Improving Route-Freshness

Raj Kumar Rajendran, Vishal Misra, Dan Rubenstein
Columbia University, New York, NY
Email: raj@ee.columbia.edu,{misra,danr}@cs.columbia.edu

***Abstract*—Ad-hoc routing protocols in mobile environments face a fundamental tradeoff between delay and communication overhead. Since routes become stale as nodes move, nodes need to frequently recompute routes, leading to an increase in control traffic overhead. In dealing with this tradeoff, protocols often adopt a fixed stance: some protocols assume frequent node movement, regularly recompute routes and incur large bandwidth overheads while others assume a stable environment, recompute routes only when existing routes fail, and suffer large delays when nodes move.**

In this work we demonstrate how reactive protocols can improve the freshness of their routes while maintaining control bandwidth by dynamically estimating route-staleness. We show that nodes can use a technique called Strong Detection to analyze their routes for inconsistencies that reveal staleness. We then provide algorithms to selectively refresh routes until the inconsistencies that reveal staleness disappear.

We apply this Strong Detection technique to the AODV and DSR protocol and experimentally demonstrate that the freshness of AODV routes can be improved by approximately 30%. For the DSR protocol we show that the validity of routes can be improved by around 5% and delays due to unnecessarily long routes reduced from between 10% to 15% without incurring additional control overhead.

I. Introduction

Routing protocols in mobile ad-hoc network environments face fundamental tradeoffs between delay and overhead. If they are *proactive* and compute routes to nodes before data is routed to them, there is no route-computation delay but they continuously use a large portion of the network capacity to keep the routing information current. If there is significant node movement, topology change may be more frequent than route requests and most of the computed route information may never be used wasting network capacity. To reduce the route-computation overhead, some protocols are *reactive* and only compute a route to a node when data needs to be sent to it. In these protocols, when a route is needed, a global search (e.g., flooding) is used. This implies that the first time data needs to be routed to a node, there is an additional route-computation delay seriously impairing interactive sessions and situations where data is exchanged sporadically between mobile nodes and not applicable to real-time communication. Furthermore, the global search procedure employed causes significant traffic.

Among reactive routing algorithms different protocols take different positions with regard to route-freshness. Some algorithms like AODV [?] periodically purge routes that have not been used for a preset *route-timeout* interval ensuring that in a mobile environment, old routes that are likely to be invalid are not used. It also avoids the large delay incurred when data is sent over a stale route, and the error-recovery mechanism needs to be invoked to discover a new route. However this policy implies that a route-computation delay is incurred whenever data needs to be routed to a node after a interval of time larger than the preset route-timeout interval. Additionally frequent recomputation of routes is wasteful when node-movement is relatively small. Because of these problems other reactive routing algorithms such as DSR [?], take the position that the additional delay incurred in attempting to send data over an invalid route is an acceptable alternative to the overhead incurred in regularly recomputing routes. Therefore when new data needs to be routed, the last known route is used. Only if the data fails to reach is a new route computed. Such an approach is therefore more suited to an environment where node-movement is relatively small. However, in addition to the delay incurred in detecting undelivered data and recomputing routes this approach suffers from additional delay due to non-optimal routes. Since routes are not recomputed unless they are broken, a new shorter route, even if available, will never be discovered.

However all reactive algorithms, whether they purge routes or not, suffer in environments where nodes are quiescent at certain times and mobile at others. Therefore mechanisms that allows these reactive routing algorithms to adapt to changing mobility will render them more useful. Our work contributes exactly this: it allows protocols to first *identify* if their routes are fresh. Then, if staleness is detected, it efficiently *reduces* staleness by *selectively refreshing routes* thus reducing control overhead. Specifically, our contributes the following:

- It shows how route-staleness can be *identified*. It modifies the notion of Strong Detection (SD) for static topologies proposed in [?] to detect stale routes in mobile ad-hoc networks. Experiments show that this modified SD detects route-staleness more than 70% of the time.
- It provides algorithms to efficiently *reduce* route staleness. It shows that the small sub-set of routes that likely caused staleness can be identified. Refreshing *only* these routes reduces control overhead.
- It shows how staleness-identification and staleness-reduction can be performed for the AODV and DSR ad-hoc routing protocols.
- It provides experimental evidence that the freshness of AODV routes can be improved by around 30% and the validity of DSR routes by around 5 to 10%. It additionally shows how delays due to unnecessarily long routes can be reduced by 10 to 15% in DSR.

When the staleness identification and reduction technique (SIR) runs, it incurs computational and bandwidth costs. It has a running time of $O(|N|^3)$ and $O(|N|)$ for AODV and DSR respectively, where $|N|$ is the number of nodes to which a host has routes. Since AODV constantly purges routes, its $|N|$ tends to be small keeping the computational cost low. The SIR technique for DSR requires no additional bandwidth, but the SDF technique for AODV does use additional bandwidth when fixing stale routes. Our experiments ensure that the additional bandwidth used compares favorably with other means of improving route-freshness.

While our work shows how SD can be modified to detect and fix stale routes for the distance-vector based AODV and source-routing based DSR protocols, it can easily be modified with little effort for other ad-hoc protocols based on the distance-vector and source-routing protocols. SD for link-state routing is relatively simple, and therefore can modified for link-state based ad-hoc protocols with little effort.

A. Prior Work

Examples of proactive protocols include the Destination-Sequenced Distance Vector (DSDV) protocol [?], Wireless Routing Protocol (WRP) [?], Temporally-Ordered Routing Algorithm (TORA) [?], and Lightweight Mobile Routing (LMR) protocol [?]. Examples of on-demand protocols include the Ad hoc On Demand Distance Vector (AODV) protocol [?] and Dynamic Source Routing (DSR) protocol [?], [?].

