The University of Kansas

Information and Telecommunication Technology Center

Technical Report

Multi-Path Routing Protocol for Rapidly Deployable Radio Networks

Fadi Wahhab, Gary Minden and
Joseph Evans

ITTC-FY99-TR-13380-06

February 1999

Project Sponsor:
Information Technology Office
of the
Defense Advanced Research Projects Agency

Table of Contents

List of Figures

Abstract

Multi-hop wireless networks are an ideal technology to establish an instant communication infrastructure for civilian and military applications. Many different protocols have been proposed to solve the multi-hop routing problem in ad hoc networks, each based on different assumptions and intuitions. In this work, we present the Wireless Multi-path Routing Protocol (WMRP) designed for Rapidly Deployable Radio Network (RDRN) architecture. We provide a realistic, quantitative analysis comparing the performance of WMRP with other multi-hop wireless routing protocols. Simulation results show that the performance of WMRP is very good at all mobility rates and movement patterns.

Chapter 1: Routing Basics

1.0 Introduction

Routing is moving information across a network from source to destination. Along the way, typically at least one intermediate node is encountered. Several routing protocols have been designed over the years; by far, Routing Information Protocol (RIP) has been the most commonly used. Other popular routing protocols that are used in today's Internet include Cisco's EIGRP (Enhanced Internet Gateway Routing Protocol) and OSPF (Open Shortest Path First). Routing Algorithms are usually classified as either Distance Vector Algorithms (DVA) or Link State Algorithms (LSA). The key advantage of DVA is that they scale well for a given combination of services taken into account in a cost metric [2].

Wired networks have high bandwidth and an infrequently changing topology; while wireless networks have limited bandwidth and a frequently changing topology. Current routing protocols were designed for wired networks. Such routing protocols when used for a wireless network take a long time to converge (e.g. RIP), and incur a high overhead because of re-broadcasting complete topology information to each node (e.g. OSPF).

We show here our work on a routing protocol for rapidly deployable radio networks, which increases service availability by maintaining multiple routes between switching nodes. We provide a realistic quantitative analysis of the protocol and compare its performance to other proposed wireless routing protocol. In the rest of this chapter we discuss the basics of routing, distance vector algorithms and link state algorithms.

1.1 Distance Vector Algorithms

In a Distance Vector Algorithm (also known as Distributed Bellman-Ford), the routers send all or some portion of its routing table to its neighbors. Each router calculates the best path to each destination separately, trying to find a path that minimizes a simple metric, such as the number of hops to destination or path cost.

1.1.1 Routing Information Protocol

RIP maintains only the best route to a destination. When new information provides a better route, this information replaces old route information. Network topology changes can provoke changes to routes, causing a new route to become the best route to a particular destination. When network topology changes occur, they are reflected in routing update messages. For example, when a router detects a link failure or a router failure, it recalculates its routes and sends routing update messages. Each router receiving a routing update message that includes a change updates its tables and propagates the change. Each entry in a RIP routing table includes the ultimate destination, the next hop on the way to that destination, and a metric. The metric indicates the distance in number of hops to the destination. RIP permits a maximum

hop count of 15. Any destination greater than 15 hops away is tagged as unreachable. RIP's maximum hop count restricts its use in large networks, but prevents a problem called count to infinity from causing endless network routing loops. The *count-to-infinity* problem is described below. Consider the network shown in Figure 1.

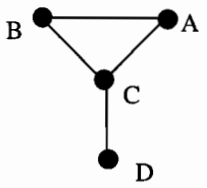


Figure 1: RIP Example

The routes for destination D from each router are shown below

C: directly connected, metric 1.

B: route via C, metric 2.

A: route via C, metric 2.

Now consider what will happen if the link from C to D fails. C notices it and sets the distance to D as infinity, but it gets a scheduled update from router A before it has had a chance to tell router A about its own change. So C updates its routing table entry for destination D as distance 3, next hop A. Then C will send updates to both A and B announcing that it can get to D in three hops. Then A will reply that it can get to D in four hops. This goes on until the metric for destination D reaches 16 (infinity) at all routers.

There are several things that can be done to prevent problems like this. The ones that RIP uses are called '*split horizon with poisoned reverse*' and '*triggered updates*'. Triggered updates speed up the process of discovering changes. Whenever a router changes its metric for a route it sends an update announcing the change. Split Horizon means that a router should not report a route to the next-hop system of that route. Routers A and B in the example above should never tell C that they have a route to D, because those routes have D as their next hop. Poisoned reverse goes further, with poisoned reverse routers A and B will have an entry for destination D in their update to C but with the metric set to infinity.

Even with the above solution, count to infinity problem might still occur in case an update gets lost (due to a transmission error). In the above example when router C notices that the link has failed it sets the distance to D as infinity and triggers updates to both A and B. Say now that the update to A never makes it. In the next scheduled update for router A it will inform B that it has a route to D with a metric 3. B will set its metric to D to 4 and send an update to C, which in turn will set its metric to D as 5, and so on...

1.1.2 EIGRP

IGRP was Cisco's solution to overcome RIP's shortcomings. It is a distance vector algorithm, but it uses a sophisticated metric using a formula that takes many factors into consideration including the network's delay and bandwidth and it uses a

technique called '*path holddown*' to prevent the counting to infinity problem. Holdown imposes a quarantine period after the detection of a link failure; during that period the destination is 'on hold' and no update is accepted.

EIGRP uses the same metric and routing formula as IGRP, but it cuts down on the routing traffic by sending updates only after a change and sending only the altered entries. Hello messages are added to monitor the reachability of neighbors. A reliable mechanism is used to ensure reliable delivery of messages between routers. EIGRP uses an algorithm called DUAL (Diffusion Update Algorithm) to prevent loops. The basic idea of DUAL is:

'*If a path takes you closer to a destination, then the path cannot be a loop.*'

EIGRP keeps a list of all possible paths in memory, rather than just one per destination. The advantage of multi-path algorithms is better throughput and reliability [3]. Routers can immediately switch to the next best path if the preferred path becomes unavailable. If the best route becomes unavailable and there are no alternate paths, the EIGRP router begins a new distributed calculation for a loop-free route to the destination by contacting its neighbors. A neighbor having a loop free route responds with the route; otherwise the neighbor contacts its neighbors. In essence, requests for recalculation flood through the network until a new loop-free route is returned.

1.1.3 Extended Bellman-Ford (ExBF)

The idea is to increase the information content of the routing tables and the routing updates, adding a second-to-last hop (predecessor) indication to the destination and metric pairs. This information is used to check recursively that the route to a destination is loopless. The condition is verified if the route to the predecessor is loopless and if the predecessor is not the local router; except if the route has exactly one hop. This algorithm was presented by two teams simultaneously in 1989 [12,13].

1.1.4 Merlin-Sigall (MS)

Merlin-Segall is a distance vector algorithm that avoids loops by coordinating the next hop updates for each destination as a diffusion computation which is started by the destination. When a failure/repair occurs, a request message is sent to each destination. The destination starts a new computation periodically or upon receiving a request message.

When a node receives a distance message from its next hop, the node calculates its distance, sends it to all of its neighbors except its next hop. After receiving a distance message from all its neighbors, the node chooses its new next hop, and sends the new distance to its old next hop.

1.2 Link State Algorithms

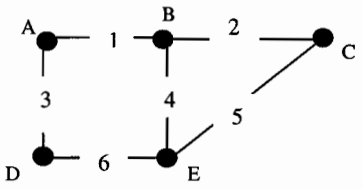
In Link State Algorithms (LSA), routers exchange information regarding the link characteristics instead of exchanging distances to destinations. Each node maintains a

map of the network that is updated quickly after any change in the topology (replicated distributed database approach). OSPF [24] and IS-IS [25] are the most popular LSA protocols.

1.2.1 OSPF

Thought the principle of OSPF is simple, it is a slightly complex routing protocol (The documentation of OSPF is a 244 page-document [24]). OSPF is composed of three sub-protocols: hello, exchange, flooding.

Figure 2 shows an example. The network consists of 5 nodes in which the routing database is replicated at each router. Each router computes the shortest path to the other routers with regards to itself as root (Shortest Path First). Any change in a state of any link is reflected in the database of each router with the flooding algorithm.



