Caching Strategies in On-Demand Routing Protocols for Wireless Ad Hoc Networks

Yih-Chun Hu David B. Johnson

Computer Science Department
Carnegie Mellon University
Pittsburgh, PA 15213-3891 USA

http://www.monarch.cs.cmu.edu/
{yihchun,dbj}@cs.cmu.edu

Abstract

An *on-demand routing protocol* for wireless ad hoc networks is one that searches for and attempts to discover a route to some destination node only when a sending node originates a data packet addressed to that node. In order to avoid the need for such a route discovery to be performed before each data packet is sent, such routing protocols must cache routes previously discovered. This paper presents an analysis of the effects of different design choices for this caching in on-demand routing protocols in wireless ad hoc networks, dividing the problem into choices of cache *structure*, cache *capacity*, and cache *timeout*. Our analysis is based on the Dynamic Source Routing protocol (DSR), which operates entirely on-demand. Using detailed simulations of wireless ad hoc networks of 50 mobile nodes, we studied a large number of different caching algorithms that utilize a range of design choices, and simulated each cache primarily over a set of 50 different movement scenarios drawn from 5 different types of mobility models. We also define a set of new *mobility metrics* that allow accurate characterization of the relative difficulty that a given movement scenario presents to an ad hoc network routing protocol, and we analyze each mobility metric's ability to predict the actual difficulty in terms of routing overhead experienced by the routing protocol across the scenarios in our study.

1. Introduction

Caching is an important part of any on-demand routing protocol for wireless ad hoc networks. In an ad hoc network [10, 6], all nodes cooperate in order to dynamically establish and maintain routing in the network, forwarding packets for each other to allow communication between nodes not directly within wireless transmission range. Rather than using the periodic or background exchange of routing information common in most routing protocols, an *on-demand routing protocol* is one that searches for and attempts to discover a route to some destination node only when a sending node originates a data packet addressed to that node. In order to avoid the need for such a route discovery to be performed before each data packet is sent, an on-demand routing protocol must cache routes previously discovered. Such caching then introduces the problem of proper strategies for managing the structure and contents of this cache as nodes in the network move in and out of wireless transmission range of one another, possibly invalidating some cached routing information.

Several routing protocols for wireless ad hoc networks have used on-demand mechanisms, including TORA [14], DSR [9], AODV [15], ZRP [4], and LAR [11]. For example, in the Dynamic Source Routing protocol (DSR) [1, 8, 9] in its simplest form, when some node **S** originates a data packet destined for a node **D** to which **S** does not currently know a route, **S** initiates a new Route Discovery by beginning a controlled flood of a request packet through the network. When a copy of this request packet reaches either **D** or another node that has a cached route to **D**, this node then returns to **S** the route discovered by this request.

Performing such a Route Discovery can be an expensive operation, since it may cause a large number of request packets to be transmitted, and since it adds latency to the subsequent delivery of the data packet that initiated it, but this Route Discovery may also result in the collection of a large amount of information about the current state of the network that may be useful in future routing decisions. In particular, **S** may receive a number of replies in response to its Route Discovery flood, each of which returns information about a route to **D** through a different portion of the network; a node may also learn information about the state of the network by eavesdropping on the Route Discovery packets from other nodes. By caching and making effective use of this collected network state information, the amortized cost of Route Discoveries can be reduced and the overall performance of the network can be significantly improved.

In this paper, we analyze the effects of different design choices in caching strategies for on-demand routing protocols in wireless ad hoc networks. These design choices generally fall into three areas: cache *structure*, cache *capacity*, and cache *timeout*. For a wide range of different caching algorithms based on these design choices, we evaluate their effect on the ability of the routing protocol to successfully route packets to different destinations, the number of overhead packets generated by the routing protocol, the average latency required to deliver data packets, and the optimality of the routes used relative to the shortest route that physically existed at the time each packet was sent. Our evaluation in this paper is based on the caching behavior in the Dynamic Source Routing protocol (DSR) [1, 8, 9], although many of the results presented here we

This work was supported in part by the Air Force Materiel Command (AFMC) under DARPA contract number F19628-96-C-0061. Yih-Chun Hu was also supported by an NSF Graduate Fellowship. The views and conclusions contained here are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either express or implied, of AFMC, DARPA, NSF, Carnegie Mellon University, or the U.S. Government.

To appear in *Proceedings of the Sixth Annual International Conference on Mobile Computing and Networking* (MobiCom 2000), August 6–11, 2000, Boston, MA, USA.

believe can be generalized to apply to other on-demand wireless ad hoc network routing protocols as well.

To evaluate the effects of the different caching design choices, we performed a set of detailed simulations of wireless ad hoc networks of 50 mobile nodes, with realistic modeling of factors such as medium access control and contention, collisions, wireless signal strength and propagation delay, carrier sense, and capture effect [2], based on our extended version of the *ns-2* network simulator [3]. We simulated each of the caching algorithms primarily over a set of 50 different movement scenarios drawn from 5 different types of mobility models. To better characterize the relative difficulty that each movement scenario presents to the routing protocol, we utilize a set of *mobility metrics*, including the geometric metric presented by Johansson et al [7] and several improved metrics that we define here.

Section 2 of this paper gives an overview of the basic operation of the DSR protocol. In Section 3, we discuss the caching strategy design choices considered in the paper, and in Section 4, we describe the specific caching algorithms based on these choices that we used in our evaluation. In Section 5, we describe the methodology of our simulation study, including our simulator features, the performance metrics we evaluated, and the communication model we used. In Section 6, we define several mobility metrics and describe a number of mobility models, and we show the correlation between the mobility metrics of a scenario and the performance of DSR in that scenario. In Section 7, we present the cache performance results of this study, and in Section 8, we present conclusions.

2. Overview of the DSR Protocol

We use the Dynamic Source Routing protocol (DSR) [1, 8, 9] in this paper to illustrate the effects of different caching strategies in on-demand routing protocols, since DSR operates *entirely* on-demand and thus clearly shows the caching behavior. DSR is composed of two mechanisms that work together to allow the discovery and maintenance of source routes in the ad hoc network. *Route Discovery* is the mechanism by which a node **S** wishing to send a packet to a destination node **D** obtains a source route to **D**. Route Discovery is used only when **S** attempts to send a packet to **D** and does not already know a route to **D**. *Route Maintenance* is the mechanism by which node **S**, while using a source route to **D**, is able to detect when the network topology has changed such that it can no longer use its route to **D** because a link along the route no longer works. When Route Maintenance indicates a source route is broken, **S** can attempt to use any other route it happens to know to **D**, or can invoke Route Discovery again to find a new route for subsequent packets that it sends. Route Maintenance is used only when **S** is actually sending packets to **D**. This section describes the basic operation of Route Discovery and Route Maintenance, although a number of optimizations to this basic operation exist [1, 9] that are not discussed here due to space limitations.

To initiate a new Route Discovery for a node **D** (the target of the Route Discovery), **S** transmits a ROUTE REQUEST packet, which is received by other nodes located within direct wireless transmission range of **S**. Each node that receives the ROUTE REQUEST packet appends its own address to a record in the packet and rebroadcasts it to its neighbors, unless it has recently seen another copy of the ROUTE REQUEST for this Route Discovery or it finds that its address was already listed in the route record in the packet. The forwarding of the ROUTE REQUEST terminates when it reaches node **D**; this node then returns a ROUTE REPLY packet to **S**, giving a copy of the accumulated route record from the ROUTE REQUEST, indicating the path that the ROUTE REQUEST traveled to reach **D**. The forwarding of the ROUTE REQUEST also terminates when it reaches a node that has in its cache a route to **D**; this node then returns a ROUTE REPLY packet to **S**, giving the route as a concatenation of the accumulated route record from the ROUTE REQUEST together with this node's own cached route to **D**. The returned source route from the ROUTE REPLY is cached by **S** for use in sending subsequent data packets.