The performance of the popular MANET routing protocols AODV, DSR, DSDV and TORA has been extensively simulated and studied [?], [?], [?], [?], [?]. The authors in [?] and [?] show that AODV suffers much smaller delays than DSR but sends out as much as six times more control packets in very mobile environments. However DSR's control packets are larger due to its use of source-routes. Many variations of the popular routing protocols that enhance aspects of their performance have been proposed [?], [?], [?]. The work of [?] adds proactive route discovery and maintenance to AODV and DSR. When the transmission power of a link approaches the breaking threshold, information of an impending break is sent out and discovery of a new route initiated. The work states that the performance of both AODV and DSR can be improved with a small increase in routing overhead. The work of [?] presents a pre-emptive on-demand routing protocol based on the distance-vector algorithm. They also conclude that it is feasible to use an estimate of an impending link break to schedule route-rediscovery or alternate routing, and that such pre-emptiveness improves route-delivery and delay. The authors of [?] use multipath routing to improve the performance of DSR. However we are aware on no works that analyze the consistency of routes to assess route-freshness.

The idea of Strong Detection proposed in [?], [?] was among the first techniques that provided a method for detecting static inconsistencies while being grounded in a theoretical framework. However Strong Detection, to date, has been only applied to detecting errors in static networks.

The rest of the paper is organized as follows. We first show how staleness is identified in Sec. II. Next we show how the detected staleness is reduced in Sec. III. Then we show how staleness identification and reduction is done for the AODV and DSR protocols in Sec. IV and V. We present our experimental setup in Sec. VI and describe our experiments in Sec. VII. The experimental results for the AODV and DSR protocols are described in Sec. VIII and IX.

II. Detecting Route Staleness

Consider the nodes pictured in Fig. 1 where node N has two neighbors A and B. Initially N has no routes and requires a route to node C. It sends out a route-request for C, and neighbor A replies that it has a route to C with distance 1 while neighbor B reports a distance of 4 as pictured in Fig. 1(a) (assume that routes through N aren't used). Suppose that later a new node D appears within communication range of A and B, as pictured in Fig. 1(b), and node N wishes to communicate with it. When N sends out a route-request for node D neighbors A and B both report routes of distance 1 to D. Now node N's route-table would be as pictured in the first table of Fig. 2(a). However this route-table does not make sense. When node D moved into range of nodes A and B, it made a route of length 2 available between A and B. Since A can reach C in one hop, B should now be able to reach C in 3 hops. However N's route table still claims that B has a distance of 4 to C. Therefore N's information about B's route to C has become stale since it doesn't reflect the fact that node D moved into range of A and B. We would like to use such inconsistencies to *detect* staleness. Strong Detection provides a tool to do this.

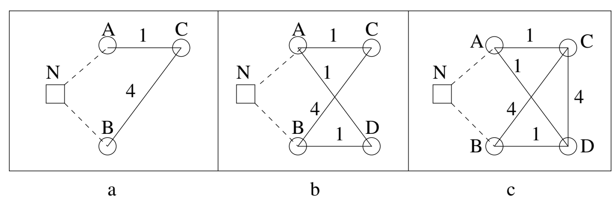


Fig. 1. Strong Detection Example (Networks)

A. Strong Detection

Strong Detection (SD) states that, in a network where the routing protocol P has converged, a node's N's routes must be *consistent*. By consistency SD means this: that there *must* exist some valid network G such that running the routing protocol P on G results in node N having the routes it does. The intuition behind this notion of consistency is that, even though an individual node may not be aware of the topology of the whole network, its routes *must* correspond to a valid network since its state is the result of the routing protocol being run on a valid network. If a node, given unlimited computational resources, cannot find a valid network that would result in its routes, the routes must be inconsistent.

Therefore if a node can check if there exists a valid network G that corresponds to its routes, it can verify consistency. To do this a node needs to enumerate all valid networks and simulate the protocol P on each valid network until it finds a network that results in its routes, or finds that no network does. This is clearly not practical since the number of valid networks that need to be checked can be very large and potentially infinite. However the authors of Strong Detection show that, in most cases, one can construct a new graph called the *canonical graph* C from node N's routes, and checking this one graph is enough to reveal inconsistent routes. This can be done in $O(|N|^2)$ time or less for the most frequently used distance-vector, path-vector and link-state protocols.

B. Strong Detection for Distance-Vector based protocols

Consider again the network of Fig. 1 where N receives the distance-vectors $< 1, 1 >$ and $< 4, 1 >$ from A and B indicating distances to C and D. This results in the route-table S of Fig. 2(a) where $d(R, Y)$ denotes the distance to node Y reported by neighbor R. To check if nodes N's state-table S is consistent, SD constructs the canonical graph C from S and runs the distance-vector protocol on this canonical-graph C. This results in a new state-table S' for node N. If $S = S'$ SD concludes that the state-table is consistent, else if $S \neq S'$ then SD concludes that S is inconsistent.

The canonical graph C containing all the nodes in S is constructed as follows: all pairs of nodes X, Y in C have an edge between them where the edge-weight e_{xy} is determined to be $e_{xy} = max_{R \in \mathcal{R}}(|d(R,X) - d(R,Y)|)$ In other words, it is the maximum difference in the distance reported to X and Y by any neighbor. The canonical-graph constructed for the state-table pictured in Fig. 2(a) is shown in Fig. 1(c). In this figure $e_{CD} = max(|d(A,C) - d(A,D)|, |d(B,C) - d(D,D)|) = max(0, 3) = 3$.

SD then runs the distance-vector protocol on the canonical graph resulting on a new state-table S' for node N. The reader can verify that running the distance-vector protocol on the canonical-graph pictured in Fig. 1(c) will indeed result in the state-table of Fig. 2(b).

a

	A	B
C	1	4
D	1	1

b

	A	B
C	1	3
D	1	1

c

	A	B
D	1	1

Fig. 2. Strong Detection Example (State-Tables)

C. Strong Detection for Path-Vector protocols

In path-vector routing algorithms, consistent routes will form a tree with the source node as the root. Inconsistencies among routes will reveal themselves as cycles. So investigating stale routes simply involves detecting the presence of cycles. Consider the example of Fig. 3 where node N has two neighbors A and B. Neighbor N initially requests a route to node D. Neighbor A reports a source-route A-C-D, while neighbor B has no route to D. Suppose, due to node movement, D then moves within communication range of neighbor B and the situation is as pictured in Fig. 4. Now if N requests a route to node E, neighbor B will report the source route B-D-E while A reports A-C-D-E and node N will choose the shorter route B-D-E.

Node N's routes to D and E now form the cycle N-A-C-D-B-E caused by the fact that N's route to D doesn't reflect the information that node D has moved within B's range while its route to E does. Note that all the links in the cycle are still valid. Cycles, then, in on-demand path-vector protocols indicate a lack of freshness of some routes.