From	To	Link	Distance
A	B	1	1
A	D	3	1
B	A	1	1
B	C	2	1
B	E	4	1
C	B	2	1
C	E	5	1
D	A	3	1
D	E	6	1
E	B	4	1
E	C	5	1
E	D	6	1

Figure 2: OSPF example

When a link comes up, the two adjacent nodes synchronize their databases (each router will send the new neighbor a full copy of its database). Update records are transmitted to all other nodes using the flooding protocol. Routes are computed at each router after each change in the database.

1.3 Distance Vector versus Link State

Experts don't seem to agree which is the *better* algorithm: LSA or DVA. To quote two contradicting views, Jochen Behrens and J.J. Garcia-Luna-Aceves wrote:

> *'The main scaling problem of today's link state protocols are three: flooding consumes excessive communication resources, requiring each router to compute routes using the same topology at every router consumes excessive processing resources and communicating complete topology information is unnecessary.'*

While Huitema writes in [1]:

> *'The bandwidth argument is not very convincing. In fact, RIP – a distance vector algorithm- requires more bandwidth than OSPF because it needs to refresh the routing information periodically for each destination...'*

In fact he writes five pages on why link state is better, summarized by:

- Fast loopless convergence.
- Support of precise metrics and multiple metrics.
- Support of multiple paths to a destination.

A performance comparison between ExBF, SPF and MS is given by A.U. Shankar et al [14] using MaRS (Maryland Routing Simulator). The simulations use the NSFNET-backbone topology (14 nodes, 21 links) and compare the two algorithms with respect to delay and throughput under two different types of workload. The simulation results show that:

- All three routing protocols are practically equivalent with respect to throughput, and the routing load is negligible fraction of the overall network capacity.
- Under link failures and repairs, SPF and ExBF are practically equivalent in terms of delay; while MS had larger delays for high frequency of topology changes.

Both link state algorithms and distance vector algorithms are classified as proactive protocols. Proactive protocols continuously evaluate routes within the network, so that when a packet needs to be forwarded, the route is already known and can be immediately used.

Chapter 2 discuses some of the proposed protocols that try to tackle the routing problem in wireless networks, and gives a detailed description of WMRP. Chapter 3 describes the simulation environment and analyzes the results and also compares the performance of WMRP to other wireless routing protocols.

Chapter 2: Routing Protocol for Rapidly Deployable Radio Networks

2.0 Introduction

Wireless networks can change their topology quite often, requiring a large amount of routing traffic. This contradicts with the fact that wireless networks generally have less bandwidth. Several ongoing projects are working on this problem. One approach is to use a reactive protocol [16, 17, 18, 9]. Reactive protocols invoke route determination upon demand only. Another approach is to use hybrid reactive/proactive routing protocols [19, 20]. Other approaches are used in [7, 21]. The following protocols are discussed in the next section:

- Destination-Sequenced Distance-Vector (DSDV).
- Dynamic Source Routing (DSR).
- Temporally-Ordered Routing Algorithm (TORA).
- Ad Hoc On-Demand Distance Vector (AODV).
- Wireless Routing Protocol (WRP).

2.1 Review of Existing Protocols

2.1.1 Destination-Sequenced Distance-Vector (DSDV)

DSDV [21] extends the basic DV algorithm to include a sequence number with each route. Each node periodically broadcasts its own routing table. The transmitted

routing table also includes the sequence number created by the transmitter. A route decision is based on the sequence number (larger sequence number means a more recent route which is favorable). If two routes have the same sequence number then the decision is based on the metric (lower metric is favorable). DSDV also introduces a quick re-broadcast when a new or substantially modified route information is received to reduce the convergence time as much as possible. DSDV defines two types of dumps, the first is called *full dump* which carries all available routing information, and the second is called *incremental dump* which carries information changed since the last full dump. Full dumps can be transmitted relatively infrequently when no movement of nodes is occurring.

Situations can develop where a node receives new routing information in a pattern that causes it to consistently change routes from one next hop to another, even when the destination node has not moved. A node can always receive two routes to the same destination, with a newer sequence number, one after another (from different neighbors), but always get the route with the worse metric first. This could lead to a continuing burst of new route transmissions. This problem can be solved by delaying the advertisement of such routes, when a node can determine that a route with a better metric is likely to show up soon.

2.1.2 Dynamic Source Routing (DSR)

DSR [18] uses source routing. The sender of packet determines the complete sequence of nodes through which to forward a packet, and lists this route in the packet's header. When received by each node along the path, the packet is simply retransmitted to the next hop specified the packet's header. DSR doesn't use any periodic updates. Each node maintains a *route cache* in which it caches source routes that it has learned. When a node needs to send a packet to another node, it checks its route cache for a source route to the destination. If a route is found, the sender uses this route to transmit the packet. If no route is found, the sender attempts to find one through a mechanism called *route discovery*. In route discover, the sender broadcasts a *route request* packet, which propagates as needed through the network. Each route request from a sender has a unique request id. When receiving a request, if the node has previously seen this request id, it discards this copy of the request. When a route is found, the reply is sent in a *route reply* packet. Each node maintains a cache of source routes that it has learned or overheard which it uses to limit the propagation of the route requests.

If while forwarding a packet with a routing header, the next hop specified in the source route is found to be unreachable, a *route error* packet is returned to the originator. This mechanism is called *route maintenance*. The originator can then invoke route discovery again to find a new route. If the forwarding node can find a

route in its cache for another route to the destination, it replaces the broken source route on the packet with the route in its cache and retransmits the packet.

2.1.3 Temporally-Ordered Routing Algorithm (TORA)

TORA's [16] underlying algorithm is neither a distance-vector nor a link-state; it is one of a family referred to as *link reversal* algorithms. It is designed to discover routes on demand and provide multiple routes to a destination. Route Optimality is considered to be of secondary importance, and longer routes are often used to avoid the overhead of discovering newer routes.

Each node in the network need only maintain information about adjacent nodes, and runs a separate copy of TORA for each destination to which routing is required. TORA has three basic functions: creating routes, maintaining routes, and erasing routes. When a node requires a route to a destination, it broadcasts a *query* packet containing the address of the destination. The packet propagates through the network until it reaches a node having a route to the destination. The recipient then broadcasts an *update* packet listing its *height* with respect to the destination. As the update packet propagates through the network, each node that receives the update sets its height to a value larger then the height of the neighbor that it received the update from. This builds a direct acyclic graph (DAG) rooted at the destination. When a node discovers that a route to a destination is no longer available, it adjusts its height and

transmits an update packet. When a node detects a network partition, it generates a *clear* packet that removes all invalid routes from the network.

TORA was designed to run over IMEP (Internet MANET Encapsulation Protocol), which provides reliable and in order delivery of the routing messages, plus notification of link failures. IMEP also tries to reduce overhead by aggregating TORA and IMEP control messages together in a single packet.

2.1.4 Ad Hoc On-Demand Distance Vector (AODV)

AODV [17] is a combination of DSR and DSDV. It combines the hop-by-hop routing, and sequence numbers from DSDV, and the basic on-demand route discover mechanism from DSR.

When a node determines that it needs a route to destination, it broadcasts a route request. A node receiving the route request message checks if it has received the same request previously. If such a request had been received, the node discards the new request. Otherwise, it checks to see whether it has a route to the destination. If the node doesn't have a route, it rebroadcasts the request. When the request reaches a node that has a route for the destination, it compares the destination sequence number for that route with the destination sequence field in the request. If the destination sequence number field for that route is smaller than the destination sequence number field of that route, the node rebroadcasts the request, just as if it did not have a route

for that destination at all. If the destination sequence number of the route is larger than the destination sequence number of the request, the node generates a route reply that contains the number of hops necessary to reach the destination and destination sequence number from its route table entry. Each node that participates in forwarding the route reply toward the originator creates a forward route that to the destination.

AODV maintains routes by requiring each node to periodically transmit a hello message. Failure to receive three consecutive hello messages from a neighbor is taken as an indication that the link to that neighbor is down. AODV also suggests that a node may use link layer methods to detect link breakage.

