A Highly Adaptive Distributed Routing Algorithm for Mobile Wireless Networks

Vincent D. Park[a] and M. Scott Corson[b]
[a]*Naval Research Laboratory, USA*
[b]*University of Maryland, USA*

Abstract

We present a new distributed routing protocol for mobile, multihop, wireless networks. The protocol is one of a family of protocols which we term "link reversal" algorithms. The protocol's reaction is structured as a temporally-ordered sequence of diffusing computations; each computation consisting of a sequence of directed link reversals. The protocol is highly adaptive, efficient and scalable; being best-suited for use in large, dense, mobile networks. In these networks, the protocol's reaction to link failures typically involves only a localized "single pass" of the distributed algorithm. This capability is unique among protocols which are stable in the face of network partitions, and results in the protocol's high degree of adaptivity . This desirable behavior is achieved through the novel use of a "physical or logical clock" to establish the "temporal order" of topological change events which is used to structure (or order) the algorithm's reaction to topological changes. We refer to the protocol as the Temporally-Ordered Routing Algorithm (TORA).

1.0 Introduction

We consider the problem of routing in a mobile wireless network as described in [1]. Such a network can be envisioned as a collection of routers (equipped with wireless receiver/transmitters) which are free to move about arbitrarily. The status of the communication links between the routers, at any given time, is a function of their positions, transmission power levels, antenna patterns, cochannel interference levels, etc. The mobility of the routers and the variability of other connectivity factors result in a network with a potentially rapid and unpredictably changing topology. Congested links are also an expected characteristic of such a network as wireless links inherently have significantly lower capacity than hardwired links and are therefore more prone to congestion.

Existing shortest-path algorithms [2] and adaptive shortest-path algorithms [3-9] are not particularly well-suited for operation in such a network. These algorithms are designed for operation in static or quasi-static networks with hardwired links. If the rate of topological change in the network is sufficiently high, these algorithms may not be able to react fast enough (i.e. to maintain routing) and flooding will be the only recourse. Furthermore, most of these algorithms provide only one path for routing between each given source/destination pair which exacerbates the link congestion problem. While link-state algorithms provide the capability for multipath routing, the time and communication overhead associated with maintaining full topological knowledge at each router makes them impractical for this environment as well.

Some existing algorithms which have been developed for this environment include the following: the Gafni-Bertsekas (GB) algorithms [10], the Lightweight Mobile Routing (LMR) protocol [11], the Destination-Sequenced Distance Vector (DSDV) routing protocol [12], the Wireless Routing Protocol (WRP) [13], and the Dynamic Source Routing (DSR) protocol [14]. While these algorithms are better suited for this environment, each has its drawbacks.

The GB algorithms exhibit instability in portions of the network which become partitioned from the destination. During the period of instability, nodes will non-productively transmit both control packets and message packets until such time that the network is re-connected. This results in inefficient use of the available bandwidth and is unacceptable, since partitioning is expected to be common in a mobile wireless network.

The LMR protocol also exhibits some unwanted behavior which is most prevalent in partitioned portions of the networks. The protocol can result in temporary construction of invalid routes through "false reply" propagation. While it was shown that all invalid routes would be erased in a partitioned portion of the network (with probability one), no finite bound could be placed on the time required.

DSDV is limited in that it provides only a single path for routing between each given source/destination pair. Furthermore, the protocol requires selection of the following parameters: periodic update interval, maximum value of the "settling time" for a destination and the number of update intervals which may transpire before a route is considered "stale". It is difficult to assess the impact that selection of these parameters will have on performance, but we believe good parameter selection may be critical. These parameters will likely represent a trade-off between the latency of valid routing information and excessive communication overhead. To further complicate the problem, good parameter selection will likely be dependent on the networking environment (i.e. the size of the network, rate of topological change, etc.)

While WRP is described as providing only single path routing, nodes maintain sufficient information to perform multipath routing. However, there is potentially a significant amount of overhead associated with maintaining the shortest-path spanning tree reported by each neighbor and reactions to failures may be far-reaching (i.e. every node which includes the failed link in its shortest-path spanning tree must participate in the failure reaction).

DSR is also described as providing only single path routing; although, it could be amended to support multipath routing. More significantly, it suffers from a scalability problem due to the nature of source routing. As the network becomes larger, control packets (which collect node addresses for each node visited) and message packets (which contain full source routing information) also become larger. Clearly, this has a negative impact due to the limited available bandwidth.

In our view, a routing algorithm well-suited for operation in this environment should possess the following properties:

- Executes distributedly
- Provides loop-free routes
- Provides multiple routes (to alleviate congestion)
- Establishes routes quickly (so they may be used before the topology changes)
- Minimizes communication overhead by localizing algorithmic reaction to topological changes when possible (to conserve available bandwidth and increase scalability)

Routing optimality (i.e. determination of the shortest-path) is of less importance. It is also not necessary (nor desirable) to maintain routes between every source/destination pair at all times. The overhead expended to establish a route between a given source/destination pair will be wasted if the source does not require the route prior to its invalidation due to topological changes.