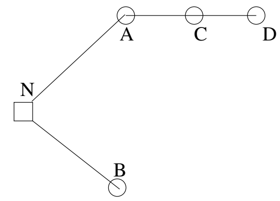


Fig. 3. Node N's routes to D

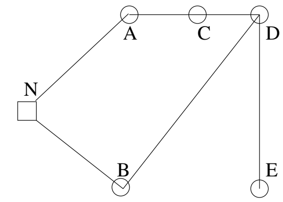


Fig. 4. N's routes to D and E

The canonical-graph C for path-vector based protocols is simply the tree formed by all the links of all the routes. In a protocol where the shortest path is preferred, the *shortest-path tree* is the canonical graph while in a protocol where the newest-path is preferred the canonical graph is the *newest-path tree* which is the shortest-path tree with the edges weighed by by the age of the links. Similarly when the preferred path is a combination of path-length and path-newness, the canonical graph is the shortest-path tree formed where the edge-weights are weighted combinations of age and distance to the root.

III. Fixing Route Staleness

Again, consider the nodes pictured in Fig. 1 where node N requests routes to C and D and receives responses from its neighbors A and B resulting in the state-table of Fig. 2(a). When SD is run on this state-table, it will result in the state-table shown in Fig. 2(b). Since these two state tables- are not the same, SD concludes that the routes are stale. We would now like to determine which particular routes contribute to the staleness and refresh them. We can do so by executing the following procedure:

- Arrange the routes chronologically, newest first.
- Initialize the state table S to be empty
- **For each** route R **do**
  - Add node and route-information about R from all neighbors N to the state table S
  - Run Strong Detection on the state table S
  - **if** Strong detection detects staleness **then**
    - $*$ Remove information about R from S
    - $*$ Refresh R
  - **else** (no staleness is detected)
    - $*$ Do nothing
- **done For**

We start with an empty state table, and add route information to it one at a time, newest to oldest. If adding a route's information causes the state-table to go stale, we remove it from the state-table and refresh it. We illustrate this procedure with the example used before: suppose that the routes to C and D of of Fig. 1 have timestamps of 5 and 10 seconds respectively. In executing the staleness-fixing procedure, node N would first add the newest route information (corresponding to route D) to an empty table resulting in the table shown in Fig. 2(c). A SD test of this table would indicate no staleness. Next, node N would add the information corresponding to the routes to C. This would result in the table of Fig. 2(a). A SD test of this table, as shown before, would result in an inconsistency, therefore node N would conclude that its route information for route C is stale, remove it from the state-table and refresh it.

A. Fixing Staleness for Path-Vector based protocols

The above procedure works for distance-vector based routing protocols. However in path-vector based protocols, inconsistencies in routes manifest themselves as cycles. Therefore to fix an inconsistency, we need to remove the cycle and restore the tree structure of the routes. When choosing the link to remove, there are two choices. Removing the link most likely to be invalid increases the chance that routes to the nodes in the cycle are valid. However removing links furthest from the root minimizes the mean path-length of routes to the nodes in the cycle. Consider the cycle N-A-C-D-B-N of Fig. 4. If we remove the link C-D furthest from the root N, the mean path-lengths from N to A,B,C and D is 1.5. If instead, we remove link N-A, the mean path-lengths from N to A,B,C and D would be 2.5!

If we have available a method by which we can test the validity of each individual link e_{ij} in the cycle, we could do the following: remove a link if it is invalid, but if all links are valid, remove the link furthest from the source node (i.e. remove $\{e_{ij}|d(N,j) > d(N,l)\forall e_{kl}\}$ where N is the source node and $d(i,j)$ is the distance from node i to node j). When we follow such a procedure, the result is a tree of routes from the host we call the *Canonical Tree*.

If, on the other hand, we do not have a method to test the validity of individual links, but have a method to estimate $P_v(e_{ij})$, the likelihood that a link is valid (say, from its age) then we can calculate the *weighted likelihood* of validity $V_{ij} = P_v(e_{ij})/d(N,j)$ for each link and remove the link with the smallest V_{ij}. When we use such estimation to remove cycles, we call the resulting tree the *Estimated Canonical Tree*.

We next describe how a node that has a collection of source-routes in its route-cache forms a canonical-tree or estimated canonical tree and illustrate it with an example. Note that a node only needs to form either one of the canonical trees. If it has a method available to test individual links, it forms the canonical tree, but if instead it can estimate a link's likelihood of being valid, it forms the estimated-canonical tree.

1) Forming a Canonical or Estimated Canonical Tree: Consider a situation where a path-vector protocol is being used and link ages (the last time it was used successfully) is known. Consider the situation where the source node A has the five routes pictured in Fig. 5, where the alphabets indicate nodes and the numbers indicate the edge age. From these five routes node A creates the graph or network pictured in Fig. 6. To compute the canonical-trees we first compute the shortest-path and newest-path trees.

2) Shortest-Path and Newest-Path trees: The shortest-path tree can be computed using Dijkstra's algorithm. If the edge-weighs are ignored (or all set to 1), the result is a shortest-hop tree. On the other hand, if the edge-weights or link-age is taken into account, the result is a newest-path tree (where a path from node A to any node X is the path where the sum of link-ages is the lowest). The shortest-path tree of the graph of Fig. 6 is pictured in Fig. 7 while the newest-path tree is pictured in Fig. 8.