2.1.5 Wireless Routing Protocol (WRP)

The underlying algorithm of WRP [7] (Wings Routing Protocol) is a *path finding* algorithm that utilizes the information regarding the length and second-to-last (predecessor) of the shortest path to each destination to eliminate the counting to infinity problem of DBF. Each node maintains a spanning tree reported by its neighbors, and uses this information along with the cost of adjacent links to generate its own shortest-path spanning tree (this is done by path traversal on the predecessor entries). Nodes exchange their hierarchical routing trees incrementally by communicating only the distance and the second-to-last hop to each destination.

Each node checks for the connectivity with its neighbors periodically by transmitting a Hello packet if it does not have any routing update messages to transmit during the Hello interval. Because of the broadcast nature of the radio channel (which WRP was designed for), a node sends the same information to all its neighbors informing them about changes in its routing table. To ensure reliable transmission, WRP implements a message retransmission list that collects ACK messages from each neighbor for each update message sent. A node retransmits the update message after a timeout if it does not get an ACK from all its neighbors with the list of neighbors that need to ACK this update.

In WRP, a node checks the consistency of predecessor information reported by all its neighbors each time it processes an event (regenerating the whole spanning tree). This feature accounts for faster convergence but also makes the algorithm complex and computationally expensive.

2.2 Rapidly Deployable Radio Networks (RDRN)

The RDRN project is funded by the Information Technology Office (ITO) of the Defense Advanced Research Projects Agency. RDRN is made up of two types of nodes: Mobile End Points (MEP) and Mobile Access Points (MAP) that can be deployed rapidly in areas of military conflicts or civilian disasters that lacks the communication infrastructure. Both types of nodes contain GPS receivers for location determination, software controlled radios with phased-array antennas for beam

forming and pointing in the right direction using GPS derived location information, and network control software.

The RDRN architecture consists of three overlaid radio networks:

1. A low bandwidth, omni-directional order wire packet radio network for broadcasting location information, network configuration and management.
2. A cellular like radio network for connecting mobile end points to mobile access points.
3. A high capacity wireless backbone network providing connections between mobile access points with multiple directional beams.

Each RDRN node determines its location from GPS, broadcasts this information over the order wire network, then establishes high capacity links to nearby nodes using the steerable phased array antenna. Nodes keep updating the location information of other nodes and change the network topology accordingly.

RDRN is also adaptable to changes in the quality of the radio communication environment. While ATM is designed to operate on high quality (almost error free) wired links, typical radio links suffer higher error rates and the link quality changes as a function of time due to mobility and changes in the environment. By estimating the channel parameters such as multi-path spread and signal to noise ratio, communication parameters at the link and network levels are adapted to provide appropriate throughput and quality of service.

2.3 Wireless Multi-Path Routing Protocol (WMRP)

The idea of the proposed protocol is a multiple path extended distance vector algorithm that utilizes information regarding the length and second-to-last (predecessor) of the shortest path to each destination to prevent loops. To reduce routing overhead, updates are sent only after a topology change. To ensure connectivity, Hello packets are exchanged between neighboring nodes every Hello interval. If no such packets are received at a certain node from its neighbor for three consecutive Hello intervals, the node assumes that connectivity with that neighbor has been lost. The protocol is similar in nature to WRP. We also provide multiple paths, make use of the channel nature of RDRN network to send different information to different neighbors (split horizon) which increases the route availability, and send changes in a route to the neighbors affected by that change. We also enforce a hold-off time before new neighbors exchange their routing tables. A detailed description of the protocol is given below.

Notation and Assumptions

To describe WMRP, we model the network as an undirected graph represented as G(V,E), where V is the set of nodes and E is the set of links (or edges) connecting the nodes. Each node represents a router, When (u,v) is an edge of the graph G, u is said to be adjacent to v (or a neighbor of v), and v is a neighbor of u. A route from node x to destination node j is a sequence of adjacent nodes ($x, k_1, k_2, \ldots k_n, j$) denoted by R_{xj}. A path from x to j via node k is denoted R^k_{xj} where k is a node adjacent to x. The distance is the number of hops between x and j (sum of link weights each of weight

equal to 1) $D(R_{xj})$. The predecessor node of a path R_{xj} is defined to be the last node preceding node j in the sequence of node in R_{xj}. denoted as $P(R_{xj})$. N_x is a list (set) of neighbors of node x.

The description makes the following assumptions:

- Each node has a unique identifier.
- A node detects the existence of a new neighbor within a finite time (order wire agent).
- An underlying (Data Link Control) protocol ensures that packets are delivered correctly. (Reliable transmission of update messages can be implemented by means of retransmissions if no such layer exists).

All links are assigned the same weight (a value of 1). All links between MAP nodes in RDRN are identical with respect to bandwidth and delay. Reliability (error rate), on the other hand, is highly variant during the operational period of the link; so it would require a constant monitoring and changing the link weight if the reliability is to be included in the calculation of the link weights which increases the routing overhead.

2.3.1 Information maintained at each node

Each node maintains the folowwing information

1. A *routing table,*
2. A list of neighbors (N_x) and

3. A list of new neighbors (B_x).

The routing table of node x contains an entry for each destination j, each of those entries contain the following:

- The destination identifier (j).
- A list of neighbors, if any, that would be affected by the change and need to be informed about it (N_{xj}).
- One or more path(s) information, each path is identified by a different neighbor and contains the following information:
  - The identifier of the next hop (k).
  - Metric representing the distance to j $D(R^k_{xj})$.
  - The identifier of the predecessor (next to last) $P(R^k_{xj})$.

Note that the number of paths from node x to any destination is limited by the degree (number of adjacent nodes) of node x because each entry (path) is identified by a different neighbor. When the routing agent is informed about a new neighbor k (through order wire agent) and that a new link has been established, x adds k to the list of new neighbors (B_x), x and k start exchanging Hello messages. Three Hello messages have to be exchanged within three Hello intervals (Hold-off time) before the two nodes exchange their routing information. This procedure is adopted for the following reasons:

1. Allows the nodes to check wireless reachability (not just order wire reachability) and makes sure that the link is reliable.

2. Checks that the other node is not just passing by quickly.
3. Gives the nodes enough time to detect any link failures with previous neighbors. A new link coming up, would be due to a change in the topology, which means the one of the two nodes (if not both) has moved and most probably lost connectivity with one or more of its previous neighbors. The hold-off would give enough time for invalid routes to timeout before exchanging information. This prevents propagating wrong information through the network.

Note that four Hello messages can be exchanged with a three Hello interval period, so this works as a 3 out-of 4 system. If three Hellos are exchanged successfully within the hold-off time (three Hello intervals), then the two nodes will exchange their routing information (x will remove k from B_x and add it to N_x, and send it a summary of its routing table). Otherwise, x removes k from its list of new neighbors (B_x).

2.3.2 Information exchanged among nodes

Nodes exchange periodic Hello packets. Routing table update messages are only sent after a change in the routings table (reflecting a change in the topology). Updates are attached with the next Hello packet. Each entry (U^k_j) in an update packet (U^k) from a neighbor k about destination j contains the following:

- The identifier of the destination node *j*.
- The distance to j $D(U^k_j)$.
- The identifier of the predecessor node $P(U^k_j)$.