Route Maintenance is performed by each node that originates or forwards a data packet along a source route. Each such node is responsible for confirming that the packet has been received by the next hop along the source route given in the packet; the packet is retransmitted (up to a maximum number of attempts) until this confirmation of receipt is received. This confirmation may be provided at no cost to DSR, either as an existing standard part of the MAC protocol in use (such as the link-level acknowledgement frame defined by IEEE 802.11 [5]), or by a *passive acknowledgement* [10]. If neither of these confirmation mechanisms are available, the node transmitting the packet may set a bit in the packet header to request a DSR-specific software acknowledgement be returned by the next hop. If this confirmation is not received after some maximum number of local retransmission attempts, this node returns to the original sender of the packet a ROUTE ERROR message, identifying the link over which the packet could not be successfully transmitted. When receiving the ROUTE ERROR, this original sending node removes this broken link from its cache. In addition to returning a ROUTE ERROR message, this node may also attempt to *salvage* the original packet [2], if it has a route to the intended destination of the packet in its own cache. If so, the node replaces the original source route on the packet with the route from its cache and forwards the packet along that route; otherwise, the node discards the packet since no correct route is available.

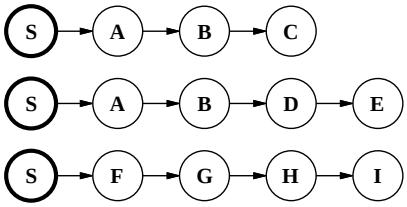
In response to a single Route Discovery, a node may learn and cache multiple routes to any destination. Nodes may also learn routing information from any packets that they forward or that they can overhear through optionally operating their network interface hardware in promiscuous mode; in particular, routing information may be learned from a ROUTE REQUEST, ROUTE REPLY, or ROUTE ERROR packet, or from the source route in the header of a data packet.

3. Caching Strategy Design Choices

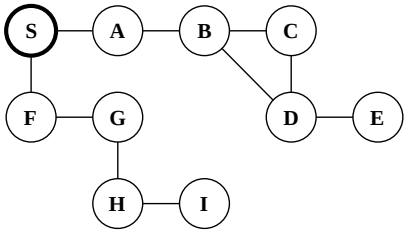
3.1. Cache Structure

In developing a caching strategy for an on-demand routing protocol for wireless ad hoc networks, one of the most fundamental design choices that must be made is the type of data structure used to represent the cache. In DSR, the route returned in each ROUTE REPLY that is received by the initiator of a Route Discovery represents a complete path (a sequence of links) leading from that node to the destination node. By caching each of these paths separately, a *path cache* can be formed; Figure 1(a) illustrates an example path cache for some node **S** in the ad hoc network. Alternatively, a *link cache* could be created, in which each individual link in the routes returned in ROUTE REPLY packets is added to a unified graph data structure of this node's current view of the network topology; Figure 1(b) illustrates an example link cache for node **S**.

A path cache is very simple to implement and easily guarantees that all routes are loop-free, since each individual route from a ROUTE REPLY is loop-free. To find a route in a path cache, the sending node can simply search its cache for any path (or prefix of a path) that leads to the intended destination node. On the other hand, to find a route in link cache, a node must use a much more complex



(a) Path Cache



(b) Link Cache

Figure 1 Alternative Cache Data Structures for a Node **S**

graph search algorithm, such as the well-known Dijkstra's shortest-path algorithm, to find the current best path through the graph to the destination node. Such an algorithm is more difficult to implement and may require significantly more CPU time to execute.

However, a path cache data structure cannot effectively utilize all of the potential information that a node might learn about the state of the network. In a link cache, links learned from different Route Discoveries or from the header of any overheard packets can be merged together to form new routes in the network, but this is not possible in a path cache due to the separation of each individual path in the cache. For example, if node **S** with the cache as shown in Figure 1(b) learns of a new link from node **A** to node **G**, it can use this link to also form new routes to nodes **H** and **I** (through **A** and **G**) that it could use if the link from **F** to **G** later breaks, but a node using a path cache would be unable to take advantage of these additional routes.

3.2. Cache Capacity

The capacity of a route cache is another important area of choice in designing a caching strategy for on-demand routing protocols. For a link cache, the logical choice is to allow the cache to store any links that are discovered, since there is a fixed maximum of N^2 links that may exist in an ad hoc network of N nodes. However, for a path cache, the maximum storage space that could be required is much larger, since each path is stored separately and there is no sharing in the data structure even when two paths share a number of common links. We thus consider the effects of different limits on the capacity of path caches in terms of the number of individual paths it can store. In general, our intuition was that the larger the capacity of a path cache, the better the routing protocol should perform, since it is able to keep a more complete set of routes. However, as we show in Section 7, a *smaller* cache size actually can have an indirect effect in improving performance.

An additional design choice with respect to cache capacity that we consider is the division of the cache into two halves: one half for paths that have been used by this node (the primary cache) and a second half for paths that have not yet been used since being learned (the secondary cache); when a path (or a prefix of a path) in the secondary cache is first used, that path (or prefix) is promoted to the primary cache. This division of the cache avoids forcing out of the cache paths that this node has found useful, when attempting to insert some new path into the cache that has just been learned and has not yet been used (and may never be used). Old paths in the secondary cache are removed due to capacity limits and the natural operation of the cache when adding new paths as they are learned, whereas old paths in the primary cache are more actively removed due to the operation of Route Maintenance as they are used. We refer to such a divided cache as a *generational* cache, in a manner similar to the way a generational garbage collector works in a language runtime system with dynamic storage allocation.

3.3. Cache Timeout

As with cache capacity, cache timeout policy introduces a number of design choices to consider in a caching strategy. Because a path cache generally has a mechanism for removing entries through a capacity limit, we did not implement a timeout for path caches. For link caches, the timeout on each link in the cache may be either static or adaptive.

For a static timeout, each link is removed from the cache after a specified amount of time has elapsed since the link was added to the cache. For an adaptive timeout, a node adding a link to its cache attempts to determine a suitable timeout after which the link will be deleted from the cache, and this timeout value should be based on properties of the link or the nodes that are the endpoints of the link. Finally, similar to the generational path caching alternative, it is possible to allow a link that is being used to not expire by increasing its timeout when it is used.

4. Caching Algorithms Studied

From the caching strategy design choices given in Section 3, we chose a collection of path caches and link caches to simulate and evaluate. We also simulated an "omniscient expiration" cache, which although unimplementable in a real system, gives us a benchmark against which our other cache algorithms can be compared.

4.1. Path Caches

Path caches store a set of complete paths (sequences of links), each starting at the caching node. We analyzed the following algorithms that use path caches:

- *Path-Inf* is a path cache with no capacity limit (infinite capacity).
- *Path-FIFO-64* is a path cache with a 64-path capacity limit. The cache replacement policy used on paths in the cache is FIFO.
- *Path-FIFO-32* is the same as *Path-FIFO-64*, except that it uses a 32-path capacity limit.
- *Path-Gen-64* is a generational path cache that employs a 30-element FIFO primary cache to store paths that have been used or were returned directly to this node in a ROUTE REPLY, and a separate 64-element FIFO secondary cache to store other paths; the total capacity of this cache is 94 elements.
- *Path-Gen-34* is the same as *Path-Gen-64*, except that the size of the secondary cache is 34-elements; the total capacity of this cache is 64 elements, the same as *Path-FIFO-64*. This specific caching algorithm, of this size, is the same as that used in our original *ns-2* simulation of DSR [2].

4.2. Link Caches

Link caches store a set of individual links, organized as a graph data structure. We analyzed the following algorithms that use link caches:

- *Link-NoExp* is a link cache with no timeout (no expiration).
- *Link-Static-5* is a link cache in which links normally are expired 5 seconds after they are put into the cache. This is a generational cache, such that links that are used to source packets sent by this node are marked to not timeout.
- *Link-Adapt-1.25* is a link cache in which a link's timeout is chosen according to a stability table. Each node keeps a table recording its perceived stability of each other node. When a link is used, the stability metric for both endpoint nodes is incremented by the amount of time since that link was last used, multiplied by some factor; when a link is observed to break, the stability metric for both endpoints is multiplicatively decreased by a different factor. A link entering the cache is given a lifetime equal to the stability of the less-"stable" endpoint of the link, except that a link is not allowed to be given a lifetime under 1 second. As with *Link-Static-5*, this is a generational cache, such that links that are used to source packets sent by this node are marked to not timeout. For this cache, the additive increase factor is 4, and the multiplicative decrease factor is 1.25. The stability table for each node is initialized to 25 seconds.
- *Link-Adapt-2* is the same as *Link-Adapt-1.25*, except that the multiplicative decrease factor is 2.
- *Link-MaxLife* is the same as *Link-Adapt-2*, except that when a node chooses a route from the cache, it chooses the shortest-length route that has the longest expected lifetime (highest minimum timeout of any link in the path), as opposed to an arbitrary route of shortest length.