We have developed a routing algorithm which is tailored for operation in this highly dynamic network environment. The algorithm is based, in part, on the work presented in [10] and [11]; however, it does not share their undesirable characteristics associated with network partitions. The protocol is designed to *minimize* reaction to topological changes. A key concept in its design is that it *decouples the generation of potentially far-reaching control message propagation from the rate of topological changes.* Such messaging is typically *localized* to a very small set of nodes near the change without having to resort to a dynamic, hierarchical routing solution with its attendant complexity. A possible enhancement to the protocol (to be discussed later) would be to imbed far-reaching control message propagation into the protocol as a secondary mechanism. This propogation would occur periodically at a very low rate—independent of the network topology dynamics—and would be employed as a means of infrequent route optimization and soft-state route verification.

The algorithm is distributed in that nodes need only maintain information about adjacent nodes (i.e. one hop knowledge). It guarantees all routes are loop-free, and typically provides multiple routes for any source/destination pair which requires a route. Like LMR, the protocol is "source initiated" and quickly creates a set of routes to a given destination only when desired. Since multiple routes are typically established, many topological changes require no reaction at all as having a single route is sufficient. Following topological changes which do require reaction, the protocol quickly re-establishes valid routes. This ability to initiate and react infrequently serves to minimize communication overhead. Finally, in the event of a network partition, the protocol detects the partition and erases all invalid routes within a finite time.

2.0 The Protocol

2.1 Notation and Assumptions

We model a network as a graph $G = (N, L)$, where N is a finite set of nodes and L is a set of initially undirected links. Each node $i \in N$ is assumed to have a unique node identifier (ID), and each link $(i, j) \in L$ is assumed to allow two-way communication (i.e. nodes connected by a link can communicate with each other in either direction). Due to the mobility of the nodes, the set of links L is changing with time (i.e. new links can be established and existing links can be severed). From the perspective of neighboring nodes, a node failure is equivalent to severing all links incident to that node. Each initially undirected link $(i, j) \in L$ may subsequently be assigned one of three states; (1) undirected, (2) directed from node i to node j, or (3) directed from node j to node i. If a link $(i, j) \in L$ is directed from node i to node j, node i is said to be "upstream" from node j while node j is said to be "downstream" from node i. For each node i, we define the "neighbors" of i, $N_i \in N$, to be the set of nodes j such that $(i, j) \in L$. For the subsequent discussion, we assume the existence of a link-level protocol which ensures that each node i is always aware of its neighbors in the set N_i; although the logic of the protocol remains the same if this is not the case—i.e. there may be an arbitrary delay in the time between a link status change and subsequent protocol notification of the change. We also assume that all transmitted packets are received correctly and in order of transmission. Finally, since existing networks of this type typically employ omnidirectional antennas, we have assumed that when a node i transmits a packet, it is broadcast to all of its neighbors in the set N_i,. The rules of the protocol described herein reflect this assumption; however, only slight modifications to the rules would be required to make it work in networks where only a subset of the neighbors receive a transmission—i.e. those which incorporate Space Division Multiple Access (SDMA) techniques.

2.2 Foundation and Basic Structure

A logically separate version of the protocol is run for each destination to which routing is required. For the following presentation, we will focus on a single version running for a given destination.

The protocol can be separated into three basic functions: creating routes, maintaining routes, and erasing routes. Creating a route from a given node to the destination requires establishment of a sequence of directed links leading from the node to the destination. This function is only initiated when a node with no directed links requires a route to the destination. Thus, creating routes essentially corresponds to assigning directions to links in an undirected network or portion of the network. The method used to accomplish this is an adaptation of the query/reply process described in [11], which builds a directed acyclic graph (DAG) *rooted at the destination* (i.e. the destination is the only node with no downstream links). Such a DAG will be referred to as a "destination-oriented" DAG. Maintaining routes refers to reacting to topological changes in the network in a manner such that routes to the destination are re-established within a finite time. By this we mean that its directed portions return to a destination-oriented DAG within a finite time. Two GB algorithms, which are members of a general class of algorithms designed to accomplish this task, are presented in [10]. However, the GB algorithms are designed for operation in connected networks. Due to instability exhibited by these algorithms in portions of the network which become partitioned from the destination, they are deemed unacceptable for the current task. We have designed a new algorithm in the same general class, which is more efficient in reacting to topological changes and capable of detecting a network partition. This leads to the third function—erasing routes. Upon detection of a network partition, all links (in the portion of the network which has become partitioned from the destination) must be undirected to erase invalid routes.

The protocol accomplishes these three functions through the use of three distinct control packets: query (QRY), update (UPD), and clear (CLR). QRY packets are used for creating routes, UPD packets are used for both creating and maintaining routes, and CLR packets are used for erasing routes.

2.3 General Class of Algorithms

It is beneficial at this point to briefly review the GB algorithms. Consider a connected DAG with at least one node (in addition to the destination) which has no downstream links. We shall refer to such a DAG as "destination-disoriented." The following excerpts from [10] loosely describe two algorithms designed to transform a destination-disoriented DAG into a destination-oriented DAG.

Full Reversal Method: At each iteration each node other than the destination that has no outgoing links reverses the direction of all its incoming links.

Partial Reversal Method: Every node i other than the destination keeps a list of its neighboring nodes j that have reversed the direction of the corresponding links (i, j). At each iteration each node i that has no outgoing links reverses the directions of the links (i, j) for all j which do not appear on its list, and empties the list. If no such j exists (i.e. the list is full), node i reverses the directions of all incoming links and empties the list.