2.3.3 Processing an Update

When node *x* receives an update message for destination j (route R_{xj}) from a neighbor k, it verifies the information in the update by checking if it has in its routing table an entry for the predecessor with the same neighbor k (R^k_{xp}) and a distance $D(U^k_j)$. If the update is verified, x updates its table and set the distance to destination j ($D(R^k_{xj}) = D(U^k_j) +1$); otherwise $D(R^k_{xj})$ is set to ∞. The Update procedure checks the neighbors that will be affected by the change in the route R^k_{xj} and adds those neighbors to N_{xj}. Figure 3 shows the update algorithm in more details.

```
Procedure Verify-Update (U^k_j)

    // if reported distance to j is infinity
    if ( D(U^k_j) == infinity )
        Update-Table (R^k_xj , U^k_j)
        return
    fi

    // if we have an entry in the table for the
    // predecessor with the same reported
    // distance
    // or the reported distance is 0 and the
    //  destination is a neighbor
    if ( ( D(U^k_j )== 0 && k ∈ N_x) ||
          D(U^k_j ) == D(R^k_xp) )
        Update-Table (R^k_xj , U^k_j)
        return
    else
        U^k_j .D = infinity
        Update-Table (R^k_xj , U^k_j)
        return
    fi
end
```

```
Procedure Update-Table (R^k_xj , U^k_j )

    // find the shortest path information
    // that was reported to each neighbor
    // before updating the information
    for each n in N_x
        min_before[n]=Find-Min (R_xj ,n)
    rof
    // update the entry
    D(R^k_xj) = D(U^k_j) + 1
    P(R^k_xj) = P(U^k_j)
    // find the shortest path information
    // to each neighbor after the update
    for each k in N_x
        min_after = Find-Min (R_xj ,n)
        // if the shortest path information
        // changed for a neighbor
        if(min_after≠min_before[n] ||
            min_before[n]==k)
                // add the neighbor to the list of
                // neighbors to be updated
                N_xj = N_xj + k
        fi
    rof
end
```

Figure 3: Update Algorithm

When an update is made to an entry in x's table (R_{xj}), the neighbors that are affected by this change are added to the neighbors list (N_{xj}) so that they can be informed about the change with the next Hello packet. Note also that x informs each neighbor k about the minimum path it has to a destination j that doesn't have k as the next hop or predecessor (split horizon); so x will send an update k only if that minimum changes.

2.3.4 Sending Updates

The Send-Update procedure (shown in Figure 4) is called every Hello interval to send a Hello packet to each neighbor, and checks which updates should be sent to which neighbors. Send-Summary procedure (shown in Figure 4) is called after successfully exchanging three hello packets with a neighbor. The complete algorithm is shown in the Appendix A.

```
Procedure Send-Updates ()
    // for each neighbor
    for each k in Nx
        new UPDATE (U^k)
        for each destination j
            // if neighbor k is affected by
            // changes in this entry
            if(k ∈ Nxj)
                U^k_j = FindMin (Rxj ,k)
                Add U^k_j to U^k
                Nxj = Nxj – k
            fi
        rof
        sort U^k
        Send U^k to k
    rof
end

Procedure Send-Summary (k)
    new UPDATE (U^k)
    for each destination j
        // Find shortest path not through k
        U^k_j =FindMin(Rxj ,k)
        Add U^k_j to U^k
    rof
    // sort the entries in the update by
    // distance
    sort U^k
    send U^k to k
end
```

Figure 4: Send-Update Algorithm

2.3.5 Examples

This section illustrates different features of the protocol using a set of examples. For the sake of simplicity the examples show how the protocol works for only one destination (node J). The table by a node represents the routing table at that node for the destination. Each row in the table gives the destination - next hop – hop count – second-to-last (in the same order). The arrows next to the links indicate the direction of update messages

Example 1

Consider the network shown in Figure 5. The network topology is stable until node J gets within range of node A. Nodes J and A discover each other through the order wire agents, and a wireless link is initiated between the two nodes. When the routing agents at the two nodes are informed about the new link, they'll try to exchange three Hello packets, over a period of three Hello intervals. If this completes successfully, nodes A and J will exchange the routing information.

Node A adds an entry in its routing table for J, with a distance 1. Node A then informs nodes B and D about J with the next scheduled Hello packet. Nodes B and D inform node C about J. Node C then informs B about the route through D, and

informs D about the route through B. Figure 5 shows the network with the route to J at each node.

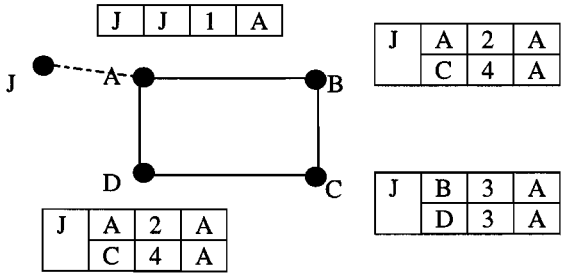


Figure 5: Example 1

Note that node B for example, will not send to A telling it about the route through C, because it can tell that node A is the predecessor node on that path to node J. But if there was another node, say Y (not shown in the figure), on the other side of J, then node B will send to node A telling it about a route to node Y. Node A will be able to detect the loop in that route by checking the second-to-last node.

Now, every node in the system has in its routing table the information about every available path to node J (not containing a loop). If no more changes to the topology happen, the network stabilizes and the nodes just exchange Hello packets.

Example 2

This example shows how the protocol works in case of a link failure. Figure 6 shows the same network shown in the previous example. Say now that node B is sending data to node J (which takes the path B-A-J) and node B moves as shown in the Figure

below. As soon as node B detects the link failure between A and B, it starts using another route to destination J which it already has in its database; routing the data through node C (link failure will be detected after three Hello intervals, unless a lower layer protocol can detect such a failure and provide feedback to the routing agent earlier). Node B also updates its routing table, and informs node C about it. Node C will also update its routing table and inform node D about it with the next Hello packet. The idea from this example is to show that the data would still be routed correctly before the network converges. Figure 6 shows the routing entries at each node for destination J after the network converges.

Note that the amount of data packets dropped is minimal, because the routers keep multiple paths to a destination so the nodes can make use an alternative routes even before the network converges.

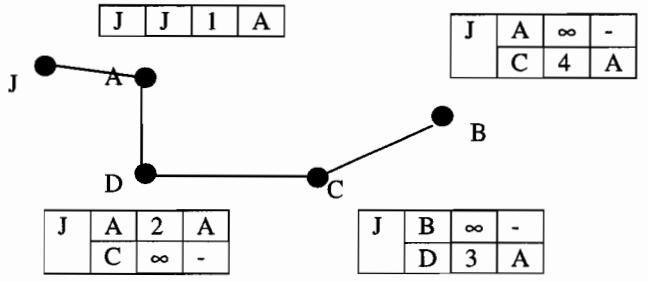


Figure 6: Example 2

Example 3

Consider a five-node network shown in Figure 7. The network is stable, the routing entries for destination J are shown by the nodes. As long as the network topology is stable, no routing updates go through the network, except for Hello packets, which would ensure the connectivity through the network, and detect any change in the topology.

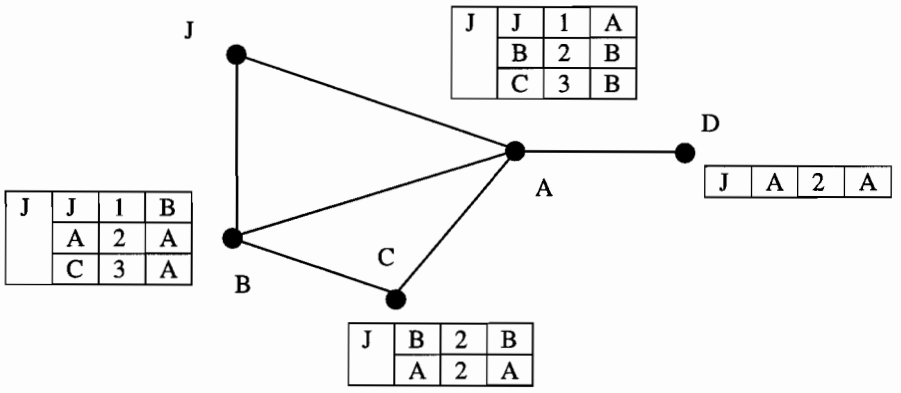


Figure 7: Example 3

When the link (A, J) fails, nodes A and J send update messages to their neighboring nodes as in Figure 8. We show here the updates about J. Node A sends updates to its neighbors D, B and C. Updates to D and C provide an alternate route while A is forced to report an infinite distance for destination J in its update to B. Node C updates it routing table, and sends update to node B with a infinite distance to destination J.

Though Node B has made changes to its table, it need not inform any of its neighbors about this change because the change does not affect its shortest path to destination J. Figure 8 shows the network with the routes to destination J at each node, after the network converges.

