path using the selected neighbor(s) fail, a RERR is sent to the source, which then starts a new ordered walk. This time, there will be some ordering of the nodes with respect to the destination from the previous search and this information is used in the OSA algorithm to make the DFS more efficient. The farther from the source the failure occurs, the easier it will be repaired, because the search tree is significantly pruned with each successful step.

Algorithm 2 HandleRREQ(*RREQ*)

```
1:  P ← RREQ(Path)
2:  PN ← Preferred Neighbor
3:  ProcessPath(P)
4:  if (Node = Destination) then
5:      Initiate RREP()
6:      return
7:  end if
8:  if (TTL = 0 or has No New Neighbors) then
9:      Initiate RERR()
10:     return
11: end if
12: if PN and PNUse > 0 and PN ∉ P) then
13:     PNUse = PNUse − 1
14:     Relay RREQ(PN)
15: end if
16: O ← BestOrderedHop(P)
17: RelayRREQ(O)
18: return
```

5.3 Multi-walk OWL

DFS can be a slower form of route discovery than BFS because fewer nodes are being searched at any given time. To speed up route discovery in OWL, a pair of ordered walks can be used instead of just a single walk. If the walks do not overlap, then the search should be twice as fast.

When a node needs to find a destination, it initiates a pair of RREQs. One of these requests is independent of

past information about the destination and the other uses this past information when available. The use of two paths speeds up the route discovery process and overcomes the possible problem of relying on invalid past information. While past information can often guide the search in the correct direction, it is not always the case that the destination remains in the same vicinity. This problem is mitigated by allowing one of the ordered walks to be independent of past information. RREQs that are issued simultaneously from a source use the same source sequence number. It is possible that a node receives both route request packets and in this case it forwards both of them. This does not result in routing loops, because each route request carries the path information and this is used to ensure that each path is loop-free even if multiple paths intersect. If a node receives multiple RREQs with the same sequence number it must be that there were different ordered walks initiated at the same time.

Intuitively, the advantage of multiple simultaneous walks becomes more pronounced in a network where each node has many neighbors and the nodes are highly mobile. As mobility increases, past location information becomes less useful and searches can take longer to terminate. When there is no past information (e.g. for the first search) or past information is irrelevant to the current search, multiple searches can reduce the search time. However, if there is low mobility, once the destination is discovered subsequent searches within the same neighborhood are more likely to be successful than searches in other areas of the network. In such cases, additional searches will only add to the network overhead with little benefit. As the number of simultaneous walks increases, the search will look more like a BFS and the goal of this paper is to demonstrate that DFS, which is one walk in its strictest sense, can be effective for route discovery.

The source initiates two RREQs, and should both arrive at the destination, two RREPs will be sent, one along each path. This redundancy will allow more options for route repair in the future. However, a source only needs to receive one route reply for the search to be considered successful.

5.4 Path reinforcement

Hello packets are also used to advertise ordering to destinations for which the sending node or one of its immediate neighbors is on the path to the destination. When a path to the destination is set up, the nodes along that path and their neighbors that overhear the RREP designate themselves as *active* for that path for up to two hello intervals after first overhearing the RREP. This reinforces the ordering to the destination and provides alternate routes for routing data and future searches.

5.5 Learning

We have discussed how to make efficient uninformed DFS decisions, but it is even better to make informed search decisions when possible. To accomplish this, nodes need to learn about the topology, but this must be done with little cost.

OWL uses two mechanisms to gain topology information. Periodic "Hello" packets, which contain a list of one-hop neighbors and destinations for which the sending node is active are broadcasted by each node in the network and this is used to provide two-hop neighborhood information. Also, once a node finds the destination, it uses the past path information to guide future searches.

In a mobile environment, past relative position of the destination becomes outdated (more so as the search approaches the destination). Nonetheless, some of the information may be useful.

At the start, most nodes may have no ordering for an intended destination. After each search, some nodes are ordered with respect to other nodes, be they the desired destinations or not. This ordering is important to the success of OWL. Promiscuous listening allows for ordering of nodes near the paths being set up. To cope with mobility, distance information is assumed to be valid for up to two searches per sequence number.

Over time, the ordering in the network can become robust, thus reducing the search time. This claim is substantiated by the simulation results presented in Sect. 6.

5.6 Learning and mobility

With ordered walks, the search is directed towards the last known position of the destination. This is beneficial, as long as the destination remains in the same position or at least very close to that position. In a mobile environment, this is not always the case and, for this reason, a known ordering with respect to the destination is valid only for a few (set to two in OWL), searches. Such an event indicates substantial movement of the destination and an ordered walk is initiated and performed using uninformed routing decisions.

However, nodes do not move very quickly and are likely to gradually move from one neighborhood to another in many MANETs. Once the known path is broken, it is likely that an alternate route can be found that contains most of the nodes as the original route as well as additional nodes to route around the link failure. The results indicate that OWL performs well in the face of mobility.

5.7 OWL versus random walk

We argue that ordered walks are better than random walks in the context of MANETs.

The expected path that results from an ordered walk should be shorter than that of a random walk. The TTL used in OSA forces an upper bound on the resulting path length. The fact that the OSA is performed on an approximated minimum-depth spanning tree ensures that the TTL is sufficiently large to find a path without allowing the path to be excessively long. Furthermore, successive nodes in the ordered walk would be, on average, physically farther apart than those of random walks. In a random walk all neighbors have an equal probability of being the successor in the search, but ordered walks favor successors that are farther apart because these nodes would have fewer neighbors in common. Because the average distance between successive nodes is larger, the average number of hops will be smaller for ordered walks than for random walks.