4.3. Omniscient Expiration Cache

For comparison against the other caching algorithms that we studied, we also analyzed the following "omniscient expiration" caching algorithm:

- *Link-OmniExp* is a link cache that performs omniscient expiration of cached links, such that a link is removed from the cache exactly when it ceases to physically exist. The simulator has omniscient knowledge of the location of all nodes, and *Link-OmniExp* bases cache expiration on a nominal wireless transmission range for each link of 250 m.

5. Methodology

5.1. Simulator

We analyzed the effects the different caching strategy design choices through detailed simulation of the different caching algorithms described in Section 4. The experiments were conducting using the *ns-2* network simulator [3], which we have extended to support the simulation of wireless and mobile networks [2]. The simulator properly models signal strength, RF propagation, propagation delay, wireless medium contention, capture effect, interference, and arbitrary continuous node mobility. The radio model is based on the Lucent Technologies WaveLAN 802.11 product, providing a 2 Mbps transmission rate and a nominal transmission range of 250 m. The link layer modeled is the Distributed Coordination Function (DCF) of the IEEE 802.11 wireless LAN standard [5].

5.2. Communication Model Used

The communication model simulated in all scenarios was a script consisting of 20 Constant Bit Rate (CBR) data connections, each transmitting 4 packets per second; the size of each packet is 64 bytes. Each node was the source of at most 2 CBR connections.

5.3. DSR Performance Metrics

We evaluated the performance of DSR on each of the caching algorithms according to four metrics:

- *Packet Delivery Ratio*: The fraction of packets sent by the "application layer" on a source node that are received by the "application layer" on the corresponding destination node.
- *Overhead*: The total number of packets transmitted by the routing protocol. This includes routing packets forwarded, but does not include data packets forwarded.
- *Latency*: The delay from when a packet is sent by the "application layer" on a source node until it is received by the "application layer" on the corresponding destination node. This can only computed for packets that are successfully delivered.
- *Path Optimality*: The difference between the number of hops over which a packet was routed and the number of hops in the shortest route that physically existed when the packet was sent. The simulator is able to determine this theoretical shortest route at all times, based on the nominal wireless transmission range for each link of 250 m.

6. Mobility Models Studied

6.1. Mobility Metrics

The purpose of a mobility metric is to evaluate the relative difficulty of routing in a given ad hoc network scenario.

6.1.1. Geometric Mobility Metric

Johansson et al [7] describe a *geometric mobility metric* that is computed for a given scenario by

$$\frac{2}{N(N-1)T}\sum_{i=1}^{N}\sum_{j=i+1}^{N}\int_{t=0}^{T}\left|\frac{d\,\|P_j(t)-P_i(t)\|_2}{dt}\right|dt$$

where each $P_k(t)$ is the position of a node k at time t, N is the number of nodes, T is the length of the simulation, and the sum is calculated over all pairs of nodes over all time.

For the results in their paper [7], they approximated this metric by computing it with a 0.1-second time granularity and rearranged [12] the equation to compute

$$\sum_{i=1}^{N}\sum_{j=i+1}^{N}\int_{t=0}^{T}\left|\frac{d\,\|P_j(t)-P_i(t)\|_2}{dt}\right|dt \approx \frac{1}{2}\sum_{i=1}^{N}\sum_{t=0}^{10T}\left|\sum_{j=1}^{N}\left\|P_j\left(\frac{t}{10}\right)-P_i\left(\frac{t}{10}\right)\right\|_2 - \sum_{j=1}^{N}\left\|P_j\left(\frac{t-1}{10}\right)-P_i\left(\frac{t-1}{10}\right)\right\|_2\right|$$

This approximation, however, can lead to a very inaccurate calculation in some cases. For example, on scenarios generated as described in Section 6.2.1, the approximate mobility metric (with 0.1-second granularity) was too small by more than a factor of 2.2.

Instead, we used to following technique to calculate their geometric mobility metric precisely: split the integral so that each integral is along an interval in which there is no change in the velocity of either i or j. Define $f(t) = \|P_j(t) - P_i(t)\|_2$. We want $\int_{t_1}^{t_2}\left|\frac{df(t)}{dt}\right|dt$. If there is no relative velocity, then the integral is 0. If f has no local minima on $[t_1, t_2]$, then the integral evaluates to $\left|f(t)\right|_{t=t_1}^{t_2}\Big|$. Otherwise, if $t' \in [t_1, t_2]$ is a local minima of f, then the integral is $f(t_1) + f(t_2) - 2f(t')$.

6.1.2. Minimal Route-Change Metrics

A difficulty with the geometric mobility metric [7] is that it cannot distinguish between mobility that changes the network topology and mobility that instead has no effect on any links in the network. If we have information about the nominal wireless transmission range of the radios used in the network, we can more accurately determine how mobility affects the difficulty of routing.

The *minimal shortest route-change metric* for a pair of nodes i and j is the minimum number of times that i and j would need to change routes in order to always have a shortest (least hops) path to each other, assuming all links are bi-directional.

An alternate metric, that we call the *minimal route-change metric*, is the same as the minimal shortest route-change metric, except that a route counted by the metric only changes when it breaks, not when a shorter route begins to exist.

6.1.3. Communication Model-Dependant Metrics

The minimal shortest route-change metric and the minimal route-change metric both provide a number per pair of nodes; to arrive at a metric for a scenario, we can either sum over only those node pairs that communicate at least once during the scenario, or simply over all pairs of nodes regardless of communication behavior.

6.2. Mobility Model Specifications

We chose the parameters for our different mobility models to make the average speed of a node 10 m/s, and to keep the nodes as randomly distributed as the model would allow. All mobility models were generated for 50 nodes moving over a simulated time of 900 seconds, and all models confine the nodes to move within a 1500 m $\times$ 500 m space. Unless otherwise noted, the initial position of each node is chosen as (x_0, y_0), with x_0 uniformly distributed over $[0, 1500\text{ m}]$ and y_0 uniformly distributed over $[0, 500\text{ m}]$. Ten different scenarios were generated for each model.

6.2.1. Brownian Motion

Nodes in our Brownian motion mobility model change speed and direction at discrete time intervals, such that at the beginning of each interval, each node chooses $r \in [0, v_{max}]$ and $\theta \in (-\pi, \pi]$ and moves with velocity vector $(r \sin\theta, r \cos\theta)$ during that interval. If this movement would cause a node to end the interval beyond the boundaries of the rectangular area, the node instead picks the point within the rectangular area closest to the intended destination and moves to that point at the originally chosen velocity. The parameters used in our implementation of this model are given in Table I.

6.2.2. Column Motion

The column mobility model was developed by Sanchez [16]. In our implementation of this model, each node is either moving in the positive x direction or the negative x direction. The initial position of each node i is $(10i, 10i)$, and all nodes start moving in the positive x direction. The motion of the nodes is divided into discrete intervals, such that at the beginning of each interval, each node chooses $v \in [0, v_{max}]$ and moves with that speed in the same direction as it has been moving. If this movement would cause the node to cross the boundary of the rectangular area, the direction is instead flipped, and the node moves with speed v in the new direction rather than in the original direction. The parameters used in our implementation of this model are given in Table II.

Table I Parameters for Brownian Motion

Movement interval duration	0.1 s
v_{max}	20 m/s

Table II Parameters for Column Motion

Movement interval duration	0.1 s
v_{max}	20 m/s

6.2.3. Random Gauss-Markov Motion

The random Gauss-Markov mobility model was developed by Liang and Haas [13] and was described by Sanchez [17]. The motion of the nodes is divided into discrete time intervals, such that at the beginning of each interval, a node updates its velocity vector as

$$v_{x_t} = \alpha v_{x_{t-1}} + (1-\alpha)\overline{v_x} + R\sqrt{1-\alpha^2}$$
$$v_{y_t} = \alpha v_{y_{t-1}} + (1-\alpha)\overline{v_y} + R\sqrt{1-\alpha^2}$$

at interval t, where R is a normally distributed random variable with mean 0 and variance σ_{v_x}. When a movement would cause a node to exceed the boundaries of the rectangular area, the sign of the velocity vector in that dimension is flipped.

The parameters used in our implementation of this model are given in Table III. The choice of σ_{v_x} and σ_{v_y} was made to have the median of $\|(R_x, R_y)\|_2$ be equal to 10 m/s. The value of α was chosen to be equal to the value used by Sanchez in his implementation [18].