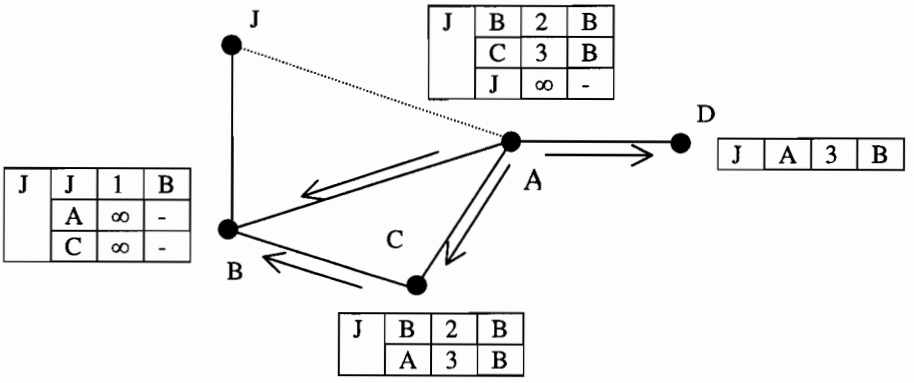


Figure 8: Example 3 (continued)

Chapter Three: Performance Comparisons

3.0 Introduction

In this chapter we discuss the performance measurements and behavior of the proposed protocol when varying the Hello interval, communication patterns, movement patterns, and topology configuration algorithms. We also compare its performance with other wireless routing protocols over a set of different configurations. We first describe the model used, then discuss the behavior of the protocol and compare it with other protocols.

3.1 Model Description

The performance measurement simulations were carried out on a model built by CMU [22] (The CMU Monarch Project's Wireless and Mobility Extensions to ns) which extended *ns* [26] to model a wireless ad hoc network, and provide a quantitative analysis comparing the performance of a variety of wireless routing protocols: DSR, DSDV, AODV, and TORA. We changed the channel model. In the original simulation the channel was originally modeled as a shared media, we changed it to represent a non-shared media to accurately model the nature of RDRN channel. We also used different topology configuration algorithms and limited the number of channels available to make the model more realistic. CMU [22] reported that DSR has the best performance (highest throughput and least overhead), DSDV has the worst performance at low load, and TORA has the worst performance at high

load. We compare the performance of WMRP, DSR, and DSDV (covering periodic advertisements and on demand design choices) on the RDRN channel model in terms of overhead and packet delivery ratio. Packet delivery ratio describes the overall throughput in the network and the loss rate which characterizes the completeness and correctness of the routing protocol. Routing overhead measures the scalability and efficiency of the protocol.

3.1.1 Movement and Communication Patterns

The simulations use a set of 50 nodes that move according to a model called "random waypoint" in a space of 1500mx300m. At the beginning of the simulation, each node starts at a random position, remains stationary for a pause time, then picks a random destination and moves with constant velocity toward the destination (It then again pauses for a pause time, and so on). The node speed is uniformly distributed between 0 and 20m/s (average speed 10m/s). Seven different pause times are used: 0, 30, 60, 120, 300, 600, and 900 seconds. A pause time 0 represents continuous motion, and a pause time 900 (the duration of the simulation) represents no motion. We used 2 different movement patterns (CMU study used 10 different movement patterns). We also vary the communication patterns by changing the number of CBR sources (10, 20, and 30), and one more communication pattern with an average of 20 CBR sources with varying the source-destination pairs. (CMU study used a set CBR communication pattern where the same sources transmit for the whole duration of the simulation to the same destinations. We created one more pattern where a source

transmits to a certain destination for 60 seconds, then stops and another source starts keeping an average of 20 CBR sources at any point of time). Each CBR transmits at a rate of 4 packets/sec (packetsize 512 bytes). CBR sources were used to ensure a uniform load (in the number of packets sent and the time at which a packet is originated) when using different routing protocols to allow direct comparison between them.

3.1.2 Address Resolution and Medium Access Control

Since routing protocols operate at the network layer using IP addresses, an implementation of ARP was included in the simulation and used to resolve IP addresses to link layer addresses.

The model also implements the complete IEEE 802.11 standard Medium Access Control (MAC) protocol. A Unicast packet is preceded by a Request-to-Send/Clear-to-Send (RTS/CTS). A correctly received unicast packet is acknowledged by the receiver. The sender retransmits the packet a limited number of times until it receives an ACK. Broadcast packets are sent only when virtual and physical carrier sense that the medium is clear. The MAC layer also provides a link breakage feedback which indicates that it could not forward the packet to its next hop.

3.1.3 Physical Layer Model

The physical layer uses a model that attenuates the power of a signal as $1/r^2$ at short distances (less than 100m), and $1/r^4$ for longer distances. As mentioned earlier, we changed the model to make it a non-shared media and provide multiple channels

between nodes. We used both a distributed and global topology configuration algorithms. In a distributed algorithm, a node would know the position of other nodes that are within range and it would connect to the nearest neighbors with a maximum number of links (This is the current implementation of RDRN topology configuration algorithm). We had to set the maximum number of links to 16 to keep the network connected (using 4 links is more realistic but that would result in a disconnected network).

3.2 Simulation Results

To validate our simulations, we performed a series of tests using small networks and checking the information kept at each node manually. We also validated the results reported by CMU by rerunning some of the scenarios using the original channel model and checking the checking the results.

3.2.2 Using RDRN Topology Configuration Algorithm

This section describes the results using a distributed topology configuration algorithm, connecting each node to its nearest neighbors while limiting the maximum number of links to a node to 16.

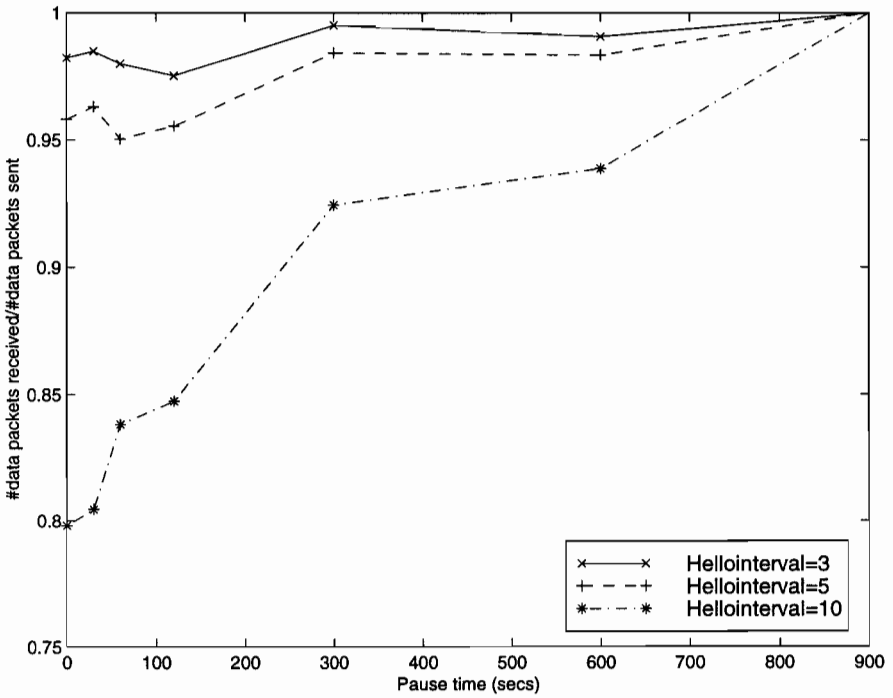


Figure 9: Comparing the fraction of application data packets successfully delivered when changing the Hello interval

Figure 9 and Figure 10 show the throughput and overhead, respectively, for three different Hello intervals using 20 CBR sources. The routing overhead decreases as we increase the Hello interval, but the throughput decreases. When the pause time decreases (mobility rate increases), the overhead increases, and throughput decreases for all three Hello intervals, which is expected. Note also that the performance degrades at low pause times (higher mobility) when using a Hello interval of 10 seconds.

Based on these results, we set our Hello interval to 3 seconds and we used it to compare to the performance of the other protocols (in this section).