These two algorithms are subsequently re-stated in the context of a generalized numbering scheme which we will summarize here; however, much detail will be left out. For a thorough understanding, one should review the original paper. Essentially, a value is associated with each node at all times, and the values are such that they can be totally ordered. For example, in the full reversal method, a pair (α_i, i) is associated with each node where i is the unique ID of the node and α_i is an integer. The pairs can then be totally ordered lexicographically (e.g. $(\alpha_i, i) > (\alpha_j, j)$ if $\alpha_i > \alpha_j$,or if $\alpha_i = \alpha_j$ and $i > j$). Let us refer to the value associated with each node i as its "height" and denote it h_i. Now, assume that we assign an initial height to each node in the destination-disoriented DAG such that node i is upstream from node j if and only if $h_i > h_j$. Then it is clear that node i has no downstream links when, measured by its height, it is a local minimum with respect to its neighbors, $h_i < h_j$ for all $j \in N_i$. To achieve the desired behavior in the full reversal method, node i must select a new height such that it becomes a local maximum with respect to its neighbors, $h_i > h_j$ for all $j \in N_i$. Node i simply selects a new value $\alpha_i = \max\ \{\alpha_j \mid j \in N_i\}+1$ and broadcasts the value to all of its neighbors. The partial reversal method can neither be viewed conceptually nor explained as easily. Again, a node selects a new height only when it is a local minimum, but it does not always become a local maximum. To reverse only some of its links (i.e. partial reversal), a node selects a new height which is higher than its own previous height and the height of some of its neighbors, but not higher than all of its neighbors.

Further details of these two algorithms are not relevant to the development and discussion of our protocol. What is relevant is that [10] goes on to describe a general class of algorithms based on a generalized numbering scheme. This class of algorithms is shown to be loop-free, and terminate in a finite number of iterations to a destination-oriented DAG. Furthermore, only nodes which have lost all downstream paths to the destination react to a given failure. These properties all apply to the new algorithm as it is a member of this class. The new algorithm is similar to the partial reversal method in that it often reverses only some of its links. However, the rules for the selection of a new height are significantly more complex, in order to provide its partition detection capability. These rules are discussed in detail in section 2.4.2.

The basic idea is as follows. When a node loses its last downstream link (i.e. becomes a local minimum) as a result of a link failure, the node selects a new height such that it becomes a *global* maximum by defining a *new* "reference level". By design, when a new reference level is defined, it is higher than any previously defined reference levels. This action results in link reversals which may cause other nodes to lose their last downstream link. Any such node executes a partial reversal with respect to its neighbors that have heights already associated with the newest (highest) reference level. In this manner, the new reference level is propagated outward from the point of the original failure (re-directing links in order to re-establish routes to the destination). This propagation will only extend through nodes which (as a result of the initial link failure) have lost all routes to the destination. Any node, which prior to the start of this reaction had only downstream links, may experience link reversals (as a result of the same initial link failure) from all its neighbors. Any such node must select a new height such that it becomes a local maximum. This is accomplished by defining a higher sub-level associated with the new reference level, which we refer to as the "reflected reference level". This node essentially "reflects" this higher sub-level back toward the node which originally defined the new reference level. Should this reflected reference level be propagated back to the originating node from all of its neighbors, then it is determined that no route to the destination exists. The originating node has then detected a partition and can begin the process of erasing the invalid routes.

2.4 Detailed Description

At any given time, an ordered quintuple $H_i = (\tau_i, oid_i, r_i, \delta_i, i)$ is associated with each node $i \in N$. Conceptually, the quintuple associated with each node represents the height of the node as defined by two parameters: a reference level and a delta with respect to the reference level. The reference level is represented by the first three values in the quintuple while the delta is represented by the last two values. A new reference level is defined each time a node loses its last downstream link due to a link failure. The first value representing the reference level, τ_i, is a time tag set to the "time" of the link failure. For now we will assume that all nodes have synchronized clocks. This could be accomplished via interface with an external time source such as the Global Positioning System (GPS) [15] or through use of an algorithm such as the Network Time Protocol [16]. As we will discuss in section 2.5, this time tag need not actually indicate or be "time," nor will relaxation of the synchronization requirement invalidate the protocol. The second value, oid_i, is the originator-ID (i.e. the unique ID of the node which defined the new reference level). This ensures that the reference levels can be totally ordered lexicographically, even if multiple nodes define reference levels due to failures which occur simultaneously (i.e. with equal time tags). The third value, r_i, is a single bit used to divide each of the unique reference levels into two unique sub-levels. This bit is used to distinguish between the original reference level and its corresponding, higher reflected reference level. When a distinction is not required, both original and reflected reference levels will simply be referred to as "reference levels." The first value representing the delta, δ_i, is an integer used to order nodes with respect to a common reference level. This value is instrumental in the propagation of a reference level. How δ_i is selected will be clarified in a subsequent section. Finally, the second value representing the delta, i, is the unique ID of the node itself. This ensures that nodes with a common reference level and equal values of δ_i (and in fact all nodes) can be totally ordered lexicographically at all times.