6.2.4. Random Waypoint Motion

The random waypoint mobility model was developed by Johnson and Maltz [9]. In this model, a node chooses a destination with a uniform random distribution over the area, moves there with velocity v uniformly distributed over $[0, v_{max}]$, waits there for a *pause time*, and then repeats this behavior. We used a pause time of 0, meaning continuous motion of all nodes, and chose $v_{max} = 20$ m/s. The parameters used in our implementation of this model are given in Table IV.

6.2.5. Pursue Motion

The pursue mobility model was developed by Sanchez [16]. In our implementation of this model, there are 10 groups of 5 nodes each. The motion of the nodes is divided into discrete time intervals, such that in each group, one node moves according to the random waypoint model, and the others attempt to "intercept" that node by choosing their velocity vector at each interval to be toward the point that the target node would be at at the end of the interval, given that the target node would continue to move with the same velocity. The velocity of the pursuing nodes is chosen uniform random for each interval to be in the range $[v_{pmin}, v_{pmax}]$. The parameters used in our implementation of this model are given in Table V.

Table III Parameters for Random Gauss-Markov Motion

Movement interval duration	0.1 s
Initial velocities	0 m/s
$\overline{v_x} = \overline{v_y}$	0 m/s
$\sigma_{v_x} = \sigma_{v_y}$	10.4835769 m/s
α	0.9

Table IV Parameters for Random Waypoint Motion

v_{max}	20 m/s
Pause time	0 s

Table V Parameters for Pursue Motion

Movement interval duration	0.1 s
v_{max}	20 m/s
v_{pmin}	5 m/s
v_{pmax}	15 m/s
Pause time	0 s

6.3. Evaluation of Mobility Metrics

We evaluated the mobility metrics described in Section 6.1 for each of the scenarios used throughout this paper. The geometric mobility metric was evaluated with infinite precision using the technique described in Section 6.1.1. The mobility metrics were normalized so that over all 50 scenarios, the metrics would lie in $[0, 1]$.

Figure 2 summarizes the degree to which the mobility metrics accurately characterize the difficulty of routing across the range of scenarios. Figure 2(a) shows the relationship between the normalized all-pairs geometric mobility metric for each scenario and the actual routing packet overhead generated by DSR on that scenario using the *Link-MaxLife* and *Path-Gen-34* caching algorithms. Also shown in Figure 2(a) is the best quadratic fit to the individual data points, in a least-squares sense, for these two caching algorithms. We show the results for these two caching algorithms here, since *Link-MaxLife* generally performs the best of the adaptive link caching algorithms, and *Path-Gen-34* is representative of the path caching algorithms. Figure 2(b) shows the similar relationship and type of quadratic fit for the number of ROUTE ERRORs originated during the simulation of these scenarios. Figures 2(c) and (d) show these relationships for the normalized all-pairs minimal route-change metric, and Figures 2(e) and (f) show these relationships for the normalized minimal route-change metric summed only over communicating pairs. Table VI shows the norm of residuals for the respective quadratic fit for each mobility metric, including also the all-pairs minimal shortest route-change metric, and the minimal shortest route-change metric summed only over communicating pairs.

The minimal shortest route-change metric does not reflect well the challenge presented to DSR, since DSR does not attempt to always switch to the shortest route when new routes begin to exist. Instead, DSR will continue to use its best route until it breaks or until it overhears a better route.

Table VI Norm of Residuals for Quadratic Fits of Routing Overhead and Number of ROUTE ERRORs

Path-Gen-34	Overhead	ERRORs
Geometric	120,248	9,189
Min Route-Change over All Pairs	111,699	5,973
Min Route-Change over Comm Pairs	77,144	2,877
Min Shortest Route-Change over All Pairs	168,729	10,799
Min Shortest Route-Change over Comm Pairs	160,027	9,782
Link-MaxLife		
Geometric	53,392	12,896
Min Route-Change over All Pairs	40,988	8,282
Min Route-Change over Comm Pairs	32,478	5,219
Min Shortest Route-Change over All Pairs	64,697	15,291
Min Shortest Route-Change over Comm Pairs	65,668	14,814
Link-OmniExp		
Geometric	18,963	116
Min Route-Change over All Pairs	17,616	105
Min Route-Change over Comm Pairs	17,885	106
Min Shortest Route-Change over All Pairs	19,988	116
Min Shortest Route-Change over Comm Pairs	23,093	122

As shown in Figure 2 and Table VI, the four minimal route-change metrics correlate significantly better, for both routing overhead and number of ROUTE ERROR packets, than does the geometric mobility metric, since route changes are a more direct cause of overhead and ROUTE ERRORs than is geometric mobility. Of the four minimal route-change metrics, the minimal route-change metric summed only over communicating pairs (Figure 2(e) and (f)) correlates best, since summing only among communicating pairs removes pairs which may undergo many route changes but that do not affect the routing algorithm. In addition, since the individual data points on the graphs relative to this metric are reasonably well spread and not tightly clustered, we conclude that the particular movement scenarios used in our study are generally representative of a fairly broad array of possible scenarios within the bounds used by these scenarios.

Although the four minimal route-change metrics correlate well to both the routing overhead and the number of ROUTE ERRORs, it correlates better for the number of ROUTE ERRORs. We believe this difference is due to the variable number of ROUTE REQUEST packets that may be sent as part of a Route Discovery, depending on the degree of containment of the ROUTE REQUEST flood that DSR is able to achieve for each individual Discovery attempt. We also examined the correlation of these metrics specifically to the number of Route Discoveries performed, and found fairly good correlation for *Path-Gen-34* but not for *Link-MaxLife*, which we attribute to the very small, statistically insignificant number of Route Discoveries needed by *Link-MaxLife*. Even a small change in number of Route Discoveries for any scenario with *Link-MaxLife* will result in a large relative change in the total, making correlation of any mobility metric difficult. We omit the detailed graphs and table here for number of Route Discoveries due to space constraints.

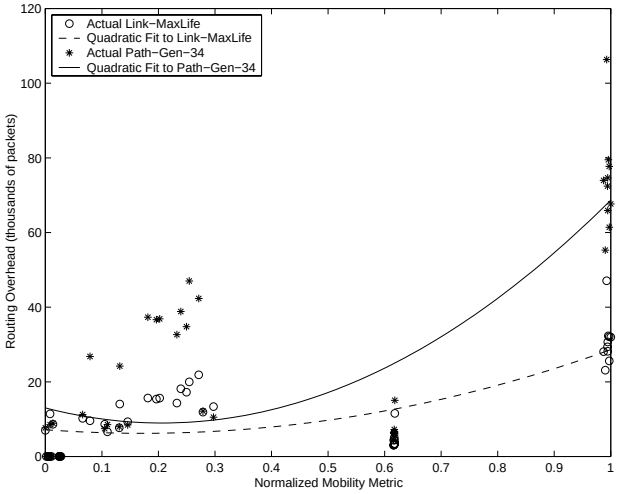
Another exception in the degree of correlation of the mobility metrics is those results obtained using the *Link-OmniExp* caching algorithm, for all of the performance indicators that we studied for the routing protocol. For all indicators, *Link-OmniExp* had relatively low correlation, since this caching algorithm creates very few Route Discoveries and even fewer ROUTE ERROR packets (and thus very small total routing overhead). As with the number of Route Discoveries needed by *Link-MaxLife*, as described above, none of the performance indicators that we studied for the routing protocol with *Link-OmniExp* are statistically significant.

7. Simulation Results

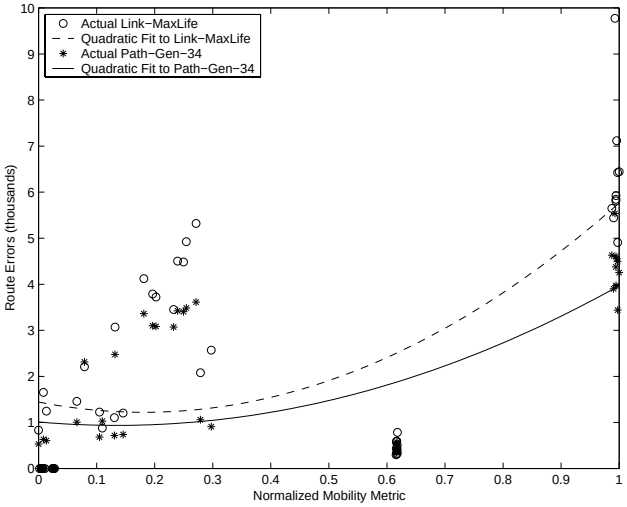
7.1. Overview of the Results

For each of the caching algorithms presented in Section 4, we ran 10 different scenarios of each of the mobility models described in Section 6.2. The scenarios were generated in advance, and the identical scenarios was used to evaluate each of the caching algorithms, allowing direct comparison of the results. Figure 3(a) shows the packet delivery ratio achieved by each caching algorithm, averaged over the 10 scenarios for each mobility model. Figure 3(b) shows the average routing packet overhead, Figure 3(c) shows the average packet delivery latency. Figure 3(d) shows the path optimality for each of the caching algorithms over the 10 scenarios from each mobility model, normalized and averaged over the 5 mobility models.