The expected number of search messages in ordered walks should be smaller than that of random walks. Each successive node in an ordered walk is chosen so as to maximize the number of nodes not yet covered by the search. For any given number of search messages, a random walk would cover at most as many nodes as those covered by an ordered walk. If the destination is randomly chosen, then each node is equally likely to be the destination. Therefore, the more nodes covered by a search, the greater the probability of finding the destination. Hence, for the same number of search messages, an ordered walk would have a probability of discovering the destination that is greater than or equal to that of a random walk.

Furthermore, because the OSA operates on a minimum-depth spanning tree, the search is guaranteed to venture into new neighborhoods, because it is forced to move away from the source with each step. In a random walk, however, there is no such restriction and the search can move through all the nodes in the same neighborhood, and therefore proceed much slower.

5.8 A simple example

Consider the simple network in Fig. 3, where node S needs to send data to node D and there is no previous routing information about D in the neighborhood of S. Node S chooses a neighbor with the fewest neighbors in common with itself. In this example, it first chooses A, which in turn selects B as its successor because they have they fewest neighbors in common. Node B does not have any neighbor that is not a child of a node already in the search path because nodes X and Y are neighbors of node A. Node B cannot proceed to a greater depth of the search tree so it must send a route error to node S.

Upon receiving this route error, node S must choose a new successor with as few neighbors in common with both itself and node A (in an attempt to guide the search in a different direction). In this example, node S chooses node C which then chooses node E. Once the RREQ arrives at node E, a path to the destination is found because the destination, node D, is in node E's two-hop neighborhood. Node E forwards the RREQ along the known path to verify that it still exists. Node D then initiates a RREP in response to the RREQ and this route reply travels along the same path as the route request. Nodes close to this path, such as nodes G and H, overhear the route replies and become active for destination D. Accordingly, they record their preferred neighbor (which is E for the case of node G) and set the number of future searches for which the information is usable to 2. Node H becomes active for destination D and therefore includes the current distance and sequence number for D in its Hello Messages which will provide node C with an alternate (and in this case a shorter) route to node D though node H.

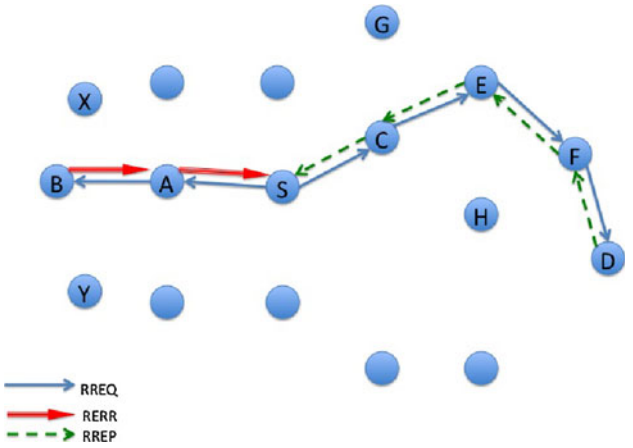

Fig. 3 An OWL example

If we increased the size of the known neighborhood, to say 4 hops, the protocol behaves in a similar manner with few changes. Instead of referencing the previous hop in the path it will reference the node 3 hops before in the path. Since the 4-hop neighborhood information is maintained, a node will know the distance from each of its neighbors to this node and if a neighbor's distance is less than or equal to the node's own distance, then that neighbor is pruned from the search tree.

5.9 OWL versus VRR

The Virtual Ring Routing protocol [8] does not use flooding (BFS) and thus implicitly employs depth-first search in the initialization and maintenance of the virtual ring. Accordingly, it is worth discussing the inherent differences between VRR and OWL.

At network initialization in VRR, a node may know neither the identifiers of its actual virtual neighbors nor their locations. Each node is initially a single-node virtual

ring and needs to merge with others in order to grow the virtual ring until all nodes in each connected component of the network form a single ring. The protocol does not use any network-wide flooding, and therefore the ring is grown incrementally at each node. The DFS that occurs during the initialization of VRR resembles repeated random walks, because a node's view of the network increases with each step and this may cause a change in the virtual neighbors stored at the node which then results in a longer walk. A node constantly replaces its virtual neighbors (members of its *vset*) as it discovers better virtual neighbors (i.e., nodes with identifiers closer to itself than its current virtual neighbours) until the virtual ring is complete.

Nodes discover their physical neighbors (*pset*) from listening to periodic heartbeat messages and initially a subset of these physical neighbors forms the virtual neighbors (*vset*). A node joining the ring, as described in [8], discovers its virtual neighbors by routing a request to itself through a proxy (which is one of its physical neighbors that is already a part of the virtual ring). Let us consider a node X wanting to join the virtual ring. At network initialization, nodes are only aware of their one-hop neighborhoods, so X's proxy forwards the request to its virtual neighbor, call it Y, with the closest identifier to the origin of the request, X. Y sends its vset to X and X then substitutes some of its virtual neighbors with those of Y which have closer identifiers to its own. When X adds a node to its vset, it sends this node a *setup* message to that node which then replies with a *setup* message of its own, containing its own vset which may cause further substitutions in X's vset.