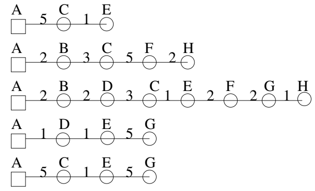


Fig. 5. Node A's route-cache

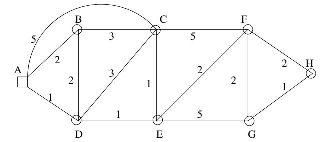


Fig. 6. The Network with all known Edges

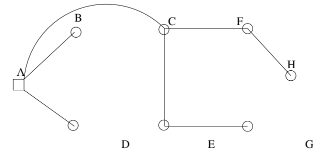


Fig. 7. The Shortest-Path tree

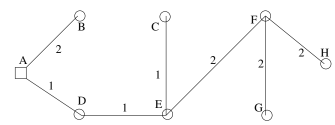


Fig. 8. The Newest-Path tree

3) The Canonical tree: Consider $sp(A,X)$ the shortest-path from A to X and $np(A,X)$ the newest-path from A to X where the shortest-path is strictly shorter than the newest-path or $len(sp(A,X)) < len(np(A,X))$. The edges that constitute $sp(A,X)$ but are not present in $np(A,X)$ provide short-cuts. Therefore to construct the canonical-tree we start with the newest-path tree and add each of these short-cut links to it one at a time. However each time we add one of these short-cut links to the newest-path tree, we will create a cycle. Therefore to restore the tree, we need to remove an edge from the cycle. If we test each link for validity and remove invalid links if they exist, but remove the link furthest from the source if all links are valid, we get the canonical tree mentioned above. The result of executing this procedure on the example graph of Fig. 6 produces the canonical-tree of Fig. 9. Note that if all candidate edges for addition turn out to be valid, the resulting Canonical tree is a shortest-path tree.

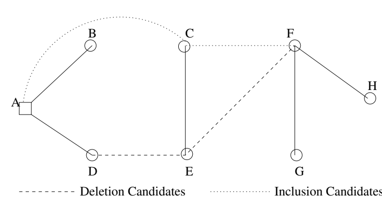


Fig. 9. The Canonical Tree

If on the other hand, if we estimate link-validity and we remove the link with the smallest weighted likelihood of validity L_{ij}, we form the estimated-canonical tree mentioned above.

IV. Improving Route-Freshness in AODV

In this section, we briefly introduce the AODV protocol and describe how the staleness-detection and staleness fixing techniques described for the distance-vector protocols can be specifically applied to AODV.

A. The AODV Protocol

The AODV protocol [?], is an on-demand protocol based on the Bellman-Ford Distance-Vector algorithm. AODV discovers routes only to nodes it wishes to send data to, and does so by network-wide broadcasting. To maintain route-freshness, it purges routes that have not been used for a pre-determined length of time called the *route-timeout interval*. However, node-movement can cause routes to become stale even before it is purged. AODV's route-table contains the distance and next-hop neighbor to each destination as in traditional distance-vector algorithms. Additionally, for each destination, it stores a sequence number that indicates freshness, generated by the destination. When discovering routes and when replying to route-requests AODV uses this sequence number to ensure that the freshest route information is used. Further, for each route, an AODV node maintains a list of active neighbors through which packets for the given destination are received so that if the link to this destination goes down, the upstream hosts using this link can be notified. The reader is referred to [?] for details about how AODV discovers routes, maintains routes and notices the arrival and departure of neighbors.

The staleness detection and staleness-fixing procedures for AODV is exactly that described for distance-vector based protocols in Sec. II-B and Sec III. We will analyze their time complexity next.

B. Time Complexity

Running Strong-Detection on a network of N nodes requires a time $O(d \times N^2)$ where d is the average out-degree of the network and N the number of destinations to which a node has routes at any time. The staleness-fixing algorithm runs Strong-Detection once for each route, and so the overall time-complexity is $O(d \times N^3)$. Since d is often small relative to N the time complexity is $O(N^3)$. Since N is small in AODV due to its on-demand nature and due to its frequent purging or routes, the staleness fixing procedure is computationally practical.

1) Optimizing Staleness-Fixing: The route-fixing algorithm's time complexity can be further reduced if a large fraction of the routes tend to be valid. In this case it is necessary only to detect the earliest route that is invalid. This can be done in $O(logN)$ time through a dictionary-search where the routes are repeatedly divided into halves and Strong-Detection is applied to the nodes in each half. The earlier of the two halves that shows an inconsistency is further divided and tested, until the two halves are just individual routes. Once the earliest invalid route is detected, the staleness-fixing algorithm of Sec. III-A is applied to the earliest invalid route and older routes. Even if around half the nodes are valid, a nearly ten-fold saving in running time can be had with this procedure.

V. Improving Route Freshness in DSR

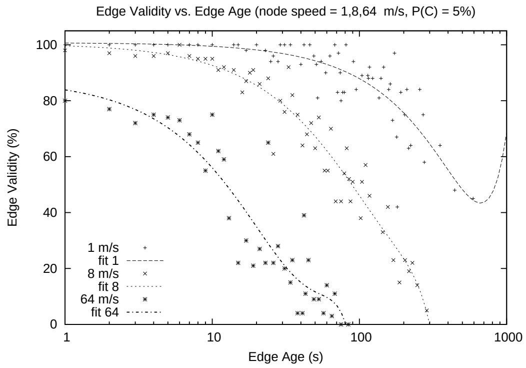
In this section we describe how the route staleness detection and fixing technique (SDF) is applied to DSR. We first briefly introduce the protocol, then show how the likelihood a link is valid is estimated so that it can be used to fix staleness and create the estimated-canonical tree described in Sec. III.

A. The DSR Protocol

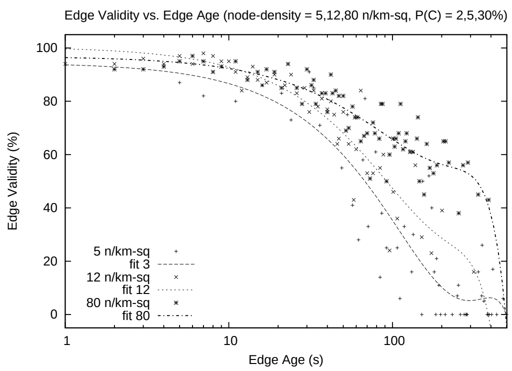
The Dynamic Source Routing (DSR) protocol [?], [?], [?] as its name suggests uses source routing where the originator of a packet determines the complete route from itself to the destination and includes the route in the packet. All intermediate hosts forward the packet based on this predetermined source route and make no routing decisions. A DSR node initiates route-discovery when it has no routes to a destination to which it needs to send a packet. If it receives multiple paths to the target node from its different neighbors it caches all these routes but uses the shortest. DSR nodes also cache routes and routes-fragments they overhear when they act as intermediate nodes in data-forwarding or route-discovery. If a node in a route discovers that it cannot forward packets to the next node in the route, it sends a route error packet to the source of the route. In case of fatal transmission, the erroneous hop is removed from all routes in the node's route cache by truncating the routes at the fatal link. When a route fails, a node uses an alternate route from its route-cache. If none exist, it re-initiates route discovery. DSR also caches routes indefinitely and unlike AODV never times-out routes. However, for each link in a DSR source-route, nodes maintain the last time it was known to be successfully used. This age information is updated by nodes on successful transmission of data. For further information on DSR's operations the reader is referred to [?].