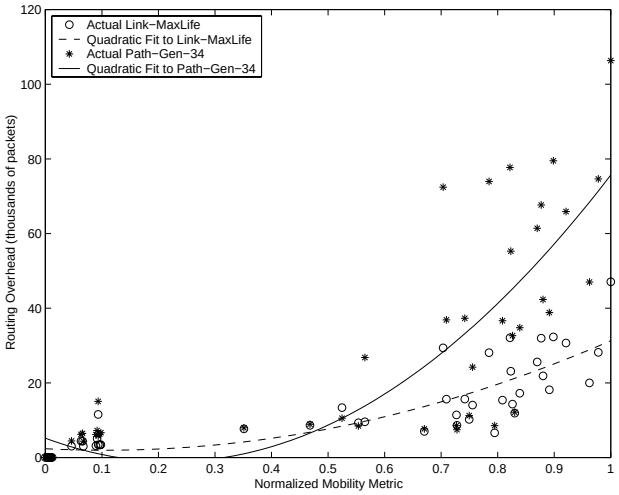
The *Link-Static-5* caching algorithm uses only a single fixed value for the cache timeout, although in general, no single timeout value



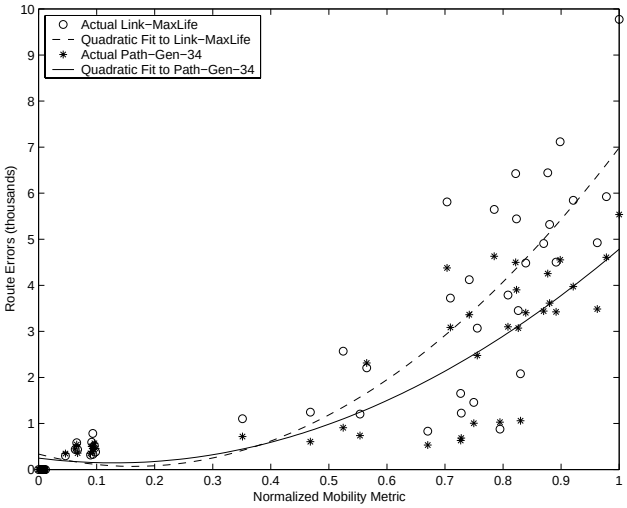
(a) Geometric Mobility Metric (Routing overhead)



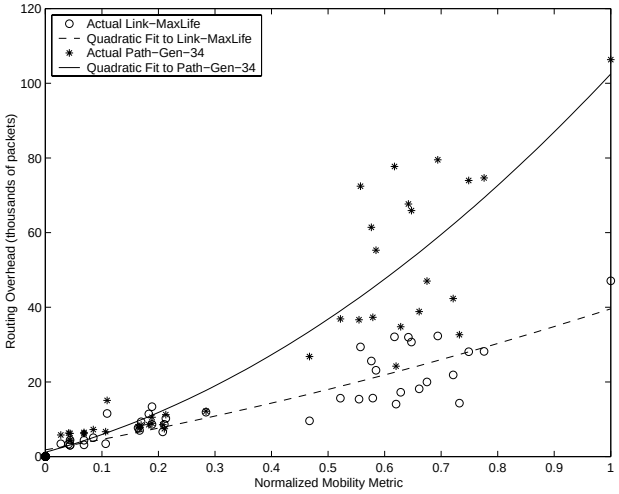
(b) Geometric Mobility Metric (ROUTE ERRORS)



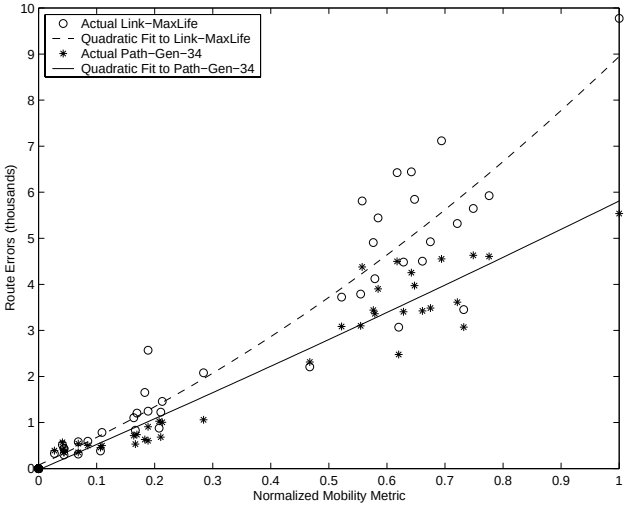
(c) Minimal Route-Change Metric, All Pairs of Nodes (Routing overhead)



(d) Minimal Route-Change Metric, All Pairs of Nodes (ROUTE ERRORS)

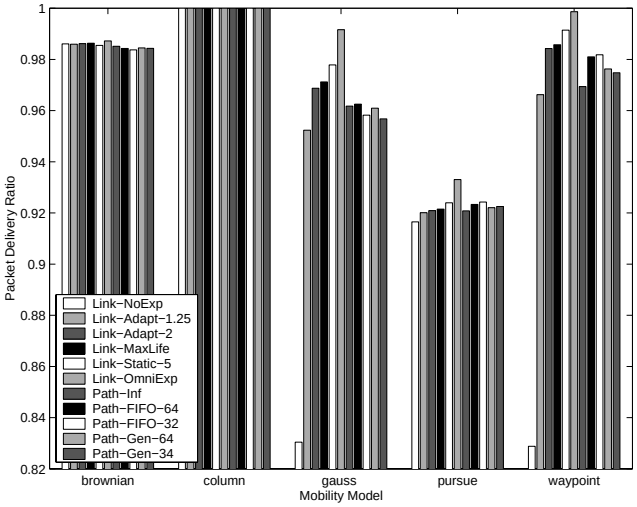


(e) Minimal Route-Change Metric, Communicating Pairs (Routing overhead)

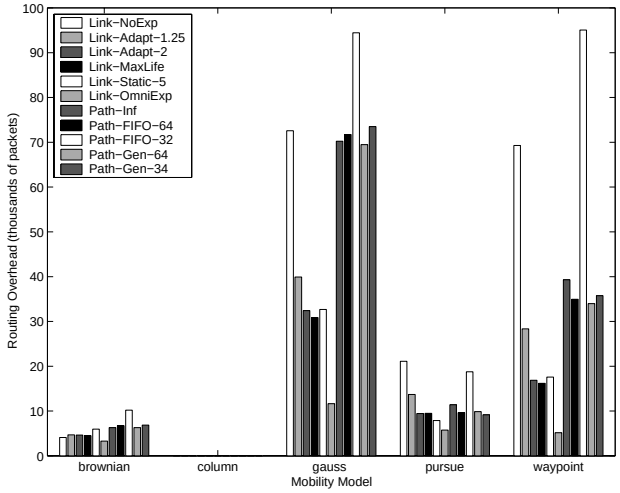


(f) Minimal Route-Change Metric, Communicating Pairs (ROUTE ERRORS)

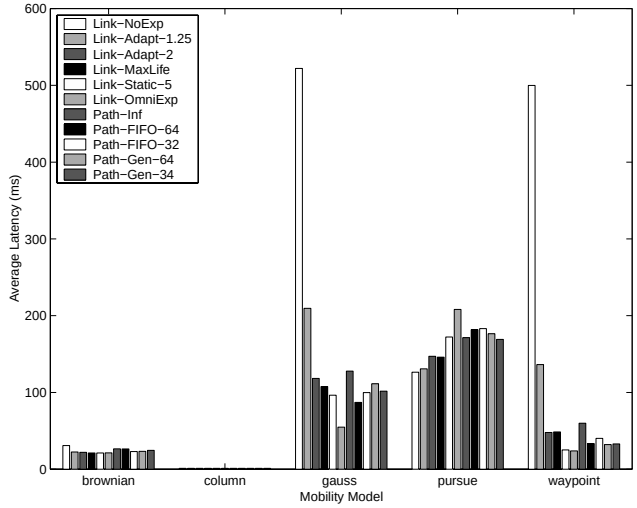
Figure 2 Correlation of Mobility Metrics to Routing Overhead and Number of ROUTE ERRORS