This process continues incrementally along r walks, where r is the size of the vset maintained at each node. In the worst case, each of these walks is of increasing length (first X to Y, then Y to X then X to Y to Z and back, etc) until there is convergence to the actual virtual neighbors. As nodes are substituted in X's vset, messasges (*setup_req, setup, setup_fail and tear_down*) are being sent back and forth at every step. The cumulative number of steps of one such walk will be $O(1+2+3+\ldots+N)$ in the worse case, which is $O(N^2)$, because each walk is repeated, with each repetition being of increasing length. There are r of these walks at each node to find the final r members of the vset, and this is done at all N nodes in the network. Therefore, the complexity of initialization in VRR is $O(rN^3)$ which, interestingly, is worse than flooding.

There may be significantly less signaling after ring initialization depending on the stability of links in the network. If a node joins the network or restarts with no routing state, it uses a proxy to route a request to itself via its virtual neighbor. In the worse case, this virtual neighbor can be up to half-way across the ring and therefore adding a node to the ring is $O(N)$. The existence of shortcuts in the greedy forwarding of messages can reduce the number of steps required, but such shortcuts are not guaranteed to exist. Upon node failure, a node's virtual neighbors must update their vsets. This process is $O(N)$ as with a node joining the established ring. To handle link failures, nodes may need to send multiple *tear-down* and *setup_req* messages to remove any path which uses this link and to establish alternate paths. In the worst case, these messages may need to travel half the virtual ring and is also $O(N)$.

The signaling and time complexity of OWL with two-hop neighborhood information are $O(N)$ as we have previously discussed. The key difference in the DFS schemes between OWL and VRR stems from *when* the searches occur. In VRR, most of the searching happens at initialization in the form of random walks to establish the ring and subsequent searches to maintain the ring are guided based on the established virtual ring. These searches are not based on any notion of distances and are not meant to render shortest paths. They can be efficient based on the availability of shortcuts along the virtual ring, but there is no guarantee of this. In OWL, the DFS only happens on-demand when there is data to route between a source-destination pair. Although distances are not used, with the approximation of a minimum-depth spanning tree together with past location information, the search can be efficient.

6 Experimental results

We compared the performance of OWL with that of representative protocols for on-demand and proactive routing based on BFS in MANETs in the Qualnet 3.9.5 simulator. OLSR [21] was used as an example of proactive routing, and AODV [22] was used as an example of on-demand routing. Our comparison highlights the ability to attain on-demand routing using a DFS approach without relying on location information, and illustrates the fact that such an approach can render comparable results to those attained with the traditional BFS scheme used in OLSR and AODV, but with only a fraction of the signaling overhead. We also compare the performance of single-path searching (which we call OWL1 in the experiments) with dual-path searching as described in the previous section (denoted OWL2 in the results).

6.1 Simulation environment

Two scenarios were used in the simulations. Scenario A was designed to rigorously test the performance of the protocols in a dynamic environment with volatile links. Scenario B, uses a greater radio range to add more stability to the links and create more multi-path opportunities.

Table 1 Constant simulation parameters

Parameter	Value
Simulation time	900s
Node placement	Stationary distribution
Mobility model	Random waypoint
Min-max speed	1–10 m/s
Propagation model	Two-ray
Physical Layer	802.11
Antenna model	Omnidirectional
MAC protocol	802.11 DCF
Data source	constant bit rate (CBR)
Number of packets per flow	400
Packet rate	4 packets per second

Table 2 Simulation results for Scenario B

	Delivery ratio	Latency	Net Load	R/R	Avg. Path Length
Scenario A					
AODV	0.63 ± 0.08	0.07 ± 0.02	13.2 ± 4.0	47	3.7
OLSR	0.36 ± 0.04	0.02 ± 0.01	21.1 ± 0.7	–	2.7
OWL1	0.74 ± 0.04	0.07 ± 0.01	5.2 ± 1.3	8.4	4.4
OWL2	0.76 ± 0.02	0.06 ± 0.01	5.4 ± 1.4	24.6	4.4
Scenario B					
AODV	0.84 ± 0.05	0.05 ± 0.01	6.1 ± 1.2	38.3	2.7
OLSR	0.44 ± 0.06	0.03 ± 0.01	16.1 ± 0.5	–	2.3
OWL1	0.93 ± 0.05	0.03 ± 0.02	1.4 ± 0.5	7.5	3.0
OWL2	0.95 ± 0.03	0.04 ± 0.01	1.9 ± 0.4	10.4	3.1

Scenario A consists of 100 nodes uniformly distributed in a grid of size 1,000 m × 1,000 m with the transmission range of the radios set to 150 m. This choice of parameters satisfies the minimum standards for rigorous MANET protocol evaluation as prescribed by Kurkowski et al. [23] as it results in an *average shortest path hop count* [23] of 4.03 and *average network partitioning* [23] of 3.9%. Other relevant simulation parameters are summarized in Table 1. This scenario ensures that packets travel several hops from source to the destination and thus tests the robustness of the protocols. The mobility model chosen was random waypoint with minimum speed of 1m/s and maximum speed of 10 m/s with a pause time of 30s.

In one set of experiments, we varied the number of flows thus testing the performance of the protocols under varied loads. In the second set of experiments we varied the pause time to test the performance under varying degrees of mobility. In experiments with node mobility we used the random waypoint model to exercise the signaling in the worst case scenario. We sampled the stationary distribution of node speed, remaining pause time and node placement according to the method outlined in [24] and used it as the initial conditions in the simulation. This ensured that the experiments started in steady state and provided consistent results. The parameters for OLSR were set to those specified in [21] (i.e. a hello interval of 2 seconds, TC interval of 5 seconds and neighbor hold time of 6 seconds).