Each node i (other than the destination) maintains its height, H_i. Initially the height of each node in the network (other than the destination) is set to NULL, $H_i = (-, -, -, -, i)$. Subsequently, the height of each node i can be modified in accordance with the rules of the protocol. The height of the destination is always ZERO, $H_{did} = (0, 0, 0, 0, did)$, where did is the destination-ID (i.e. the unique ID of the destination for which the algorithm is running). In addition to its own height, each node i maintains a height array with an entry $HN_{i,j}$ for each neighbor $j \in N_i$. Initially the height of each neighbor is set to NULL, $HN_{i,j} = (-, -, -, -, j)$. If the destination is a neighbor of i (i.e. $did \in N_i$) node i sets the height entry of the destination to ZERO, $HN_{i,did} = (0, 0, 0, 0, did)$.

Each node i (other than the destination) also maintains a link-state array with an entry $LS_{i,j}$ for each link $(i, j) \in L$, where $j \in N_i$. The state of the links is determined by the heights H_i and $HN_{i,j}$ and is directed from the higher node to the lower node. If a neighbor j is higher than node i, the link is marked upstream (UP). If a neighbor j is lower than node i, the link is marked downstream (DN). If the neighbors height entry, $HN_{i,j}$, is

NULL, the link is marked undirected (UN). Finally, if the height of node i is NULL, then any neighbor's height which is not NULL is considered lower, and the corresponding link is marked downstream (DN). When a new link $(i, j) \in L$ is established (i.e. node i has a new neighbor $j \in N_i$), node i adds entries for the new neighbor to the height and link-state arrays. If the new neighbor is the destination, the height entry is set to ZERO, $HN_{i,\,did} = (0, 0, 0, 0, did)$; otherwise it is set to NULL, $HN_{i,j} = (–, –, –, –, j)$. The corresponding link-state, $LS_{i,\,j}$, is set as outlined above. Nodes need not communicate any routing information upon link activation.

2.4.1 Creating Routes. Creating routes requires use of the QRY and UPD packets. A QRY packet consists of a destination-ID (*did*), which identifies the destination for which the algorithm is running. An UPD packet consists of a *did*, and the height of the node i which is broadcasting the packet, H_i.

Each node i (other than the destination) maintains a route-required flag, RR_i, which is initially un-set. Each node i (other than the destination) also maintains the time at which the last UPD packet was broadcast and the time at which each link $(i, j) \in L$, where $j \in N_i$, became active.

When a node with no directed links and an un-set route-required flag requires a route to the destination, it broadcasts a QRY packet and sets its route-required flag. When a node i receives a QRY it reacts as follows. (a) If the receiving node has no downstream links and its route-required flag is un-set, it re-broadcasts the QRY packet and sets its route-required flag. (b) If the receiving node has no downstream links and the route-required flag is set, it discards the QRY packet. (c) If the receiving node has at least one downstream link and its height is NULL, it sets its height to $H_i = (\tau_j, oid_j, r_j, \delta_j + 1, i)$, where $HN_{i,\,j} = (\tau_j, oid_j, r_j, \delta_j, j)$ is the minimum height of its non-NULL neighbors, and broadcasts an UPD packet. (d) If the receiving node has at least one downstream link and its height is non-NULL, it first compares the time the last UPD packet was broadcast to the time the link over which the QRY packet was received became active. If an UPD packet has been broadcast since the link became active , it discards the QRY packet; otherwise, it broadcasts an UPD packet. If a node has the route-required flag set when a new link is established, it broadcasts a QRY packet.

When a node i receives an UPD packet from a neighbor $j \in N_i$, node i first updates the entry $HN_{i,\,j}$ in its height array with the height contained in the received UPD packet and then reacts as follows. (a) If the route-required flag is set (which implies that the height of node i is NULL), node i sets its height to $H_i = (\tau_j, oid_j, r_j, \delta_j + 1, i)$—where $HN_{i,\,j} = (\tau_j, oid_j, r_j, \delta_j, j)$ is the minimum height of its non-NULL neighbors, updates all the entries in its link-state array LS, un-sets the route-required flag and then broadcasts an UPD packet which contains its new height. (b) If the route-required flag is not set, node i simply updates the entry $LS_{i,\,j}$ in its link-state array. The section on maintaining routes discusses the additional reaction that occurs if (b) results in loss of the last downstream link.

An example[1] of the route creation process is depicted in Fig. 1. The respective heights are shown adjacent to each node and the destination for which the algorithm is running is marked DEST. A circle around a node indicates that the route-required flag is set. Recall that the last value in each height is the unique ID of the node, and that lexicographical ordering (where 0 < 1 < 2... and A < B < C...) is used to direct links. Note that the height selected for node D in Fig. 1(e) reflects an arbitrary assumption that node D received the UPD packet from node E prior to the packet from node B. Had node D instead selected a height in response to the packet from node B, the direction of link (A, D) in Fig. 1(f) would have been reversed.

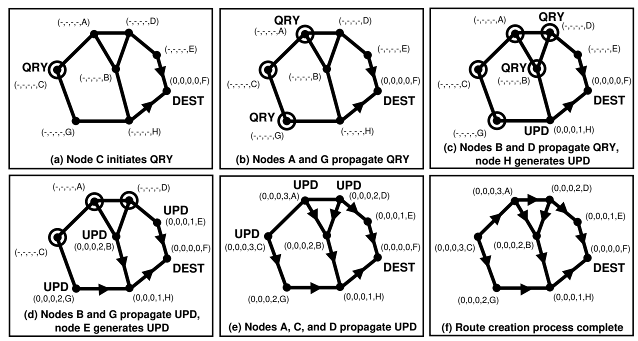

Fig. 1 Creating routes

2.4.2 Maintaining Routes. Maintaining routes is only performed for nodes which have a height other than NULL. Furthermore, any neighbor's height which is NULL is not used for the computations. A node i is said to have no downstream links if $H_i < HN_{i,\,j}$ for all non-NULL neighbors $j \in N_i$. This will result in one of five possible reactions depending on the state of the node and the preceding event. Each node (other than the destination) that has no downstream links modifies its height, $H_i = (\tau_i, oid_i, r_i, \delta_i, i)$, as follows.