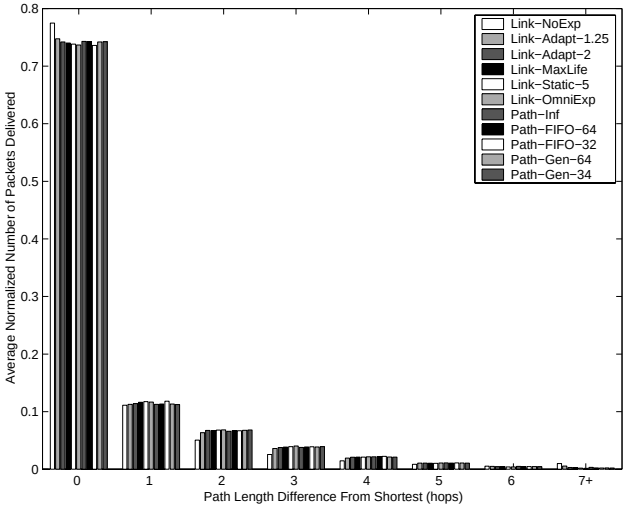
(a) Packet Delivery Ratio vs. Cache Type



(b) Routing Packet Overhead vs. Cache Type



(c) Latency vs. Cache Type



(d) Normalized Path Optimality vs. Cache Type

Figure 3 Performance of the Different Caching Algorithms on the Mobility Models

can perform best for all nodes in all ad hoc networks in all circumstances. In addition to the timeout value of 5 seconds shown in our results, we also evaluated a number of other timeouts ranging from 1 second to 40 seconds, and found that in our scenarios, the 5-second timeout performed best in terms of packet delivery ratio. As shown in Figure 3, our scenarios represent a range of different challenges for the routing protocol, but in each of our individual scenarios, all nodes in a given scenario move according to the same pattern. Thus, the advantage of an algorithm that can adapt to different timeouts for different links (between different pairs of nodes) was not fully exercised. We plan additional experiments in future work to explore this point, but here, we simply present the results for *Link-Static-5* and omit further comparison of them in this paper due to space constraints.

Although the column mobility model creates a large amount of motion among the nodes, there is very little *relative motion* among them and thus very little challenge to any of the caching algorithms. In fact, in each of our scenarios using the column mobility model, only 18 Route Discoveries were performed, regardless of the caching algorithm used. This property of the column model can also be seen using our new mobility metrics defined in Section 6.1; for example, the average geometric mobility metric over our column scenarios is 79.35% of the same metric over our random waypoint scenarios, appearing to indicate a comparable amount of mobility, yet when compared using our all-pairs minimal route-change mobility metric, this number drops to only 2.82%, clearly showing the much smaller challenge to the routing protocol.

In the pursue mobility model, the network remains partitioned much of the time; the 5 nodes in each group stay very close to each other, while the 10 separate groups are free to move over the entire simulation area, often leaving large, unoccupied spaces between the groups. For example, across all of our scenarios using the pursue mobility model, the network is partitioned on average 76.07% of the time. Due to this high occurrence of partition, the behavior of any caching algorithm used with the routing protocol will be very different than in more typical, usually connected networks, making comparison of different caching algorithms in these scenarios difficult.

In the remainder of this paper, we therefore focus in our analysis on only the scenarios using the Brownian, random Gauss-Markov, and random waypoint mobility models.

7.2. Effects of Cache Structure

For the packet delivery ratio metric, as shown in Figure 3(a), the *Link-Adapt-2* and *Link-MaxLife* link caching algorithms outperform all path caches, obtaining higher packet delivery ratio than the best path cache, evaluated individually for each mobility model. In most cases, *Link-MaxLife* performs somewhat better than *Link-Adapt-2* since it is able to select routes using links with the longest expected lifetime (based on the remaining cache timeout for each link) in addition to both algorithms' ability to select cache timeouts based on each node's stability metric.

Similarly, for the routing overhead metric, as shown in Figure 3(b), *Link-Adapt-2* and *Link-MaxLife* outperform all path caches, obtaining in most cases a reduction in overhead by a factor of about 2 or more over the best path cache for each mobility model. In addition, *Link-Adapt-1.25* performs better than the best path cache for each mobility model, although not by as much as do *Link-Adapt-2* and *Link-MaxLife*. This is consistent with the design intent of link caches over path caches, as link caches remove only a single link in response to a ROUTE ERROR (rather than removing a whole path or path suffix) and are able to combine information from different Route Discoveries to form new routes from the cached information.

In the scenarios that we studied, two primary factors contribute to the total latency experienced by a packet: the time spent by the packet waiting for a Route Discovery to complete before the packet can be sent, and the time spent in Route Maintenance detecting (through retransmissions) broken links and performing salvaging. For our Brownian motion scenarios, the dominant factor of these two is Route Discovery, which favors the link caches for low latency, since link caches generally perform fewer Discoveries than path caches, due to the increase in information that can be represented in the cache. For example, *Link-Adapt-1.25* (the *highest*-latency adaptive link cache) performs on average 431.5 fewer Route Discoveries than *Path-FIFO-32* (the *lowest*-latency path cache) in these scenarios, but it causes on average only 130.7 more ROUTE ERRORs. For the random Gauss-Markov and random waypoint scenarios, however, the number of ROUTE ERRORs becomes significant in the link caches, particularly for the *Link-NoExp* and *Link-Adapt-1.25*. For example, *Link-NoExp* and *Link-Adapt-1.25*, respectively, cause 31,117 and 10,652 ROUTE ERRORs, yet *Path-FIFO-32* (the highest-latency non-infinite path cache) causes only 1,973 ROUTE ERRORs.

All of the caching algorithms achieve good path optimality, and the differences between the results with different caching algorithms is small. In particular, the 5 path caching algorithms perform almost identically on most scenarios. However, for the link caching algorithms, path optimality differs for the *Link-NoExp* and *Link-Adapt-1.25* algorithms; these algorithms deliver a greater fraction of packets along optimal routes (path optimality 0) than do the other caching algorithms, yet also deliver a greater fraction of packets along routes 6 and 7 or more hops longer than optimal than do the other algorithms.

Both of these algorithms are able to keep a large number of unused links in the cache, as *Link-NoExp* never times out such links and *Link-Adapt-1.25* increases the node stability metrics (and thus the link cache lifetimes) much more aggressively than it decreases them. As such, these algorithms are able to opportunistically combine results from different Route Discoveries and from other routing information learned from packets forwarded or overhead, in order to more often find the shortest route that exists. However, the many unused links that these algorithms can keep in the cache also at times are a liability; many of these links may be broken, increasing the number of packets that must be salvaged multiple times, and thus increasing the total hop count for salvaged packets that are ultimately successfully delivered. In our simulations, each packet was prevented from being salvaged more than 15 times, in order to prevent the packet from possibly looping yet also allow alternate routing and backtracking of the packet in the presence of some stale cached links.

Overall, *Link-MaxLife* outperforms the other caching algorithms (excluding *Link-Static-5* and *Link-OmniExp*) on the set of performance metrics and scenarios studied. By taking advantage of the lifetime values in the route selection algorithm to differentiate between multiple routes of equal length, *Link-MaxLife* attempts to avoid using routes that may soon result in a ROUTE ERROR and a possible new Route Discovery. For example, in 26 of the 30 Brownian motion, random Gauss-Markov, and random waypoint scenarios, *Link-MaxLife* experiences fewer ROUTE ERRORs than *Link-Adapt-2*, where *Link-Adapt-2* is the same algorithm as *Link-MaxLife* without the use of lifetimes to aid in route selection.