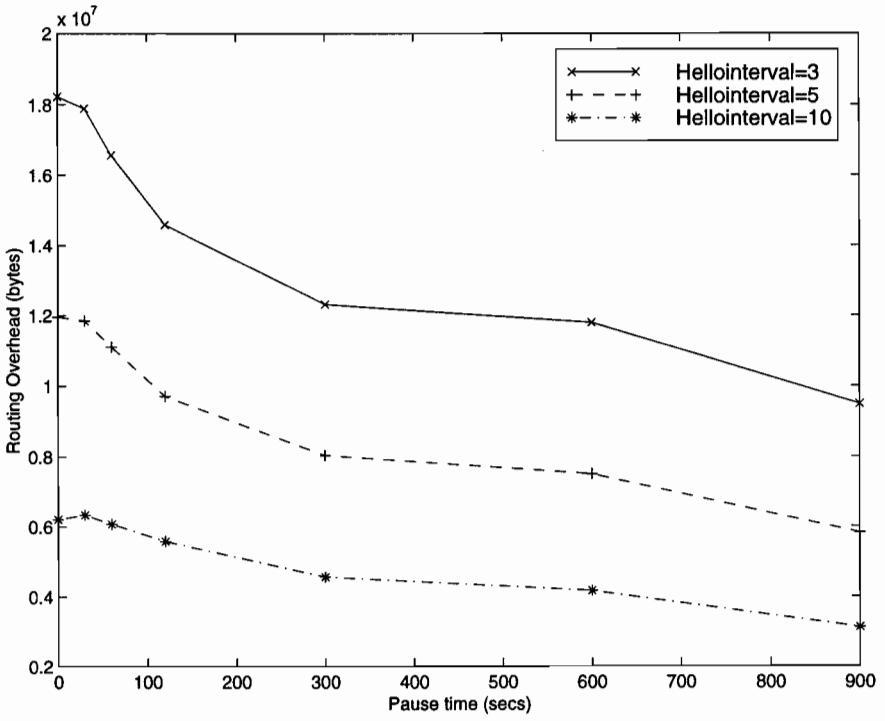


Figure 10: Comparing the routing overhead (bytes) when changing the Hello interval

The routing overhead is reported in terms of number of bytes, not as a percentage. This is so because it is difficult to calculate the capacity of the network due to its dynamic nature.

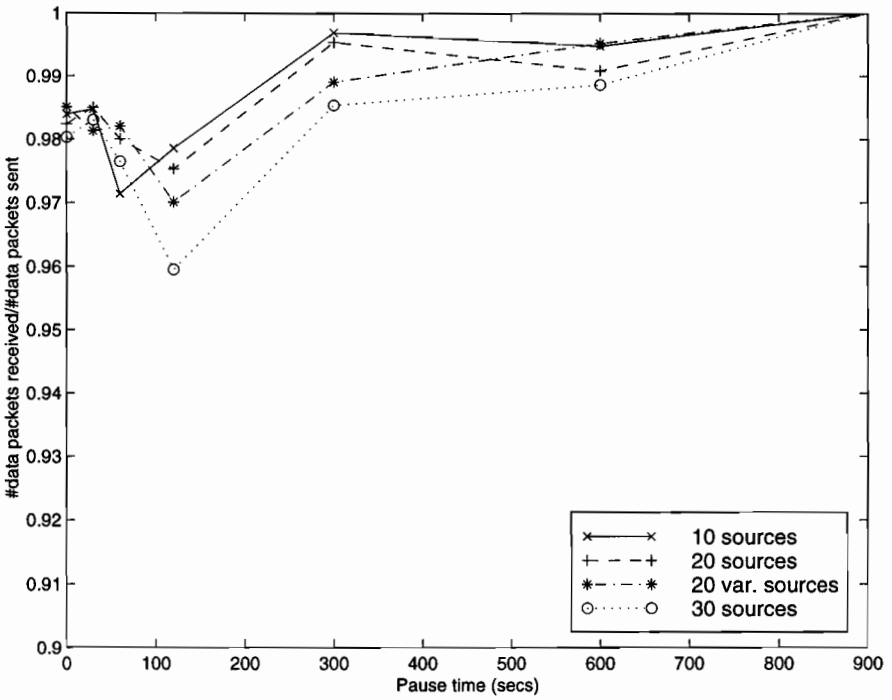


Figure 11: Comparing the fraction of application data packets successfully delivered when changing the load

Figure 11 and Figure 12 show the throughput and the routing overhead, respectively, when changing the load over the network. The Hello interval was set to 3 seconds, and four different communication patterns were used: 10, 20, 30 CBR sources (that is 40, 80, and 120 data packets/sec), and a 20 CBR source communication pattern varying the source destination pairs through the duration of the simulation, but keeping 20 sources at any point of time. The results show that the throughput as well as the overhead does not change when the load over the network is altered.

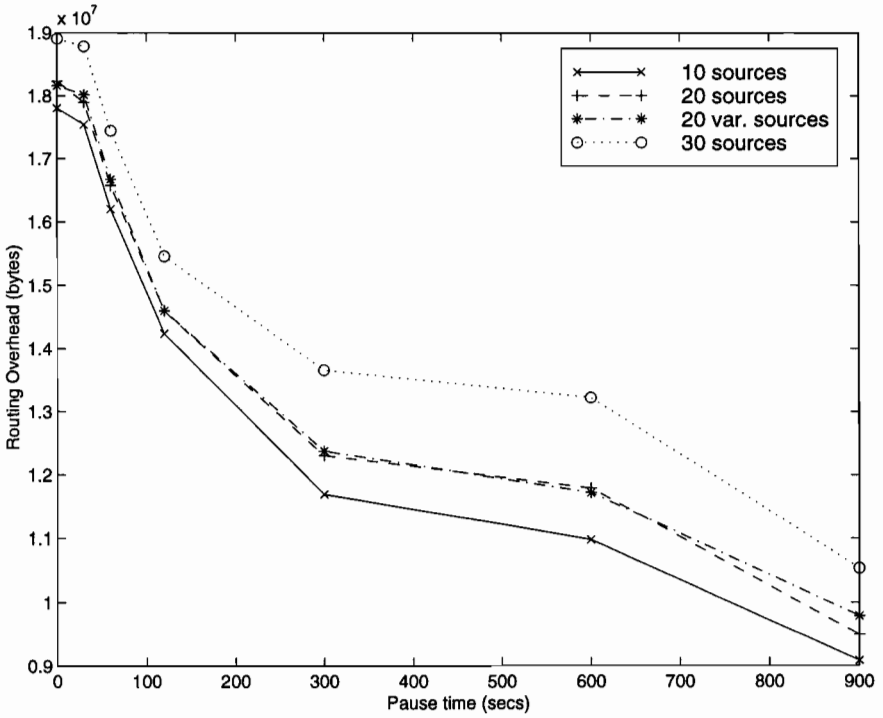


Figure 12: Comparing the routing overhead (bytes) when changing the load

In the next set of experiments, we compare the performance of WMRP to that of DSR, and DSDV using two different communication patterns. Figure 13 and Figure 14 show the throughput and routing overhead respectively on a traffic load of 20 sources. All the protocols deliver a greater percentage of the originated data packets when there is little node mobility (large pause time), converging to 100% delivery when there is no mobility. Both WMRP and DSR perform particularly well, delivering over 95% of the data packets regardless of the mobility rate while DSDV provides the least packet delivery ratio at high mobility rates (about 85% for pause time less than 120 seconds).