Case 1 (Generate): Node i has no downstream links (due to a link failure).

$(\tau_i, oid_i, r_i) = (t, i, 0)$, where t is the time of the failure

$(\delta_i, i) = (0, i)$

In essence, node i defines a new reference level. The above assumes node i has at least one upstream neighbor. If node i has no upstream neighbors it simply sets its height to NULL.

[1] While the algorithm is designed to operate asynchronously, the examples depict the algorithm executing synchronously with transmissions occurring at fixed points in time.

Case 2 (Propagate): Node i has no downstream links (due to a link reversal following reception of an UPD packet) and the ordered sets (τ_j, oid_j, r_j) are **not** equal for all $j \in N_i$.

$$(\tau_i, oid_i, r_i) = \max\left\{(\tau_j, oid_j, r_j) \,\middle|\, j \in N_i\right\}$$

$$(\delta_i, i) = \left(\min\left\{ \delta_j \,\middle|\, \begin{array}{l} j \in N_i \text{ with } (\tau_j, oid_j, r_j) \\ = \max\left\{(\tau_j, oid_j, r_j)\right\} \end{array} \right\} - 1, i \right)$$

In essence, node i propagates the reference level of its highest neighbor and selects a height which is lower than all neighbors with that reference level.

Case 3 (Reflect): Node i has no downstream links (due to a link reversal following reception of an UPD packet) and the ordered sets (τ_j, oid_j, r_j) are equal with $r_j = 0$ for all $j \in N_i$.

$$(\tau_i, oid_i, r_i) = (\tau_j, oid_j, 1)$$

$$(\delta_i, i) = (0, i)$$

In essence, the same level (which has not been "reflected") has propagated to node i from all of its neighbors. Node i "reflects" back a higher sub-level by setting the bit r.

Case 4 (Detect): Node i has no downstream links (due to a link reversal following reception of an UPD packet), the ordered sets (τ_j, oid_j, r_j) are equal with $r_j = 1$ for all $j \in N_i$, and $oid_j = i$ (i.e. node i defined the level).

$$(\tau_i, oid_i, r_i) = (-,-,-)$$

$$(\delta_i, i) = (-, i)$$

In essence, the last reference level defined by node i has been reflected and propagated back as a higher sub-level from all of its neighbors. This corresponds to detection of a partition. Node i must initiate the process of erasing invalid routes as discussed in the next section.

Case 5 (Generate): Node i has no downstream links (due to a link reversal following reception of an UPD packet), the ordered sets (τ_j, oid_j, r_j) are equal with $r_j = 1$ for all $j \in N_i$, and $oid_j \neq i$ (i.e. node i did not define the level).

$$(\tau_i, oid_i, r_i) = (t, i, 0), \text{ where } t \text{ is the time of the failure}$$

$$(\delta_i, i) = (0, i)$$

In essence, node i experienced a link failure (which did not require reaction) between the time it propagated a reference level and the reflected higher sub-level returned from all neighbors. This is not necessarily an indication of a partition. Node i defines a new reference level.

Following determination of its new height in cases 1, 2, 3, and 5, node i updates all the entries in its link-state array LS; and broadcasts an UPD packet to all neighbors $j \in N_i$. The UPD packet consists of a *did*, and the new height of the node i which is broadcasting the packet, H_i. When a node i receives an UPD packet from a neighbor $j \in N_i$, node i updates the entries $HN_{i,j}$ and $LS_{i,j}$ in its height and link-state arrays. If the update causes a link reversal which results in node i losing its last downstream link, then it modifies its height as outlined in the cases above. Fig. 2 summarizes these five cases in the form of a decision tree, starting from the time a node loses its last downstream link. In the event node i loses a link $(i, j) \in L$ which is not its last downstream link, node i simply removes the entries $HN_{i,j}$ and $LS_{i,j}$ in its height and link-state arrays.

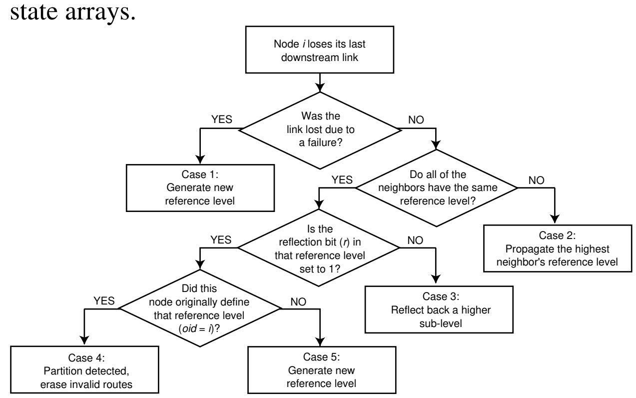

Fig. 2 Maintaining routes decision tree

The following examples illustrate how the algorithm works. Fig. 3 provides an example where no reaction is required. The network is first depicted as at the end of Fig. 1, with the addition that link (D, E) is marked as failing. Since all nodes still have downstream links following the failure, no transmissions are required. The significance of this is greater for networks which are highly connected. If a given node in the network on average has degree k (i.e. k adjacent links), then one could estimate the average number of downstream links for a given node to be (k/2). This implies that a node could tolerate (k/2)-1 downstream link failures without requiring any reaction. Fig. 4 provides an example where a reaction is required. The network is first depicted as at the end of Fig. 3, with the addition that link (B, H) is marked as failing.