7.3. Effects of Cache Capacity

In scenarios generated using the random waypoint mobility model, the *Path-Inf* caching algorithm, with its unlimited cache size, surprisingly performs much worse than the other path caches (with limited cache sizes) with respect to packet delivery ratio, as shown in Figure 3(a). This performance is due to the large number of ROUTE ERRORs caused by the use of stale routing information. When compared to *Path-FIFO-64*, *Path-Inf* experiences only 6.51% more ROUTE ERRORs in scenarios generated using our Brownian motion mobility model and 40.72% more ERRORs in our random Gauss-Markov scenarios, but experiences 165.58% more ERRORs in our waypoint scenarios.

Latency and routing packet overhead for the *Path-Inf* algorithm also suffer in the random waypoint scenarios, as shown in Figures 3(b) and (c). Sending a ROUTE ERROR typically counts as several packets of overhead since it must in general traverse several hops. In addition, when a packet is salvaged, the combined route traveled by the packet will typically be longer than the original route with which the packet was sent. When a packet must be salvaged multiple times, the resulting total routes can be quite long, causing significant increases in latency, and for each time a packet is salvaged, another ROUTE ERROR is returned to the original sender or previous salvager of the packet.

For the FIFO cache replacement policies studied here for path caches, no one cache size provides the best packet delivery ratio for all mobility models. For mobility models with large amounts of relative mobility, many Route Discoveries take place, causing a rapid turnover in each node's cache as it replaces existing cache entries with new entries learned from its own Route Discoveries or from other packets it has overheard. This cache replacement is in effect a form of adaptation in the caching algorithm, since as the amount of mobility in the network increases, the average number of broken routes created in the network increases and the average time that entries remain in a node's cache decreases with the cache turnover. However, with cache capacity as the limiting factor causing increased cache turnover, the FIFO caching algorithms have little control over *which* cache entry is replaced at which time. In

particular, in a different movement scenario with highly non-uniform behavior between different nodes, FIFO cache replacement would force the replacement of all paths (containing nodes with different behaviors) to be treated equally.

7.4. Effects of Cache Timeout

The use of a timeout on each cache entry in a link cache has a similar effect in cache replacement as the use of limited capacity has in a path cache, as described in Section 7.3. For example, the *Link-NoExp* algorithm, which has no timeout on cache entries, performs poorly with respect to packet delivery ratio for scenarios from the random Gauss-Markov and random waypoint mobility models, as shown in Figure 3(a). The movement in these models is generally quite dynamic, often resulting in the route that a node selects to use from its cache being broken even before the first packet is sent on it; this in turn causes the same type of ineffective salvaging and dropped packets as occurred when using the *Path-Inf* algorithm, with its unlimited path cache size.

The timeout value used on entries in the cache is closely related to a number of performance factors of the routing protocol. For example, if the timeout value is often too short, the number of ROUTE REQUESTs may increase, in order to rediscover links that were previously cached; if the timeout is often too long, the number of ROUTE ERRORs may increase, as more broken links are used from the cache. Similarly, the packet delivery ratio and routing overhead in a given scenario may increase or decrease, depending on the contents of the cache and the routing protocol's reaction to it.

In order to assess these relationships, we ran further simulations to collect results for static cache timeout values of 1, 2, 10, 20, and 40 seconds, in addition to the static timeout of 5 seconds of *Link-Static-5* and the infinite timeout (no expiration) cache of *Link-NoExp*. For each of these new static cache timeout values, we simulated each of our 10 random Gauss-Markov and 10 random waypoint scenarios. We present the results in Figure 4 for all 7 of these static timeout values for the random waypoint scenarios, although the results for the random Gauss-Markov scenarios are similar.

Figure 4(a) shows the relationship of the cache timeout value to the packet delivery ratio achieved during each of our random waypoint scenarios. Figure 4(b) shows the relationship to the routing overhead during each of these scenarios, and Figures 4(c) and (d), respectively, show the relationship to the number of ROUTE REQUESTs initiated and number of ROUTE ERRORs generated during each scenario. For each timeout value, in order to show the individual point representing each of the scenarios more clearly, the location of the points along the *x*-axis have been spread uniformly over the axis to the left and right of the specific timeout value, and the set of points for each individual timeout value have been colored alternately black or white. The scenarios within each timeout value in the graphs are ordered arbitrarily, in the order originally generated.

For cache timeout values of 20 seconds or less, as shown in Figure 4(a), the routing protocol was able to achieve between 98.8% (the minimum in this range, at timeout 20) and 99.1% (the maximum in this range, at timeout 5) packet delivery ratio; however, beginning at a timeout of 40 seconds, the packet delivery ratio falls sharply due to the large number of broken links that are allowed to accumulate in the route caches. This large number of broken links can also be seen in the rise in number of ROUTE ERRORs beginning at about a timeout of 10 seconds, and rising sharply above 20 seconds, as shown in Figure 4(d).

The routing overhead of the protocol is affected by the choice of cache timeout much more than is the packet delivery ratio. As shown in Figure 4(b), the routing overhead reaches a low of 15,423 routing packets in a single run at a timeout of 10 seconds (averaged over all of the scenarios with the same timeout); the overhead rises gradually to an average 25,482 packets at a timeout value of 1, and rises rapidly to an average 69,294 for the *Link-NoExp* caching algorithm with no cache timeout. This rapid rise in routing overhead with increasing cache timeout value is due to the corresponding rise in number of ROUTE ERRORs, as shown in Figure 4(d), caused by the increased accumulation of broken links in the caches.

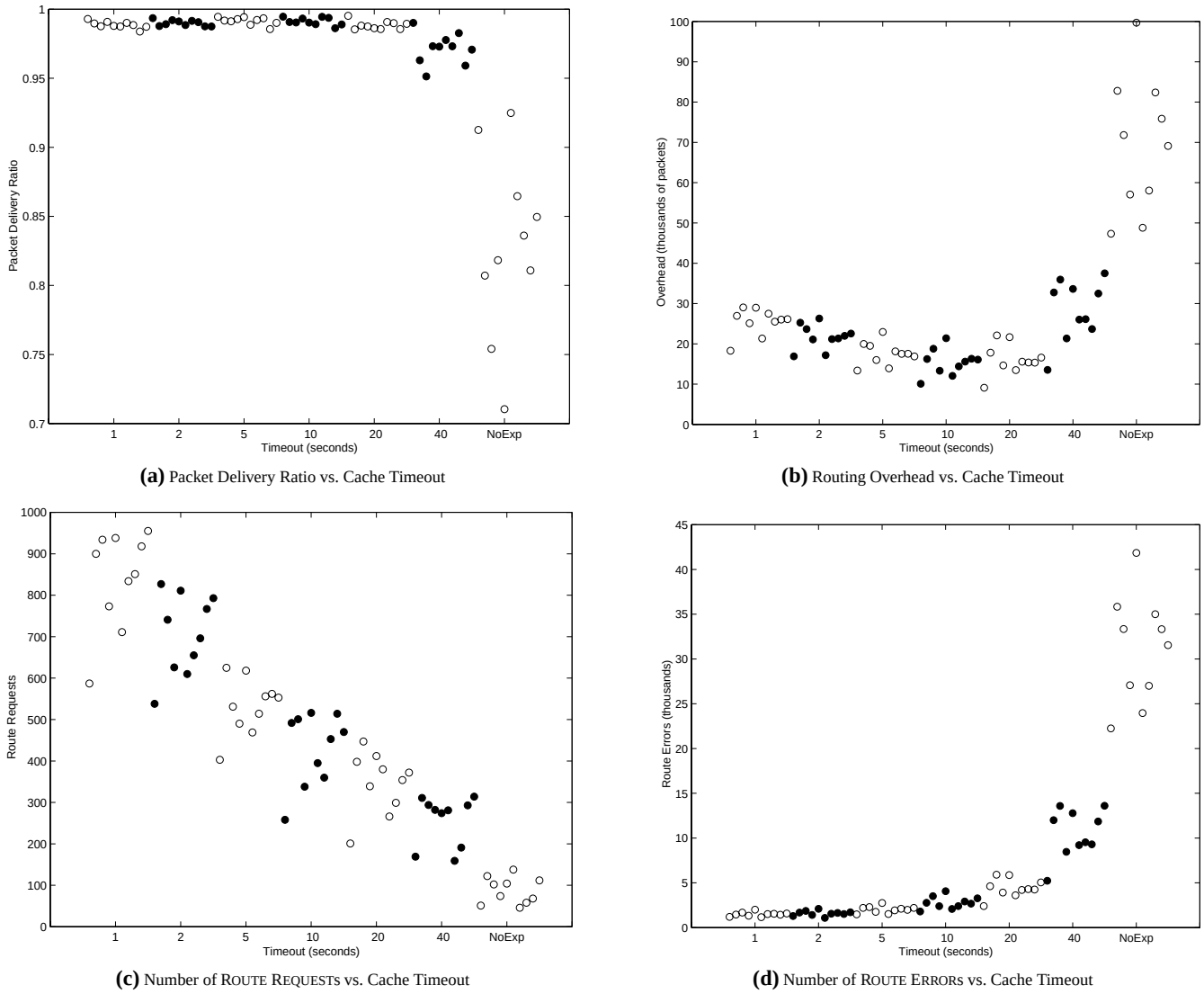
The rise in routing overhead with lower cache timeout values, below a timeout value of 10, is due to the rise in ROUTE REQUEST packets, as shown in Figure 4(c). Rather than allowing broken links to remain in the caches, such short timeout values in these scenarios often delete a link from the cache while it is still valid and still needed. The packet delivery ratio achieved decreases only slightly due to this rise in routing overhead with lower cache timeouts, as noted above, indicating the success of the routing protocol in being able to quickly rediscover and re-cache routes needed for the data packets being sent.