Detecting stale routes in DSR consists in identifying cycles formed by the links of its routes as detailed for path-vector based protocols in Sec. II. Fixing stale routes in DSR consists of eliminating detected cycles. In choosing the link to remove, there are two approaches as mentioned in Sec. III. Edges can be tested for validity or estimates about the likelihood of it being valid can be used. Since DSR caches routes indefinitely, the number of cached routes may be large and a route in the cache may not be used again for a long time by which time a current validity test may no longer be useful. For these reasons, instead of testing routes for validity, we estimate the likelihood that a link is valid.

1) Estimating Edge Validity: As links age, they are less and less likely to be valid. However two other factors can affect edge-validity: when nodes are very mobile, links are likely to become invalid quickly but if they seldom move, links are likely to remain valid longer. This effect can clearly be seen in Fig. 10(a) where edge-validity is charted as a function of node-age at different node-velocities. Similarly in an environment where the node-density is high, two nodes picked at random are more likely to be withing communication radius, while in a less-dense environment two nodes picked at random are less likely to be within communication radius. Table II showed that the likelihood that two nodes picked at random are within communication radius varied from 0.8 % to 31% as the node density varied from 2 nodes/km^2 to 80 nodes/km^2. The effect of node-density on link-validity can be seen in the charts of



(a) Node-Velocity



(b) Node-Density

Fig. 10. Edge Validity vs. Age

Fig. 10(b) where edge-validity is charted as a function of node-age at different node-densities.

Since it is clear from Figs. 10(a) and 10(b) link-validity is influenced by node-age, node-velocity and node-density, to estimate link validity from these factors we need a function that estimates a links validity from its age, mean node-velocity and node-density. Therefore we collected edge-validity data as a function of edge-age (A), node-velocity (V) and node-density (D), then used a non-linear least-square estimator to fit the second-order equation Eq. 1 to the data and estimate its parameters $p_1 \dots p_9$.

$$V = p_0 + p_1 A + p_2 V + p_3 D + p_4 A^2 + p_5 V^2$$
$$+ p_6 D^2 + p_7 AV + p_8 VD + p_9 AD \quad (1)$$

Leaving out very small terms resulted in Eq. 2 that relates the likelihood that an edge is valid to its age, node-velocity and node-density.

$$V = 92 - 0.25A - 2.57V + 0.65D + 0.026V^2 - 0.006D^2 + 0.006VD \quad (2)$$

Now that we have a method to estimate the likelihood that the links in a cycle are valid, we can proceed to fix detected route-staleness and compute the estimated-canonical tree as detailed for source-route protocols in Sec. III. We briefly describe the time-complexity of our staleness detection and fixing procedure before proceeding to the experiments.

B. Time Complexity

Constructing each route-tree has a time complexity of $O(|N|)$, where $|N|$ is the number of nodes to which a host has routes. In constructing the canonical-graphs, $|N|$ links need to be checked if they are short-cuts. If they are, the $2 \times L$ links (where L is the mean path-length) that make-up the cycle need to evaluated as removal-targets. If we assume that L is small relative to $|N|$, then the overall time complexity is $O(|N|)$.

VI. Experimental Setup

In this section we describe the experimental setup used to measure the effectiveness of the staleness detection and fixing techniques (SDF) on AODV and DSR. The experiments were conducted on the current version (v 2.3) of the ns2 simulation environment. This version of ns2 contains an implementations of AODV and DSR originally created at the Monarch Project at CMU. Our implementations of SDF are built on top of the provided code.

All our experiments had 50 dynamic nodes that moved around a square grid randomly without pausing. Each node's transmission was simulated using the two ray ground propagation model, and the detection thresholds were set so that the communication radius was 250 meters resulting in a transmission area of approximately 0.2 km^2 per node. Each experiment was run for either 110 seconds or 1010 seconds and data collected 10 seconds before the end of the experiment. During each experiment the nodes were constantly in motion with direction and velocity regularly varied according to a uniform random variable. Periodically each node established connections with other arbitrarily-chosen nodes. During each connection which lasted 25 seconds, each node transmitted a stream of UDP packets to the partner node. The average rate at which the packets were sent is set to be a constant for each experiment (but varied as a parameter across experiments as described below). The inter-packet interval was however randomized within each experiment while maintaining the required overall packet-rate. The start time of each connection, and the target node were randomly chosen according from a uniform distribution while the number of connections per unit time was determined by the desired packet-rate. The parameters, their default values and if they were varied is summarized in Table I.

A. All-Seeing oracle

To establish the ground-truth or basis of comparison in our experiments we created an all-seeing *oracle* module that at any point in time is aware of the location of all the nodes and therefore the connectivity of the network. It also computes and is aware of the best routes (or the lack of one) between any two nodes at any point in time (such a perspective is of course not available in distributed protocols such as AODV or DSR). These *optimal* routes are used, where required, as the ground-truth or basis of comparison in our experiments.

VII. The Experiments

We conducted a series of experiments. In each experiment we studied performance by measuring mean path-length and

Parameter	Default Value	Type
Node Parameters		
Number of Nodes	50	Constant
Transmission Radius	250 m	Constant
Topology (shape)	square	Constant
Topology (area)	2000x2000 m^2	Variable
Node Density	12 nodes/km^2	Variable
Link Probability	5 %	Variable
Length of each transmission	25 seconds	Constant
Node Velocity	8 m/s	Variable
Packet Size	512 Bytes	Constant
Packet Rate	8 packets/sec	Variable
Simulation Parameters		
Iterations	500	Constant
Simulation Time	110 s	Constant
AODV Parameters		
Hello Interval	1 s	Constant
Timeout Interval	10 s	Varied
Purge Interval	0.5 s	Constant
Link-Layer Detection	OFF	Constant

TABLE I
PARAMETERS AND DEFAULTS

path-validity since both these factors influence packet-delay. Each experiment was run for a pre-determined length of time (110 seconds or 1010 seconds) and 10 seconds before the end of the experiment we recorded the following:

1) **[Mean path-length]** as measured by the total number of hops.
2) **[Path Validity]** The fraction of valid paths (a valid path is one where all edges in the path are valid).