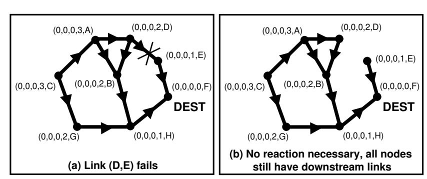

Fig. 3 Link failure with no reaction

2.4.3 Erasing Routes. Following detection of a partition (case 4), node i sets its height and the height entry for each neighbor $j \in N_i$ to NULL (unless the destination is a neighbor, in which case the corresponding height entry is set to ZERO), updates all the entries in its link-state array LS, and broadcast a CLR packet. The CLR

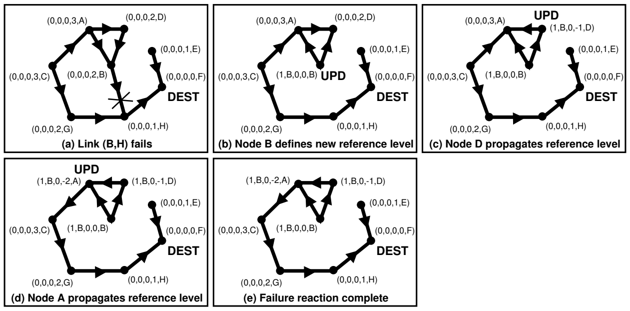

Fig. 4 Re-establishing routes after failure of last downstream link

packet consists of a *did* and the reflected reference level of node *i*, (τ_i, oid_i, 1)[2]. When a node *i* receives a CLR packet from a neighbor $j \in N_i$ it reacts as follows. (a) If the reference level in the CLR packet matches the reference level of node *i*; it sets its height and the height entry for each neighbor $j \in N_i$ to NULL (unless the destination is a neighbor, in which case the corresponding height entry is set to ZERO), updates all the entries in its link-state array *LS* and broadcasts a CLR packet. (b) If the reference level in the CLR packet does not match the reference level of node *i*; it sets the height entry for each neighbor $j \in N_i$ (with the same reference level as the CLR packet) to NULL and updates the corresponding link-state array entries. Thus the height of each node in the portion of the network which was partitioned is set to NULL and all invalid routes are erased. If (b) causes node *i* to lose its last downstream link, it reacts as in case 1 of maintaining routes. Fig. 5 provides an example which demonstrates partition detection and erasing of invalid routes. The

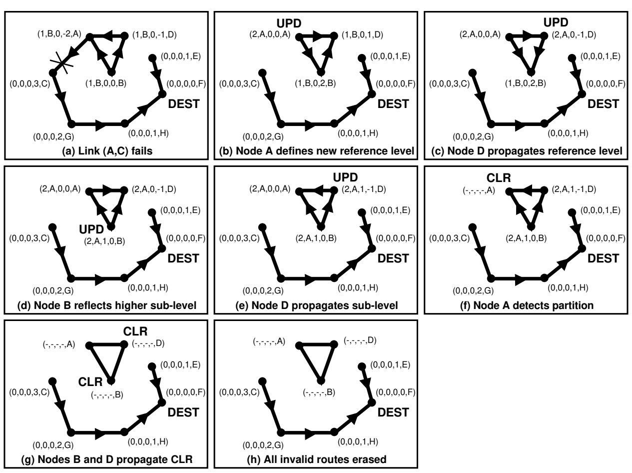

Fig. 5 Erasing invalid routes after a failure which partitions the network

[2] In actuality the value r_i = 1 need not be included since it is always 1 for a reflected reference level.

network is first depicted as at the end of Fig. 4, with the addition that link (A, C) is marked as failing.

It is advantageous to define the CLR packet with an additional one bit field, which we will refer to as a query flag. When a node would normally broadcast a CLR packet immediately followed by a QRY packet, the node sets the query flag and broadcasts only the CLR packet . Consequently, reception of a CLR packet with the query flag set is processed as if a CLR packet was received first and then a QRY packet was received.

We will summarize the results illustrated by the examples. When a failure causes a node to lose its last downstream link, the node will re-establish a route to the destination in one pass of the set of nodes affected by the failure (provided that a path to the destination exists). If a path to the destination no longer exists, the node will detect the partition in two passes of the set of affected nodes, and all invalid routes will be erased in three passes of the set of affected nodes.

2.5 Effect of Time Tag Errors

The effect of clock errors is difficult to bound analytically, or to determine quantitatively without simulation. The following is a general discussion regarding the effects of clock errors on protocol correctness and efficiency.