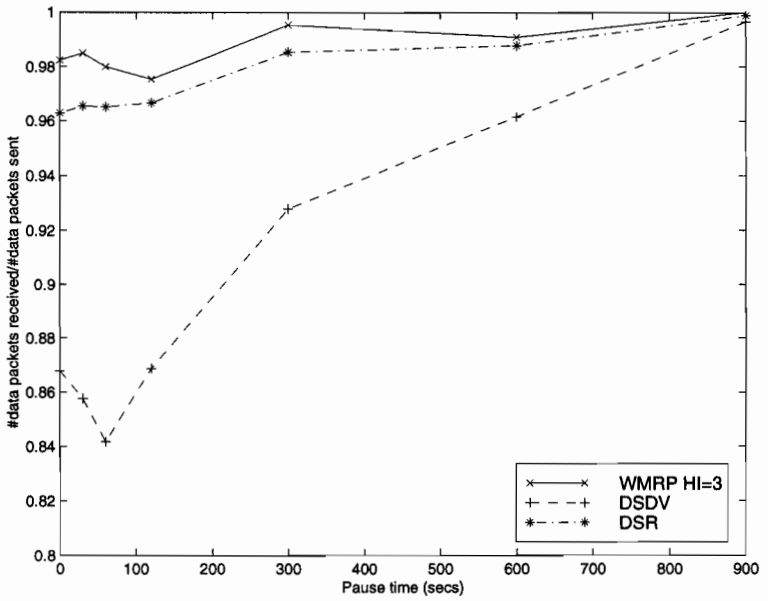


Figure 13: Comparing the fraction of application data packets successfully delivered for three routing protocols (20 CBR sources)

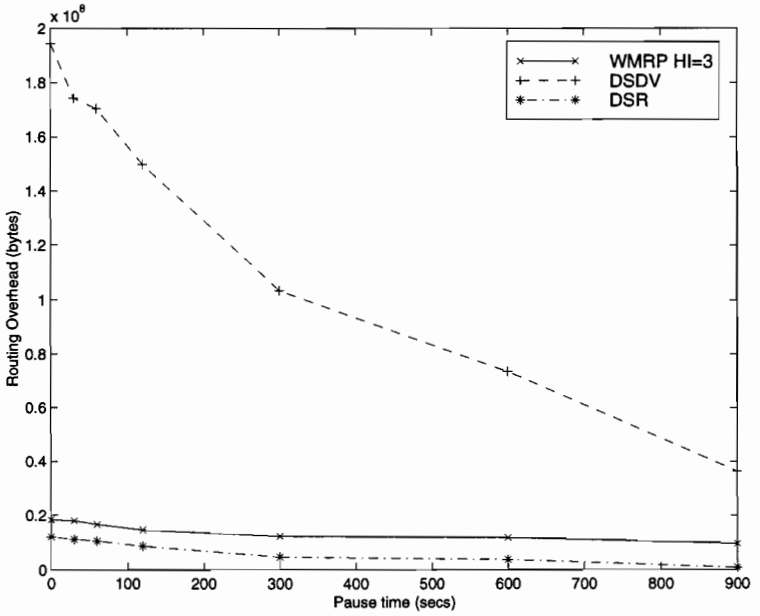


Figure 14: Comparing the routing overhead for three routing protocols

A closer look shows that while WMRP has a higher overhead than DSR, it also provides a higher throughput. Because DSR is an on demand protocol, its overhead drops as the mobility rate drops, while its overhead is closer to the overhead of WMRP at high mobility rates.

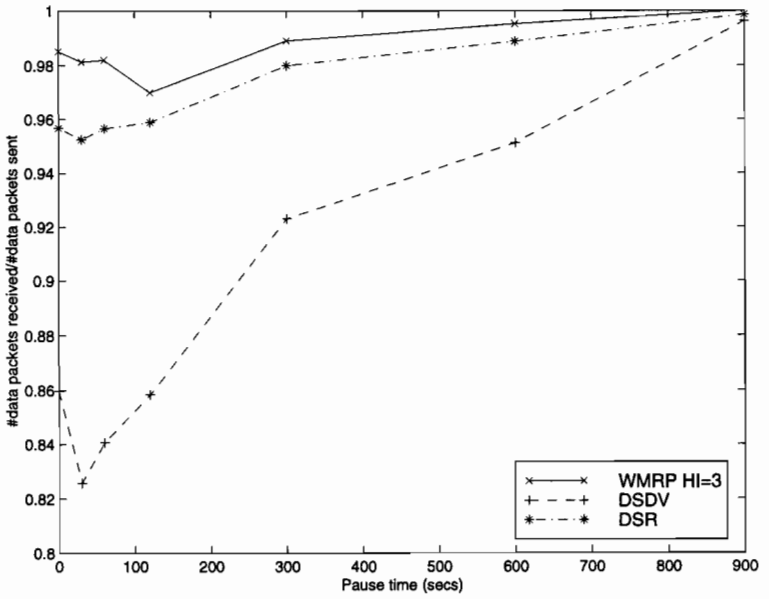


Figure 15: Comparing the fraction of application data packets successfully delivered for three routing protocols (20 var. CBR sources)

Figure 15 and Figure 16 show the throughput and routing overhead respectively on a traffic load of 20 sources and varying the source destination pairs. Again, all of them perform well when there is little node mobility (large pause time), with both WMRP and DSR performing particularly well even at high mobility rates.

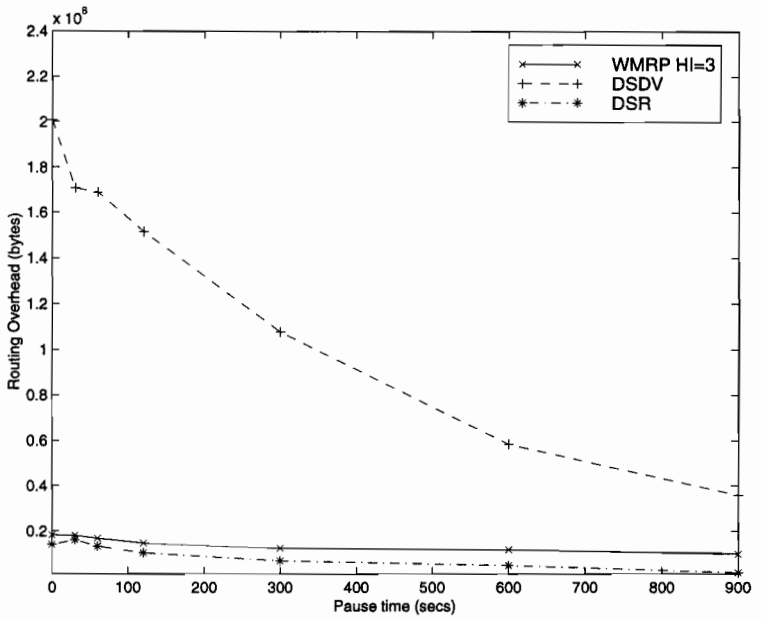


Figure 16: Comparing the routing overhead for three routing protocols (Using 20 var. CBR sources)

A closer look shows that the overhead of DSR slightly increased and its throughput slightly decreased at high mobility rates, compared to the simulation where we had the same set of source destination pairs for the whole duration of the simulation.

Figure 17 shows a comparison of the routing overhead for the three routing protocols in terms of packets. The graph shows that packet overhead in WMRP is more than that observed in other protocols. However, the overhead in terms of bytes is less than that of DSDV, and almost equal to that of DSR. This is attributed to the small size of the Hello packet (2 bytes + IP Header Length = 22 bytes). It can also be seen, in case of WMRP, that the number of routing packets is constant when the mobility rate

changes; while the number of routing packets increases as the mobility rate increases for DSDV, and DSR.

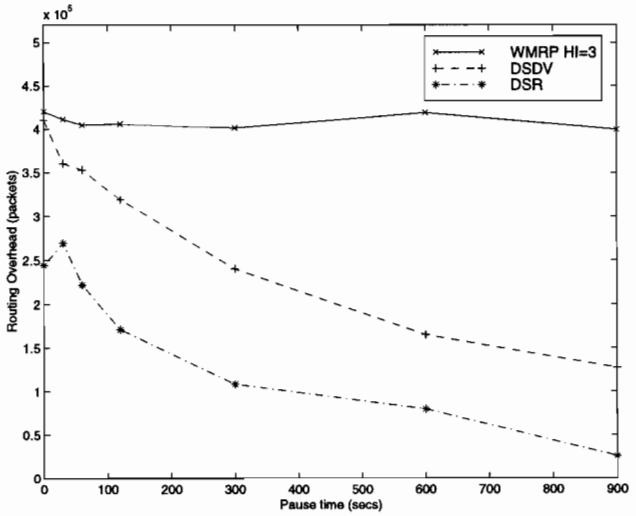


Figure 17: Comparing the routing overhead in packets for three routing protocols (Using 20 var. CBR sources)

The table below shows how the packets were lost in the network for the three protocols on one of the movement patterns using the 0 pause time and 20 CBR sources with varying the source-destination pairs.