We then studied how this performance changed as different experimental parameters varied. We chose to vary the following three parameters and study its effect on performance:

- **[The Data-Rate]** The average number of packets sent-out per second per node.
- **[The Velocity]** The average velocity with which the nodes moved around measured in m/s.
- **[The Node-Density]** The number of nodes per unit area measured in nodes per km^2.

So we conducted the three sets of experiments described below.

1) Data-Rate: In the first set of experiments we studied the effect of the inter-packet interval and data-rate on the efficacy of the staleness detection and fixing technique (SDF). We varied the rate at which packets were sent out during each connection. We recorded the performance while increasing the packet-rate from 1 packet/second to 128 packets/second.

2) Node-Velocity: In the second set of experiment we studied the effect of node-velocity on the effectiveness of SDF. We varied the mean velocity with which nodes moved around (while keeping the data-rates and node-density constant) exponentially from 1m/s to 64 m/s since a velocity of 1 m/s corresponds to human walking slowly, while a velocity of 64 m/s corresponds to the movement of a very fast train. While the required overall mean velocity was maintained at the required rate during each experiment, the actual velocity with which a node moved at any particular time was varied uniformly within the experiment.

3) Node-Density: The third and final set of experiments studied the effect of node-density on the effectiveness of SDF. The node density was changed by varying the length of the side of square grid in which the 50 nodes moved. This length was varied from 800 meters to 5000 meters in increments of approximately 25%. This resulted in node densities that ranged from 2 nodes/km^2 to 80 nodes/km^2. With this range of densities the likelihood of any two randomly chosen nodes being within communication range varies from 0.8 % to 31 % as shown in Table II.

For each set of experiments described above, the parameters that were not being varied were set to default values. The exact values taken on by the experimental parameters and their default values are displayed in Table II. It should be noted that the packet-rate, node-velocity and node-densities increase multiplicatively in each set of experiments.

Parameter	Values Taken	Default
Varied Parameters		
Data Rate	1 2 4 8 16 32 64 128	8
Velocity	1 2 4 8 16 32 64	8
Node Density	2 3 5 8 12 20 35 50 80 120	12
Connection Likelihood		
Likelihood	0.8 1.3 2 3 5 8 14 20 31	
AODV Bandwidth Parameters		
SD Interval	12 10 8 5 4 2.5 2 1.5 1 0.5	2.5
Timeout Interval	35 25 20 15 10 7 4 3 2 1	10

TABLE II
PARAMETERS AND THEIR EXPERIMENTAL VALUES

VIII. EXPERIMENTAL RESULTS FOR AODV

We first describe some of the parameters that affect the performance of AODV and the values we use for them. As mentioned, in AODV, routes once discovered are considered stale after a period of non-use called the *route timeout* interval which can be set. We used a default value of 10 seconds for this parameter, but varied it during the experiment as will be explained. The parameters of the experiments, their default values and whether they were varied are shown in Table I.

A. Is SDF Effective ?

Our preliminary experiments sought to understand the ability of our Strong Detection based detection-technique to detect a lack of freshness in a node's route-table. That is, how often does the detection-technique notice staleness *given that* routes are invalid. We measured that the detection-technique noticed inconsistency in a node's route-table approximately 72% of the time that the all-seeing oracle pointed out invalid routes. We also noticed that it occasionally pointed out a node's route-table to be invalid when all the the routes were valid but one or more were suboptimal, and longer than necessary.

We next sought to measure the effectiveness of our staleness-fixing technique where routes that contribute to staleness are detected and refreshed. Recomputing a route causes extra control packets to be generated, so running the staleness-fixing procedure increases bandwidth. Route-freshness however can also be improved in AODV by reducing the *route-timeout* interval causing routes to be recomputed more often, making them more accurate. So route-freshness can be improved at the cost of control bandwidth, with or without the staleness detection and fixing (SDF) technique. Therefore to asses if SDF is worth using, we need to measure its ability to improve route-freshness *at a given control bandwidth* and compare it to the improvement in route-freshness obtained at a similar bandwidth without SDF by just manipulating the route-timeout interval. Our experiments measure precisely this.

B. Experimental Results

For each set of experiments the route-accuracy vs. bandwidth curve is obtained with and without SDF. When SDF is not present the bandwidth is varied by setting the route-timeout value to each of the 10 values shown in Table II. When SDF is used, the route-timeout interval is set to the default value of 10 seconds, and SDF run periodically at each node. The bandwidth in this case is influenced by varying the frequency with which SDF is run to the values shown in Table II. Note that bandwidth cannot be directly controlled but is influenced by varying the route-timeout interval and the frequency with which SDF is run. Therefore our results chart the performance of AODV with and without SDF at a similar range of control bandwidths rather than at precisely the same bandwidths.

For each experiment, the simulation was run for 110 seconds and at the end of 100 seconds the following two values were recorded:

1) **[Bandwidth]** as measured by the total number of route-request packets received by each node per minute.
2) **[The fraction of sub-optimal routes]** as compared to the optimal-routes calculated by the omniscient oracle.

For each of the three experiments (node-velocity, node-density and packet-rate), we measured route-freshness with and without SDF while varying both the experimental parameter itself (for example node-velocity) and the bandwidth. This result in a surface (rather than a curve). For example when we studied the effect of node-velocity, we chose seven velocities ranging from 1 m/s to 64 ms/s. For each velocity we measured route-accuracy with and without SDF at a range of bandwidths. Therefore the experiment's results formed two surfaces (SDF and no-SDF) in a three-dimensional graph plotting route-accuracy as a function of bandwidth and node-velocity. Since it is hard to visually present surfaces, we present the result as two graphs: the first charts route-accuracy as a function of the experimental parameter (say velocity) independent of bandwidth, while the second charts performance as a function of bandwidth independent of the experimental parameter (e.g. velocity).