By using the assumption of synchronized clocks and time tagging the reference levels that are created each time a node loses its last downstream link, we are establishing the temporal order of these events. Because the algorithmic reactions are structured in this manner, we refer to the protocol as the Temporally-Ordered Routing Algorithm (TORA). If time tags are created by some other method, such that the relative ordering of the time tags still matches the temporal order of the corresponding events, the algorithm will function exactly as described thus far. For example, to evaluate the results of three link failures which occur at times 1, 2 and 3, it would not matter if the corresponding time tags were (1, 2, 3), (5, 6, 7), (10, 20, 30), or (4, 80, 900); the results would be the same. An excellent analysis on the ordering of events in a distributed system is provided in [17]. While the details will not be covered here, suffice it to say that simply establishing the order of events does not require the use of physical clocks. Such an ordering can be accomplished with “logical clocks” which can be implemented by counters with no actual timing mechanism.

Now, let us consider the effect of time tag errors such that the relative ordering of the time tags associated with a sequence of events does not agree with the actual order in which the events occurred. First, note that time tag errors do not exclude this algorithm from the general class; thus, all of the class properties are retained. What is lost, in some cases, is the efficiency with which routes are re-established.

Let us now consider how this applies to a practical implementation of the algorithm. If statistics are known about the rate of topological change in the network, one could use this information to determine a desired clock synchronization accuracy. For instance, if the average time between link failures is on the order of minutes, then achieving a clock synchronization on the order of seconds is very likely sufficient. While this would not guarantee preservation of the correct ordering of all events; intuitively, it seems unlikely that events would be incorrectly ordered very often.

3.0 Performance

There are no comparitive simulation results available at this time, although such work is underway. Instead, we present a comparitive summary of worst-case protocol complexities, augmented with a discussion of basic operation of several major protocol classes.

The complexities of TORA, along with an Ideal Link-state (ILS) algorithm, the DUAL family of algorithms, the GB full reversal algorithm, the LMR protocol, the DSDV protocol, and the WRP protocol are shown in Table 1. We borrow the complexity computations of ILS, and DUAL from [7] to which the reader is referred for details. The ILS protocol assumes that each network topology change must be sent to every node. DUAL is the lowest complexity, distance-vector, shortest-path algorithm known.

Table 1 Complexity Comparison

Protocol	TC	CC
ILS	$O(d)$	$O(2\|L\|)$
DUAL (link failure, cost increase)	$O(x)$	$O(6Dx)$
DUAL (link addition, cost decrease)	$O(d)$	$O(\|L\|)$
DSDV (link failure)	$O(x)$	$O(Dx)$
DSDV (periodic update)	$O(l)$	$O(\|L\|)$
WRP (link failure, cost increase)	$O(h)$	$O(Dx)$
WRP (link addition, cost decrease)	$O(d)$	$O(\|L\|)$
GB (connected, postfailure)	$O(2l)$	$O(lDx)$
GB (disconnected, postfailure)	∞	∞
LMR (connected, postfailure)	$O(2l)$	$O(2Dx)$
LMR (disconnected, postfailure)	$< \infty$ w.p.1	$< \infty$ w.p.1
TORA (connected, postfailure)	$O(2l)$	$O(2Dx)$
TORA (disconnected, postfailure)	$O(3l)$	$O(3Dx)$

In making comparisons, we make the same assumptions as [7]. We assume that the protocols execute synchronously. We compare the Time Complexity (TC), defined as the number of steps required to perform a protocol operation, and the Communication Complexity (CC), defined as the number of messages exchanged in performing the operation.

The complexity parameters are the number of network links $|L|$, the network diameter d, the number of nodes in a network segment materially affected by a topological change x, the length of the longest directed path in the affected network segment l, the height of the routing tree h, and the maximum nodal degree D.

The comparison shows that TORA's worst case complexity is generally better than the algorithms to which it is most closely related. In many cases, TORA would actually require only a single pass with TC $O(l)$ and CC $O(Dx)$ to react to a link failure, further improving its performance. Additionally, like GB and LMR, it has no explicit reaction to link additions further reducing its complexity relative to ILS, DUAL and WRP.

Having written this, we still refrain from placing much emphasis on worst-case complexity comparisons as they have limited value. Unfortunately, they are fuzzy and imprecise as it is difficult to compare algorithms with differing functionality in a precise, fair and meaningful fashion. For example, the number of nodes denoted by x is potentially different for each protocol, and the variables l and h are specific to given protocols. Rather, what is important is the protocol's average performance which is only obtainable via simulation.

We feel TORA has the potential to perform well relative to existing approaches based on the following reasoning. Existing approaches can be categorized into several broad classes which we now discuss. Link-state algorithms have the property that changes in link status, such as a failure, must be propagated to *all* nodes in the network. This is an example of "far-reaching" message propagation mentioned previously. Distance-vector approaches entail propagation of distance update information to a potentially large set of nodes, depending on the location of the change, in furtherance of a distributed shortest-path computation. Path-finding algorithms have characteristics of both link-state and distance-vector approaches, and seek to combine the best aspects of each into a hybrid protocol. Still, depending on the location of a change, a large set of nodes may be included in a shortest-path computation. The link-reversal mechanism of TORA forgoes propagation of link-state or distance information and, consequently, is able to *localize* its reaction to topological changes much more than the preceding classes. Its operation is best suited for relatively dense networks in which only several nearby nodes are typically involved in a reaction.