Nodes were randomly selected to be the sources and destinations of CBR flows and care was taken to ensure that there was no node that was the destination of its own flow. There were no restrictions on nodes being multiple sources, multiple destinations or a source of one flow and a destination of another. Each source would send a maximum of 400 packets of size 512 bytes at a rate of 4 packets per second. The start time of each flow was randomly determined using a uniform distribution and was within the duration of the experiment.

Using a time-based seed, 20 random scenarios were generated with the above specifications and the results were used to compare the performance of the protocols. The large number of randomly generated scenarios was used to avoid bias in the results. Each series of random numbers was generated using the *lrand48* command in C and using the current time as a seed for the random number generator.

Scenario B is similar to Scenario A, except that the transmission range is increased to 200 m. The purpose of this was to increase the average neighbor count from 7.06 to 12.6 nodes. The increased range makes some of the links more stable as nodes take longer to move out of range of each other. The value of the average shortest path would certainly be less than 4 nodes while the average network partition would be less than 5%.

Five metrics were used to evaluate and compare the performance of the protocols and they are discussed below. The hello interval in OWL was set to 15 s

The mean and a 95% confidence interval were obtained. The simulation results for the four routing protocols in scenario A are summarized in the following graphs. In Scenario A, we vary the number of flows and the pause time and observe the results. In Scenario B, we use 20 CBR flows and the nodes have a pause time of 30s; the results are shown in Table 2.

6.2 Delivery ratio

Delivery ratio is the fraction of packets that arrive at the corresponding destination by the end of the simulation. The reasons packets are not delivered to the destination are dependent on the specific protocol. Data packets can be dropped at the source if they cannot find a route to the destination. Data packets in transit can be dropped upon link failure, especially if the protocol does not perform local route repair. In OWL, when a link fails, a node can try

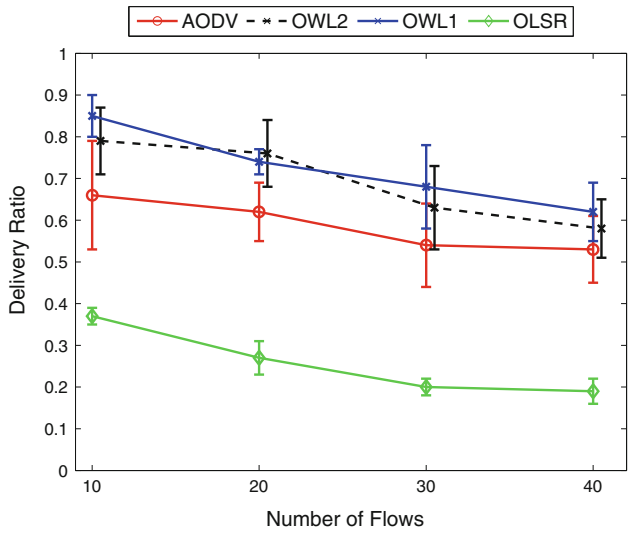

Fig. 4 Delivery ratio with varying flows

up to two different paths to send the data packet if such information is available. If no alternate routes are available the packet is dropped.

For Scenario A, the performance in terms of delivery ratio, as shown in Fig. 4, is poor for all protocols when compared to Scenario B (Table 2). This is due to the dynamic nature of the network, and the possibility of network partitions. Both versions of OWL have approximately the same delivery ratio (within the 95% confidence interval) and they perform better than AODV and OLSR under these conditions. As the number of flows is increased, the proportion of packets delivered decreases for all protocols as the network becomes more congested. The parameters used in OLSR are the ones suggested by its authors [21] and in fact OLSR, in these simulations, uses a much smaller hello interval than OWL and should have more up-to-date neighborhood information. However, under these rigid parameters OLSR's performance is noticeably worse than OWL and is indicative of transient loops under these conditions.

An important result is that under these conditions, as evident in Fig. 4, the DFS approach discovers paths from the source to the destination and delivers comparable, if not more, packets than the BFS approaches. What is surprising is that the single-path ordered walks perform marginally better than the dual-path ordered walk version of OWL. As the number of flows increases, the number of concurrent walks for different source-destination pairs increases. In a broadcast medium, increased overhead can degrade the performance of the network because of collisions. Ordered-walks, like any other broadcast packet, can disrupt data flow and give the impression of link failures resulting in new searches which can be associated with additional delays and dropped packets. As the results show, in these scenarios, a single walk is sufficient to discover a path in a timely manner and having multiple walks adds to the traffic of the network and this additional overhead is not justified when considering performance of networks that are at most a few hundreds nodes. However, as the size of the network and the number of neighbors increase, the search tree becomes deeper and wider, and multiple simultaneous walks may become more important. Also, as mobility increases, past location information can quickly become outdated and the inclusion of a walk that is not guided by past information can lead to faster route discovery. For larger networks, a hierarchical approach may be more suitable with each zone being restricted to a few hundred nodes. Nonetheless, for smaller networks more walks does not necessarily translate to better performance.

As the pause time is decreased, as shown in Fig. 5, the performance of the protocols tested improves. We notice that at low pause time, the performance of OLSR is much worse than the others, even though its hello interval is smaller than that of OWL. Once again, we see that both versions of OWL deliver almost the same fraction of packets.

By increasing the radio range (Scenario B), the nodes have more neighbors and the links are less volatile. Consequently, all protocols attain better delivery ratio than in Scenario A, but their relative performance remains the same as seen in Table 2.