Based on these simulations across a range of static timeout values, it appears that either 5 or 10 seconds may be the best static value on these scenarios. A 5-second cache timeout results in the highest packet delivery ratio, but is only slightly higher than at 10 seconds; conversely, a 10-second cache timeout results in the lowest routing overhead, but is only slightly lower than at 5 seconds. At different node movement speeds or with different wireless transmission ranges, however, the optimal static timeout on these same scenarios would be different. We did not include our adaptive cache timeout algorithms in this analysis, since in these algorithms, each link receives a different timeout, and these timeouts vary over the life of the scenario, whereas the metrics that we relate to cache timeout value in Figure 4 are aggregate measures that reflect the overall performance of the entire scenario. Based on the performance metrics shown in Figure 3, however, the performance of our *Link-MaxLife* algorithm is very close to that of the *Link-Static-5* algorithm, the best performing static timeout caching algorithm.

8. Conclusions

A number of on-demand routing protocols for wireless ad hoc networks have been proposed, including TORA [14], DSR [1, 8, 9], AODV [15], ZRP [4], and LAR [11], and earlier detailed simulation work has shown that such protocols can have excellent performance [2, 7]. One key to achieving this type of performance is the design of an appropriate caching strategy for the protocol, that can make effective use of the state information about the network collected by the protocol as part of the process of discovering routes to other nodes. Caching is important in order to avoid the overhead of discovering a new route before sending each data packet, but caching also brings with it the risk and associated expenses of retaining routing information in a cache after the information is no longer valid due to changes in different nodes' positions or changes in the wireless propagation environment.

This paper has presented an analysis of the effects of different design choices in caching strategies for on-demand routing protocols in wireless ad hoc networks, dividing the problem into choices of cache *structure*, cache *capacity*, and cache *timeout*. Our analysis is based on the Dynamic Source Routing protocol (DSR) [1, 8, 9], which operates entirely on-demand. Using detailed simulations of



(a) Packet Delivery Ratio vs. Cache Timeout

(b) Routing Overhead vs. Cache Timeout

(c) Number of ROUTE REQUESTS vs. Cache Timeout

(d) Number of ROUTE ERRORS vs. Cache Timeout

Figure 4 Performance as a Function of Cache Timeout (Random Waypoint Scenarios)

wireless ad hoc networks of 50 mobile nodes, we studied a large number of different caching algorithms that utilize a range of caching strategy design choices, and simulated each cache primarily over a set of 50 different movement scenarios drawn from 5 different types of mobility models. Our evaluations include the packet delivery ratio, routing packet overhead, packet delivery latency, and path optimality relative to the shortest path, achieved by each caching algorithm.

We found that the performance of adaptive caches is comparable to that of well-tuned static caches, and that by utilizing a cache data structure based on a graph representation of individual links, rather than based on complete paths through the network, the routing protocol was much better able to make use of the potential information available to it; for example, several of our link caching algorithms were able to achieve about a factor of 2 less routing overhead than our best path caches on many scenarios. In addition, we identified some subtle relationships between cache timeout policies and cache capacity limits, and between these choices and some performance metrics for the routing protocol, most notably the packet delivery ratio and the routing packet overhead caused by the routing protocol. Somewhat unexpectedly, we also found a strong indication that caches of unlimited capacity or with no cache timeout perform substantially *worse* than caches with reasonable capacity or timeout limits.

This paper also contributes to the emerging definition and analysis of *mobility metrics* designed to allow a characterization of the relative difficulty that a given movement scenario presents to an ad hoc network routing protocol. We improve on the geometric mobility metric defined by Johansson et al [7] and define a set of new mobility metrics that much more accurately characterizes the important mobility in the system that may affect the routing protocol.

Acknowledgements

We would like to thank Miguel Sanchez and Ben Liang, for making their mobility models available, and Tony Larsson, for sending us the source code used by Johansson et al [7] in calculating their geometric mobility metric. We would also like to thank the anonymous reviewers, whose comments and suggestions helped to improve the presentation of the paper.

References

[1] Josh Broch, David B. Johnson, and David A. Maltz. The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks. Internet-Draft, draft-ietf-manet-dsr-03.txt, October 1999. Work in progress.

[2] Josh Broch, Dave Maltz, Dave Johnson, Yih-Chun Hu, and Jorjeta Jetcheva. A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols. In *Proceedings of the Fourth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 85–97, October 1998.

[3] Kevin Fall and Kannan Varadhan, editors. *ns* Notes and Documentation. The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC, November 1997. Available from http://www-mash.cs.berkeley.edu/ns/.

[4] Zygmunt J. Haas. A Routing Protocol for the Reconfigurable Wireless Network. In *1997 IEEE 6th International Conference on Universal Person Communications Record. Bridging the Way to the 21st Century, ICUPC '97*, volume 2, pages 562–566, October 1997.

[5] IEEE Computer Society LAN MAN Standards Committee. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*, IEEE Std 802.11-1997. The Institute of Electrical and Electronics Engineers, New York, New York, 1997.

[6] Internet Engineering Task Force MANET Working Group. Mobile Ad-hoc Networks (manet) Charter. Available at http://www.ietf.org/html.charters/manet-charter.html.

[7] Per Johansson, Tony Larsson, Nicklas Hedman, Bartosz Mielczarek, and Mikael Degermark. Scenario-based Performance Analysis of Routing Protocols for Mobile Ad-hoc Networks. In *Proceedings of the Fifth Annual ACM/IEEE International Conference on Mobile Computing and Networking*, pages 195–206, August 1999.

[8] David B. Johnson. Routing in Ad Hoc Networks of Mobile Hosts. In *Proceedings of the IEEE Workshop on Mobile Computing Systems and Applications*, pages 158–163, December 1994.

[9] David B. Johnson and David A. Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In *Mobile Computing*, edited by Tomasz Imielinski and Hank Korth, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[10] John Jubin and Janet D. Tornow. The DARPA Packet Radio Network Protocols. *Proceedings of the IEEE*, 75(1):21–32, January 1987.

[11] Young-Bae Ko and Nitin Vaidya. Location-Aided Routing (LAR) in Mobile Ad Hoc Networks. In *Proceedings of the Fourth International Conference on Mobile Computing and Networking* (MobiCom'98), pages 66–75, October 1998.

[12] Tony Larsson. Personal communication, February 8, 2000.

[13] Ben Liang. Personal communication, February 4, 2000.

[14] Vincent D. Park and M. Scott Corson. A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks. In *Proceedings of INFOCOM'97*, pages 1405–1413, April 1997.

[15] Charles E. Perkins and Elizabeth M. Royer. Ad-Hoc On-Demand Distance Vector Routing. In *Second IEEE Workshop on Mobile Computing Systems and Applications*, pages 90–100, February 1999.

[16] Miguel Sanchez. RE: Mobility pattern in a MANET, June 25, 1998. IETF MANET Mailing List, Message-ID: <000a01bda055$d84f9380$11352a9e@msanchez.disca.upv.es>.

[17] Miguel Sanchez. Re: Node Movement Models in Ad hoc, July 15, 1999. IETF MANET Mailing List, Message-ID: <378DC8F6.B01CF351@disca.upv.es>.

[18] Miguel Sanchez. Personal communication, February 1, 2000.