The effect of this localization is that the *scalability* of the protocol is greatly increased. Scalability, rather than being constrained by communication and time complexity, is now limited primarily by storage complexity, which only grows linearly with the number of nodes in the network.

4.0 Conclusions

We have proposed a highly adaptive distributed routing algorithm that is well-suited for operation in mobile wireless networks. It quickly creates and maintains loop-free multipath routing to destinations for which routing is required, while minimizing communication overhead. It rapidly adapts to topological changes, and has the ability to detect network partitions and erase all invalid routes within a finite time.

As mentioned earlier, the protocol is designed to decouple (to the greatest extent possible) the generation of far-reaching control message propagation from the dynamics of the network topology. Consequently, there is no distance estimate or link-state information propagation. A negative effect of this design choice is clear; viz. over time, as the link reversal process proceeds, the destination-oriented DAG may become less optimally directed than it was upon creation.

That is, upon route creation (before any subsequent link reversals), the DAG is formed and the fourth element of each node's height δ_i essentially contains the distance in hops from the destination over the path traveled by the UPD packet to the node (recall Fig. 1f). This distance information can be used, if desired, to favor routing over links with shorter distances; although—under heavy traffic conditions—we would not advocate routing all packets over a single path due to the congestion-enhancing effect of single-path routing [2]. As links are reversed in reaction to a failure, this distance information is lost in these "reversed" network portions (as δ_i no longer denotes distance to the destination when the reference level is not zero).

A possible enhancement to the protocol would be to periodically propagate refresh packets outwards from the destination, reception of which resets the reference level of all nodes to zero and restores distance significance to their δ_i's. The usage of periodic, destination-initiated, route optimization was mentioned as a possible routing enhancement in [18] and, later, a similar technique was developed as the major mechanism for route adaptation and maintenence in [12]. Besides serving as a routing enhancement, the periodic refresh guarantees that router state errors—resulting from undetectable errors in packet transmissions or other sources—do not persist for arbitrary lengths of time. Any router state which is not explicitly refreshed will eventually time-out and be deleted (i.e. returned to a NULL value). Thus, the periodic optimation also serves as soft-state confirmation of route validity.

This refresh process permits introduction of far-reaching control message propagation into the protocol in a fashion that is independent of network topology dynamics. The refresh interval is controllable, and the refresh procedure is expected to occur at a very low rate—it can be viewed as a secondary, background mechanism. The refresh overhead only grows linearly with the number of destinations in the network.

References

[1] M.S. Corson, S. Batsell and J. Macker, Architectural considerations for mobile mesh networking, working draft, May 1996, available at http://tonnant.itd.nrl.navy.mil/mmnet/mmnetRFC.txt.

[2] D. Bertsekas and R. Gallager, *Data Networks* (Prentice-Hall, 1987).

[3] P. Merlin and A. Segall, A failsafe distributed routing protocol, *IEEE Trans. Commun.* (September 1979).

[4] J. Jaffe and F. Moss, A responsive distributed routing algorithm for computer networks, *IEEE Trans. Commun.* (July 1982).

[5] P. Humblet, Another adaptive shortest-path algorithm, *IEEE Trans. Commun.* (June 1991).

[6] J.J. Garcia-Luna-Aceves, Distributed routing with labeled distances, *Proc. IEEE INFOCOM '92*, Florence, Italy (1992).

[7] J.J. Garcia-Luna-Aceves, Loop-free routing using diffusing computations, *IEEE Trans. Networking* 1(1) (1993)

[8] J.J Garcia-Luna-Aceves and S. Murthy, A loop-free path-finding algorithm: specification, verification, and complexity, *Proc. IEEE INFOCOM '95*, Boston, MA (1995).

[9] J.J Garcia-Luna-Aceves and J. Behrens, Distributed, scalable routing based on vectors of link states, *IEEE Journal on Selected Areas in Commun.* (October 1995).

[10] E. Gafni and D. Bertsekas, Distributed algorithms for generating loop-free routes in networks with frequently changing topology, *IEEE Trans. Commun.* (January 1981).

[11] M.S. Corson and A. Ephremides, A distributed routing algorithm for mobile wireless networks, *Wireless Networks* 1 (1995).

[12] C Perkins and P. Bhagwat, Highly dynamic destination-sequenced distance vector routing (DSDV) for mobile computers, *ACM SIGCOMM*, (October 1994).

[13] S. Murthy and J.J. Garcia-Luna-Aceves, An Efficient Routing Protocol for Wireless Networks, *ACM Mobile Networks and Applications Journal*, Special issue on Routing in Mobile Communication Networks, (1996).

[14] D. Johnson and D. Maltz, Dynamic source routing in ad hoc wireless networks, T. Imielinski and H. Korth, eds., *Mobile computing*, (Kluwer Academic Publ. 1996).

[15] NAVSTAR GPS user equipment introduction, MZ10298.001 (February 1991).

[16] D. Mills, Network time protocol, specification, implementation and analysis, *Internet RFC-1119* (September 1989).

[17] L. Lamport, Time, clocks, and the ordering of events in a distributed system, *Comm. of the ACM*, (July 1978).

[18] M.S. Corson, A. Ephremides, A Distributed Routing Algorithm for Mobile Radio Networks, *Proc. MILCOM '89*, Boston, MA, (October, 1989).