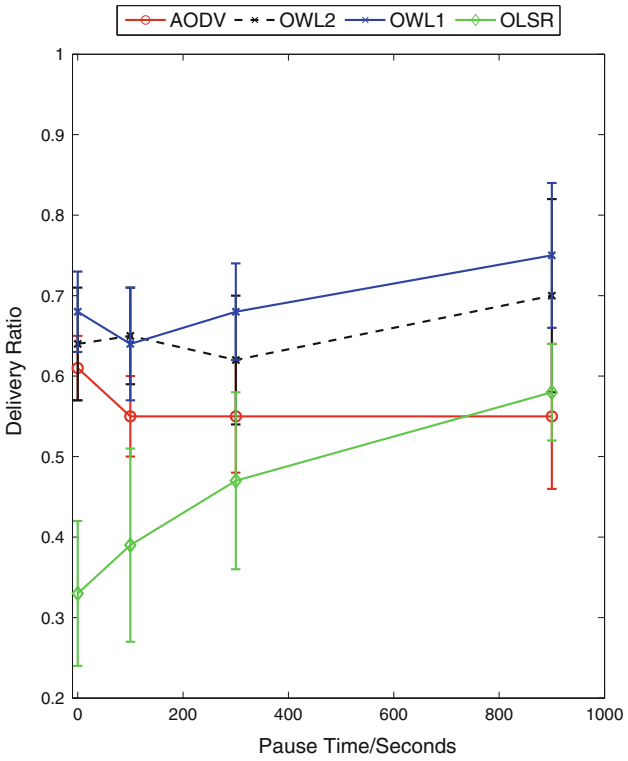

Fig. 5 Delivery ratio with varying pause time

6.3 Latency

Latency is the average time taken for a data packet to the destination node from the source node. Two main factors affect the latency: the time taken to discover a route and the frequency at which the protocol needs to rediscover a route to the destination.

For a proactive protocol like OLSR, this time is expected to be very small, because there is no delay involved in setting up a path when a data packet arrives at a node, given that one is already known. The time it takes to repair links also affects latency. In mobile networks, link failures are inevitable. Protocols should be able to repair links with minimal delay to ensure the timely delivery of data packets. This is usually done with local route repair, but local route repairs may not always succeed in obtaining new routes, and a failed local route repair incurs a greater delay penalty compared to sending an immediate route error (RERR) to the source upon link failure.

BFS would, at least according to intuition, lead to faster route discovery and faster route repair than DFS. However, in networks with multiple flows, the flooding of search packets can result in the broadcast storm problem and this results in lost packets which can lead to failed route discoveries. Furthermore, the broadcast-storm problem can disrupt flows in the network, which leads to an increased frequency of route discoveries and this increases the latency of packets.

With DFS, on the other hand, only a small fraction of the network is involved in route computations at any given time and the average time to discover a destination is longer than that incurred in BFS. However, there is significantly less overhead and consequently less disruptions of flows, therefore there is less need to re-establish paths to the destination and this results in reduced latency.

OWL makes up to two attempts at local route repair. When a local repair is successful, a RERR is not sent to the source. If the intermediate node does not already know alternate paths, or the alternate paths are broken, it sends a RERR to the source. Only nodes along the path of the RERR and those within radio range learn of the link failure encoded in the RERR. Therefore, the topology information stored in nodes is not uniform in OWL. However, for the purposes of routing RREQs, absolute accuracy is not necessary. A node is still likely to be in the same vicinity of its last known location, and this is sufficient for the purposes of guiding RREQs.

The variation of latency with number of flows is shown in Fig. 6. As the number of flows increases, there is more disruption of paths which results in greater latencies experienced by all protocols tested. For all the on-demand protocols tested, the latency is almost the same and dual-walk OWL does marginally better than single-walk OWL. With two walks, route discovery is usually quicker, but overall BFS does not result in superior latency, despite its intuitively faster route discovery process.