1) Node-Velocity: In Fig. 11(a), where we chart path-accuracy (y-axis) against node-velocity (x-axis) we see that the path-accuracy of SDF paths go from about 2% to 15% and the accuracy of non-SDF paths from 3% to 21% as the node-velocity increases from 1 m/s to 64 m/s. When path-accuracy is charted as a function of control-bandwidth in Fig. 12(a), at a bandwidth of 250 packets/min, about 4% of SDF paths are inaccurate while about 11% of non-SDF paths are inaccurate. At a rate of 450 packets/min 25% of non-SDF paths are inaccurate while only about 10% of SDF paths are inaccurate improving path-accuracy by over 50%. Note that since we have only indirect control over bandwidth in the experiments, the two curves span different bandwidth ranges.

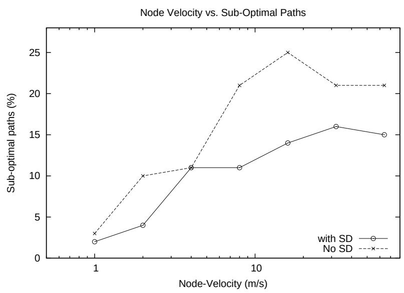
2) Packet Rate: In Fig. 11(b) where path-accuracy (y-axis) is charted as the packet-rate (y-axis) increases from 1 packet/minute to 128 packets/minute, we see that the packet-rate has little influence on the accuracy of paths. We note that non-SDF paths (top-curve) have an accuracy of about 19% across the range of packet-rates, while SDF paths (bottom-curve) vary in accuracy from 9 to 12%. When path accuracy is charted against bandwidth in Fig. 12(b) we see that the non-SDF paths use a bandwidth in the range of 250-400 packets/min while the SDF paths use bandwidths in the range of 350-500 packets/min. Since we don't control bandwidth directly in the experiments, but vary the route-timeout interval for non-SDF paths and the frequency at which SDF is run for SDF paths, the bandwidths ranges can be different.

3) Node-Density: Fig. 11(c) charts the fraction of sub-optimal or inaccurate paths (y-axis) as node-density increases from 2 nodes/km^2 to 80 nodes/km^2 (x-axis). The inaccuracy of both SDF and non-SDF paths increase with node density with SDF paths consistently more accurate. When we chart path-accuracy against bandwidth (y-axis) in Fig. 12(c) we see that non-SDF paths (top-curve) increasingly become more inaccurate until about 40% of the paths are sub-optimal at a bandwidth of about 750 packets/min. On the other hand, about 30% of SDF paths (bottom-curve) are inaccurate at about 1,500 packets/minute. SDF paths improve accuracy for a given bandwidth by 25 to 50% depending on the bandwidth.

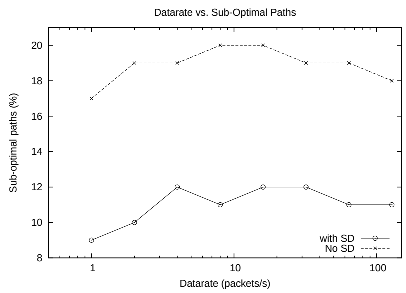
IX. Experimental results for DSR

In this section we present the experimental results of using our staleness detection and fixing technique (SDF) for DSR. To asses the performance of SD, we chart the validity and path-length of five different routes to each destination. The first route is the original DSR route to the destination. The next three routes, the shortest-path, newest-path and estimated-canonical path, are the result of executing SDF. The fifth, the canonical path, is formed by invoking the all-seeing oracle to detect invalid links. Since this oracle is not present in reality, we use this canonical-path as just a basis for comparison. We summarize below how the five different kinds of routes are formed.

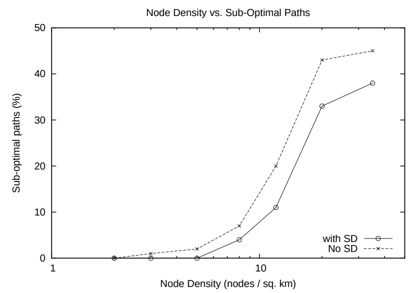
- **[DSR-Path]** The path to a node provided by the DSR protocol.
- **[Shortest-Path]** The shortest path to the node formed by using all the links found in all the routes in the node's route-cache.
- **[Newest-Path]** The newest path to the node formed by using all the links found in all the routes in the node's route-cache and *weighting each link by its age*.
- **[Canonical-Path]** The shortest path to the node formed by using all the links found in all the routes in the node's route-cache *after eliminating invalid links*.
- **[Estimated Canonical-Path]** The shortest path to the node formed by using all the links found in all the routes in the node's route-cache *using as edge-weight the link's estimated age weighted by its distance from the root*.



(a) Node-Velocity vs. Sub-Optimal paths

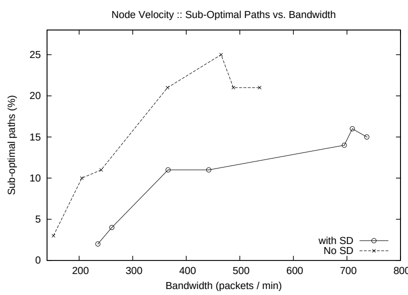


(b) Data-Rate vs. Sub-Optimal Paths

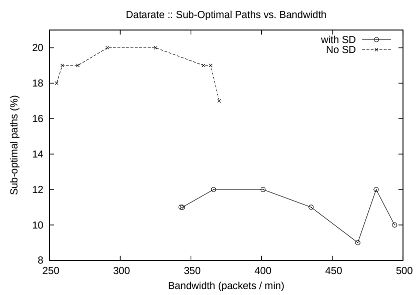


(c) Node-Density vs. Sub-Optimal paths

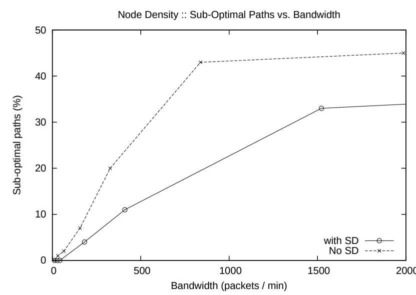
Fig. 11. AODV Performance with and without SD