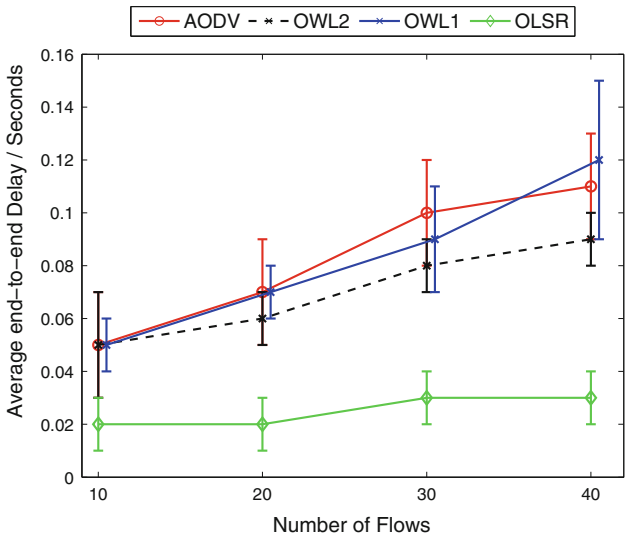

Fig. 6 Latency with varying flows

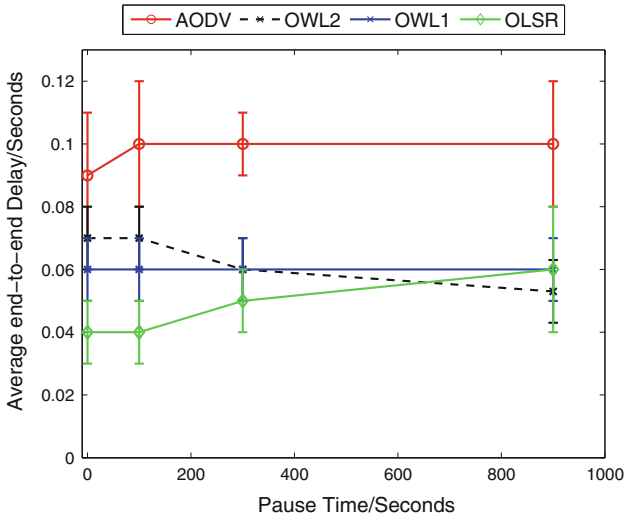

Fig. 7 Latency with varying pause time

The results for varied pause time are shown in Fig. 7. As the network becomes less mobile, more packets are delivered (Fig. 5) and there are more packets from which to derive the average latency, hence the slight increase in latency as the pause time is increased. For all pause times, the latency for single-walk and dual-walk OWL is almost the same and better than that of AODV, because the reduced overhead results in less disruption of flows.

6.4 Network load

Network load is the number of overhead packets (RREQs, RREPs, RERRs, Hellos, etc.) that were initiated or forwarded by nodes, normalized by the number of unique data

packets sent at each source node. This takes into account data packets that were sent into the network and were dropped or did not make it to the destination for any reason. Network load gives an indication of the average number of overhead packets needed to send a single new data packet from the source to the destination. The number of overhead packets needed would depend on the volatility of the network. The more frequently links are broken, the more control packets would be needed to establish new paths.

For proactive protocols like OLSR, the total number of overhead packets is constant regardless of the number of flows. Thus such protocols become more efficient in terms of overhead as the number of flows in the network increases.

For on-demand routing protocols, as the number of flows increases, the total number of control packets increases because each flow incurs overhead. The results in Fig. 8 reflect this. It is interesting to see that the overhead per data remains almost constant in OWL as the number of flows increases, whereas this metric increases in AODV. The flooding in AODV leads to more disruption of ongoing flows; hence, as the number of flows increases the number of overhead packets per data increases. In OWL, however, this is clearly not the case. Given that this metric is almost constant in OWL, it does not appear that the walks disrupt each other as much as network-wide floods do. There is a very small difference between both versions of OWL with respect to this metric. In OWL, RREQ packets contribute much less to the number of overhead packets than RREPs. This is because multiple nodes retransmit RREPs, whereas RREQs are propagated only by the node to which it was sent. Consequently, the cost of two ordered walks is not much more than that of one ordered walk (Fig. 9).

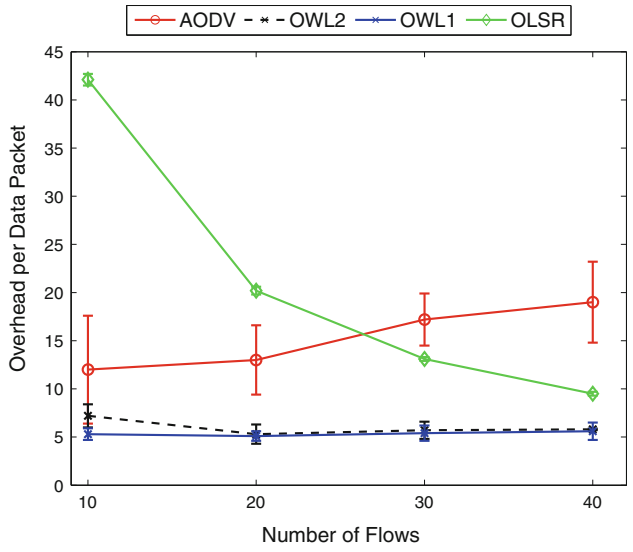

Fig. 8 Load with varying flows

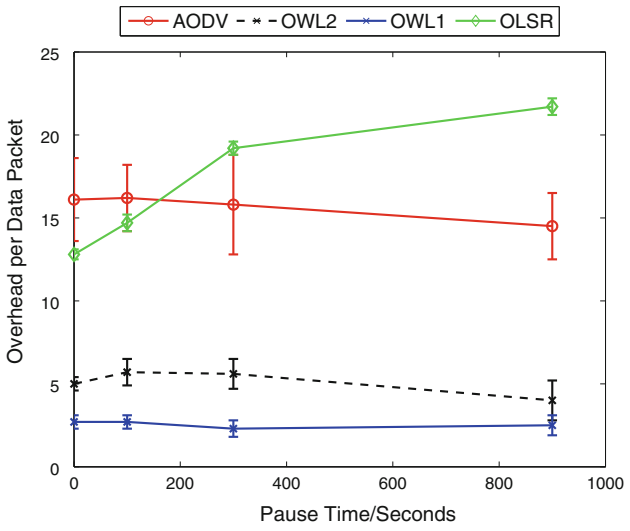

Fig. 9 Overhead with varying pause time

The advantage of ordered walks becomes quite clear when this metric is considered. OWL requires significantly fewer overhead packets than the other protocols in all scenarios tested. OWL efficiently searches the network without flooding, which allows sources to find their destinations without having to search every node.

6.5 Requests per reply

This metric is the average ratio of RREQs initiated and retransmitted per RREP initiated. It gives an idea of the efficiency of the search in that it indicates the number of nodes that must transmit a RREQ for it to be received by the destination. The fewer nodes that need to retransmit a RREQ the more efficient the search. This metric is not applicable to proactive routing protocols, as they do not use route requests or route replies.