(a) Bandwidth vs. Sub-Optimal paths

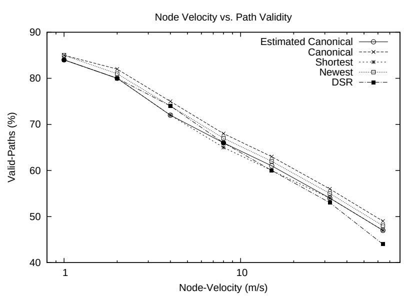


(b) Bandwidth vs. Sub-Optimal Paths

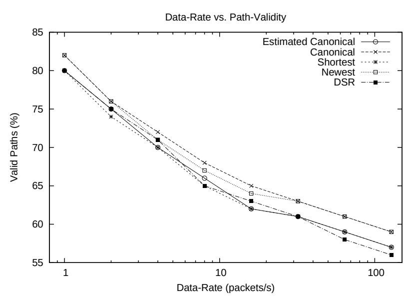


(c) Bandwidth vs. Sub-Optimal paths

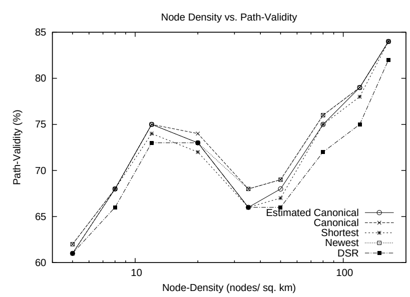
Fig. 12. AODV Performance vs. Bandwidth



(a) Node-Velocity vs. Path-Validity



(b) Data-Rate vs. Path-Validity



(c) Node-Density vs. Path-Validity

Fig. 13. DSR Performance :: Path-Validity

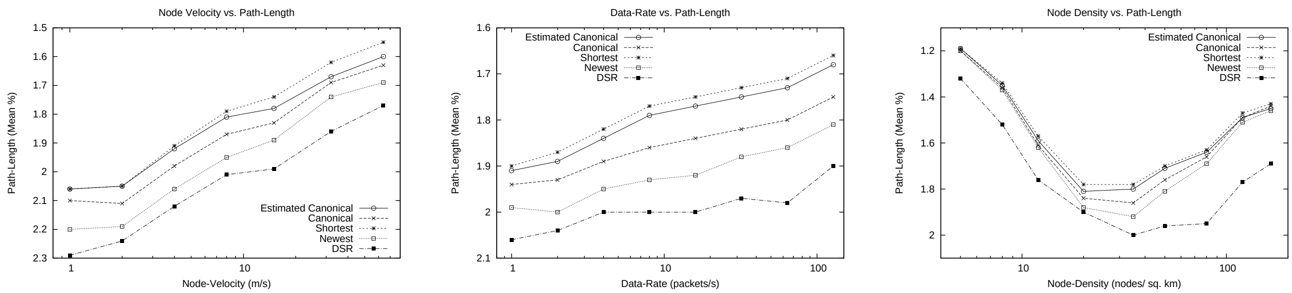
As before we provide three sets of results. One each while varying node-velocity, node-density and data-rate.

A. Node-Velocity

In Fig. 13(a) where we chart path-validity (y-axis) as the mean node-velocity (x-axis) increases from 1 m/s to 64 m/s, we see that validity decreases for all the different paths as node-velocity increases. As expected, the newest-paths have the best validity. At high node-velocities DSR paths lag the others paths by 5 to 10%. If Fig. 14(a) where we chart path-length in the y-axis, the mean path-length at a node-velocity of 64 m/s for the estimated-canonical paths (line with circles) is around 1.6 while the DSR paths (bottom curve) have a length of around 1.8. At the the node-velocity of 1 m/s, the estimated-canonical paths is about 2.15 compared to 2.3 for DSR paths. Overall the improvement is about 10%. We see that the shortest-paths, as expected, have the best path-lengths.

B. Data-Rate

In Fig. 13(b) where we chart path-validity (y-axis) against packet-rate (x-axis) we see that validity decreases with an increasing packet-rate probably because more paths are detected to be invalid as more data is transmitted. DSR paths are marginally less valid at high densities than the other paths. In Fig. 14(b) where we chart path-length (y-axis) against the packet-rate we see that the estimated-canonical paths (line with circles) are close to the optimum performance of the shortest-paths (top curve). At low packet rates, the estimated-canonical paths have a mean path-length of about 1.9 while DSR (bottom curve) has a length of about 2.07. At a high packet-rate of 128 packets/min, the estimated-canonical paths have a length of 1.68 as opposed to 1.9 for DSR, improving the mean path-length by around 12%.



(a) Node-Velocity vs. Path-Length (b) Data-Rate vs. Path-Length (c) Node-Density vs. Path-Length

Fig. 14. DSR Performance :: Path-Length

C. *Node-Density*

In Fig. 13(c) where we chart path-validity (y-axis) as node-density increases from 4 nodes / km^2 to 165 nodes / km^2 (x-axis) we see that the path-validity of all paths improve with increasing node-density except from about 12 nodes/km^2 to 30 nodes/km^2. If Fig. 14(c) where we chart path-length (y-axis) we see that at low node-densities the estimated-canonical path's length is 1.2 and is about 10% shorter than DSR paths. The short lengths are due to the network being so sparse that nodes can communicate with each other only when they are within direct communication radius of each other. As the node-density increases to 30 nodes/km^2 the path-lengths increase, showing the availability of multi-hop paths. For node-densities higher than 30, the path-lengths decrease again indicating that a choice of paths are available to the target. From the trend of the DSR curve (the bottom curve) it is clear that DSR doesn't use the shorter paths effectively. The estimated-canonical paths at high node-densities have a mean path-length of about 1.45 compared to a mean path-length of 1.69 for the DSR paths indicating a near 15% reduction in path-lengths.

X. Conclusion

We show that stale routes in mobile ad-hoc protocols can be detected by modifying a technique called Strong Detection conceived for static topologies. We then show how this detected staleness can be fixed efficiently by selectively refreshing routes. We demonstrate this for the AODV and DSR protocols and provide experimental results where staleness is detected in node's routes more than 70% of of he time. Additionally we demonstrate that combining this detection with selective refreshing of routes can improve route-freshness in AODV by about 30% and can reduce delay in DSR by improving its path-validity by 5 to 10% and path-lengths by 10 to 15%.