Scenario A was designed such that the average shortest path distance between any pair of nodes would be about four hops [23]. This would mean that the minimum possible value should be four, the case in which the RREQ travels the exact shortest path from the source to the destination.

AODV uses an expanding ring search, and while it may not search the entire network to find the destination, it searches many more nodes than OWL does, as seen in Fig. 10. A successful search helps determine the diameter of the ring to be searched in subsequent flows. However, if the radius is five hops, this would account for a ring that is a significant fraction of the network.

In OWL, the first search is usually suboptimal; however, its design is aimed at making the search as efficient as possible with even no information about the destination. Once the destination is discovered, subsequent flows can be

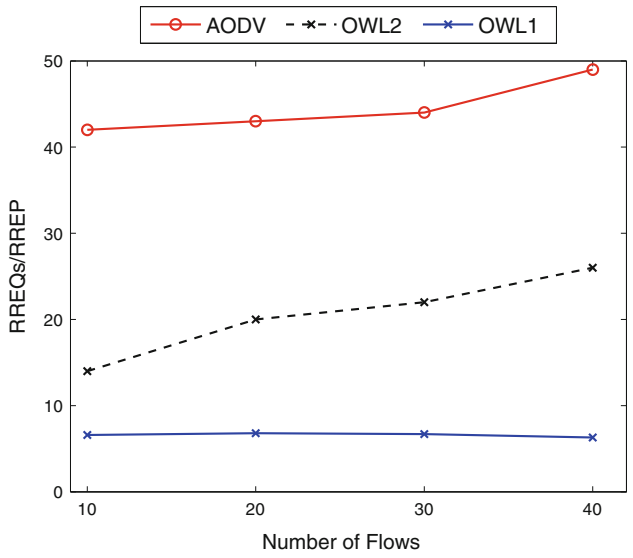

Fig. 10 RREQ per RREP with varying Flows

very efficient as long as the destination is in the same vicinity as its last location. From the results, single-walk OWL implements a much more efficient search than dual-walk OWL and AODV. The number of RREQ/RREP in dual-walk OWL (OWL2) is more than double that of single-walk OWL (OWL1) in Fig. 10. OWL2 uses an ordered walk which is independent of past information but it is possible that both walks intersect and their interaction can be to the detriment of the protocol.

In OWL, fewer nodes are involved in a route computation at any given time and this leads to less disruption of data flows already in process and route computations from other nodes and this attests to the scalability of the protocol.

Even in scenario B, AODV needs to flood half the network on average before the destination is discovered. In fact, dense networks where nodes have a large number of neighbors are more susceptible to flooding than those with fewer neighbors. The more neighbors a node has, the more times it will receive any particular flood packet and the less available bandwidth it has for data transmissions.

6.6 Towards scalability

OWL requires far fewer route requests and route replies to deliver a comparable number of packets than routing protocols based on traditional BFS searches. While the route discovery process may take marginally longer, the DFS approach used in OWL avoids the broadcast storm problem and causes fewer disruptions to other flows in the network. This becomes particularly important in larger networks as the number of flows increases. Considering all the results together, it becomes clear that ordered walks can be used to replace the flooding mechanism of reactive routing.

7 Conclusion

We argued that most routing schemes designed for MANETs to date rely on some form of BFS, and presented the ordered walk search algorithm (OSA) as a replacement for flooding. An ordered walk is a distributed approximation of DFS that is aided by known topology information to reduce the search space. We introduced the OWL routing protocol as an example of the great potential of using OSA in route signaling for MANETs. We presented the results of simulation experiments illustrating that OWL provides comparable or better delivery and end-to-end delays than AODV and OLSR, but with significantly less signaling overhead. The use of ordered walks, as presented in this paper, is a promising tool in achieving limited-signaling routing in MANETs. An interesting result is that increasing the number of simultaneous DFS does not significantly improve performance of the routing protocol.

Acknowledgments This research was partially sponsored by the U.S. Army Research Office (ARO) under grant W911NF-05-1-0246, by the U.S. Army Research Laboratory under the Network Science Collaborative Technology Alliance, Agreement Number W911NF-09-0053, and by the Baskin Chair of Computer Engineering. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

References

1. Awerbuch, B. (1985). A new distributed depth-first-search algorithm. *Information Processing Letters 20*(3), 147–150.
2. Cidon, I. (1988). Yet another distributed depth-first-search algorithm. *Information Processing Letters 26*(6), 301–305.
3. Makki, S., & Havas, G. (1994). Optimal distributed algorithms for constructing a depth first search tree. In *Proceedings of ICPP*.
4. Tian, H., Shen, H., & Matsuzawa, T. (2005). Randomwalk routing for wireless sensor networks. In *Proceedings of PDCAT*.
5. Servetto, S. D., & Barrenechea, G. (2002). Constrained random walks on random graphs: Routing algorithms for large scale wireless sensor networks. *Proceedings of WSNA*.
6. Karp, B., & Kung, H. (2000). Greedy perimeter stateless routing for wireless networks. In *Proceedings of the sixth annual ACM/IEEE international conference on mobile computing and networking*, pp. 243–254, August.
7. Acer, U., Kalyanaraman, S., & Abouzeid, A. A. (2007). Weak state routing for large dynamic networks. In *Proceedings of MobiCom*, September.
8. Caesar, M., Castro, M., Nightingale, E. B., O'Shea, G., & Rowstron, A. (2006). Virtual ring routing: Network routing inspired by dht. In *Proceedings of ACM SIGCOMM*.
9. Dabideen, S., & Garcia-Luna-Aceves, J. J. (2009). OWL: Towards scalable routing in manets using depth-first search on demand. *Sixth IEEE international conference on mobile ad-hoc and sensor systems*.
10. Ni, S.-Y., Tseng, Y.-C., Hen, Y. S. V., & Sheu, J.-P. (2001). The boradcast storm problem in mobile ad-hoc networks. In *Proceedings of MobiCom*.
11. Segall, A. (1983). Distributed network protocols. *IEEE Transactions on Information Theory*.

12. Spohn, M. A. (2005). *Domination in graphs in the context of mobile ad hoc networks*. Ph.D. dissertation, University of California, Santa Cruz.
13. Kleinrock, L., & Kamoun, F. (1977). Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks 1*(3), 155–174.
14. Guangyu Pei, M. G., & Chen, T.-W. (2000). Fisheye state routing in mobile ad hoc networks. *ICDCS workshop in wireless networks and mobile computing*.
15. Ramanathan, R., & Santivanez, C. (2001). Hazy sighted link state (hsls) routing: A scalable link state algorithm. *BBM Technical Memo BBN-TM-I30I, BBN Technologies*.
16. Xu, K., Hong, X., & Gerla, M. (2002). An ad hoc network with mobile backbones. *IEEE international conference on communications.*
17. Rubin, I., & Vincent, P. (2001). Topological synthesis of mobile backbone networks for managing ad hoc wireless networks. *IEEE international conference on management of multimedia networks and services: Management of multimedia on the internet*.
18. Beraldi, R., Querzoni, L., & Baldoni, R. (2006). A hint based probabilistic protocol for unicast communications in manets. *Elsevier Ad Hoc Networks, 4*(5).
19. Malkhi, D., Sen, S., Talwar, K., Werneck, R. F., & Wieder, U. (2009). Virtual ring routing trends. In *Proceedings of the 23rd international conference on distributed computing*.
20. Jacquet, P., Laouiti, A., Minet, P., & Viennot, L. (2001). *Performance analysis of olsr multipoint relay flooding in two ad hoc wireless network models*. INRIA, Technical report.
21. Clausen, T., & Jacquet, P. (2003). *Optimized link state routing protocol (OLSR)*. *RFC 3626*, October.
22. Perkins, C. E., & Royer, E. M. (2003). Ad hoc on-demand distance vector routing. *IETF Internet draft*, October.
23. Kurkowski, S., Camp, T., & Navidi, W. (2006). *Minimal standards for rigorous MANET routing protocol evaluation*. Technical report MCS 06-02, Colorado School of Mines.
24. Navidi, W., & Camp, T. (2006). Stationary distributions for the random waypoint mobility model. *IEEE Transactions on Mobile Computing 3*(1), 1153–1166.

Author Biographies

***Step*hen Dabideen** received his B.S degree in Computer and Telecommunications Engineering at the University of Pennsylvania in 2006. He is currently a Ph.D candidate at the University of California at Santa Cruz. He is a member of the Computer Communications Research Group lead by Prof. J. J. Garcia-Luna-Aceves and his research focus is on routing in wireless networks.

J. J. Garcia-Luna-Aceves received the B.S. degree in Electrical Engineering from the Universidad Iberoamericana, Mexico City, Mexico in 1977; and the M.S. and Ph.D. degrees in Electrical Engineering from the University of Hawaii at Manoa, Honolulu, HI in 1980 and 1983, respectively. He holds the Jack Baskin Endowed Chair of Computer Engineering at the University of California, Santa Cruz (UCSC), is Chair of the Computer Engineering Department, and is a Principal Scientist at the Palo Alto Research Center (PARC). Prior to joining UCSC in 1993, he was a Center Director at SRI International (SRI) in Menlo Park, California. He has been a Visiting Professor at Sun Laboratories and a Principal of Protocol Design at Nokia. Dr. Garcia-Luna-Aceves holds 35 U.S. patents, and has published three books and more than 400 journal and conference papers. He has directed 30 Ph.D. theses and 28 M.S. theses since he joined UCSC in 1993. He has been the General Chair of the ACM MobiCom 2008 Conference; the General Chair of the IEEE SECON 2005 Conference; Program Co-Chair of ACM MobiHoc 2002 and ACM MobiCom 2000; Chair of the ACM SIG Multimedia; General Chair of ACM Multimedia '93 and ACM SIGCOMM '88; and Program Chair of IEEE MULTIMEDIA '92, ACM SIGCOMM '87, and ACM SIGCOMM '86. He has served in the IEEE Internet Technology Award Committee, the IEEE Richard W. Hamming Medal Committee, and the National Research Council Panel on Digitization and Communications Science of the Army Research Laboratory Technical Assessment Board. He is an IEEE Fellow and an ACM Fellow, and is listed in Marquis Who's Who in America and Who's Who in The World. He is the co-recipient of the IEEE Fred W. Ellersick 2008 MILCOM Award for best unclassified paper. He is also co-recipient of Best Paper Awards at the European Wireless Conference 2010, IEEE MASS 2008, SPECTS 2007, IFIP Networking 2007, and IEEE MASS 2005 conferences, and of the Best Student Paper Award of the 1998 IEEE International Conference on Systems, Man, and Cybernetics. He received the SRI International Exceptional-Achievement Award in 1985 and 1989.