Protocol	Total Packets Dropped	No Route	TTL Expired	ARP Full	Timeout	IFQ Full	Simula-tion End
WMRP-HI3	732	78	653	0	0	0	0
DSR	2463	2430	0	32	0	0	0
DSDV	8344	8020	0	321	0	0	2

3.2.3 Using Global Topology Configuration Algorithm

This section describes the results using a shortest path between all pairs of vertices [23] algorithm and limiting the number of links to a node to 4. While this algorithm provides more realistic measures by limiting the number of connections to 4, it uses the global knowledge of each node in the network and provides a connected topology (the algorithm does not guarantee 100% connectivity when limiting the number of connections to 4).

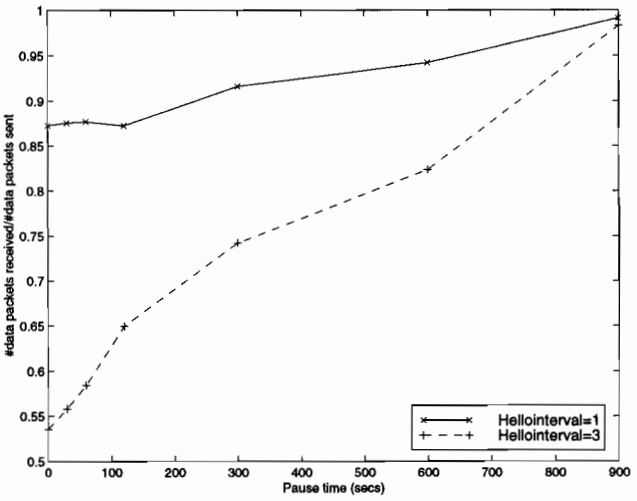


Figure 18: Comparing the fraction of data packets successfully delivered when varying the Hello interval (using 20 sources, global topology configuration)

Figure 18 and Figure 19 show the throughput and overhead, respectively, for two different Hello intervals using 20 CBR sources. As noted in the previous section, the

routing overhead decreases as we increase the Hello interval, but the throughput decreases. Note also that the performance degrades when using a 3 second Hello interval although a 3 second Hello interval provided very good performance using the distributed topology configuration. This is due to less route availability when using fewer number of links, and more frequent changes in the topology that even when multi-path are available, none might be available when the topology changes.

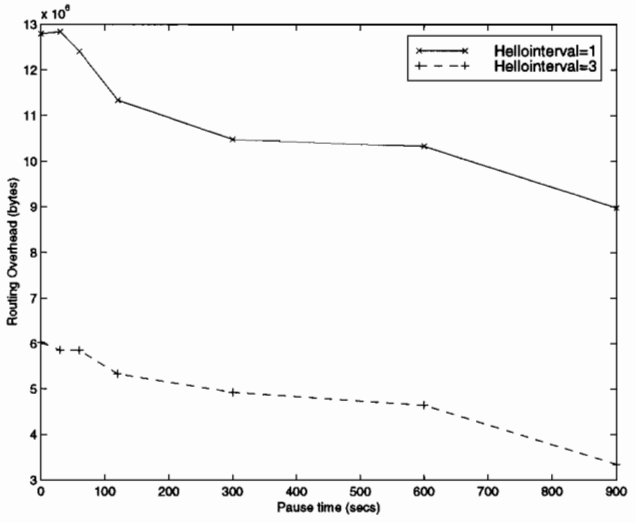


Figure 19: Comparing the routing overhead (bytes) when changing the Hello interval (using 20 sources, global topology configuration)

Also if we compare the overhead in Figure 19 to that in Figure 10 when we used the distributed topology configuration algorithm, we would expect the overhead to drop by an order of four because a node will only report changes to four neighbors instead of sixteen. But the routing overhead (when the protocol converges) is pretty much the

same due to the major changes in the topology using the global topology configuration in an attempt to keep the network connected.

Figure 20 shows the throughput when varying the load. The graph shows the fraction of data packets successfully delivered using 10, 20, and 20 variable CBR sources (20 variable CBR sources is the communication pattern with different source destination pairs). The throughput decreases when the load increases (or when varying the source destination pair) due to imperfections in the topology configuration algorithm used when cutting down the number of links to 4. Figure 11 shows how the throughput stays the same when varying the load (when we used a completely connected network with 16 links).

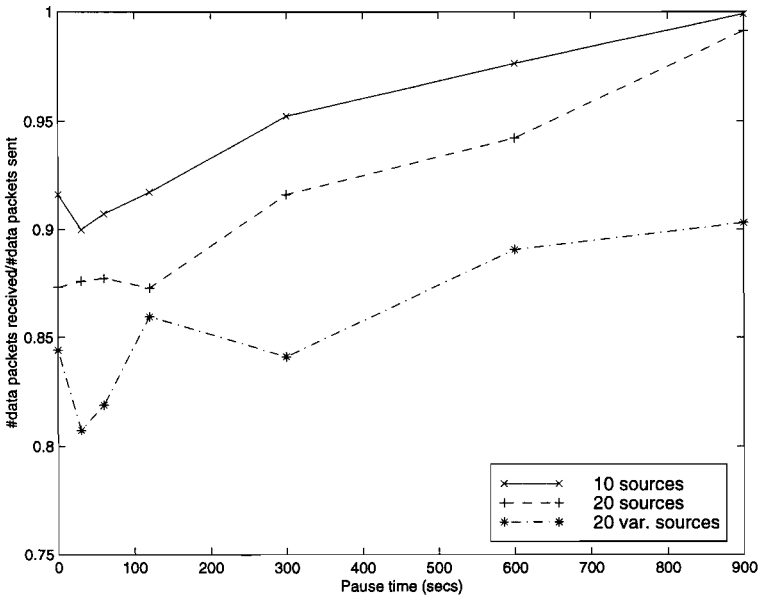


Figure 20: Comparing the fraction of application data packets successfully delivered when changing the load (HI =1 sec, global topology configuration)

Figure 21 and Figure 22 compare the throughput and the routing overhead respectively for the three routing protocols (WMRP, DSR, and DSDV) using 20 CBR sources with variable source destination pairs. All the protocols deliver a higher percentage for small node mobility rates (larger pause times) but only converge to 90% delivery ratio due to imperfections in the topology configuration algorithm. DSDV fails to converge at high node mobility rates and has the highest overhead. WMRP still converges and provides a good packet delivery ratio at high node mobility rates. DSR provides the best packet delivery ratio due to the on demand nature of the protocol which allows the protocol to converge faster when there is a major change in the topology.

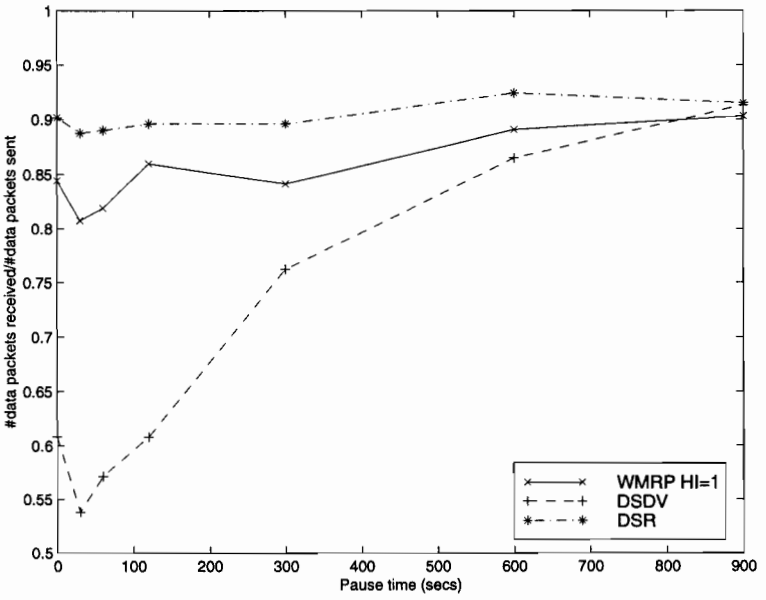


Figure 21: Comparing the fraction of application data packets successfully delivered for the routing protcols (Using 20 var. CBR sources, global topology configuration)

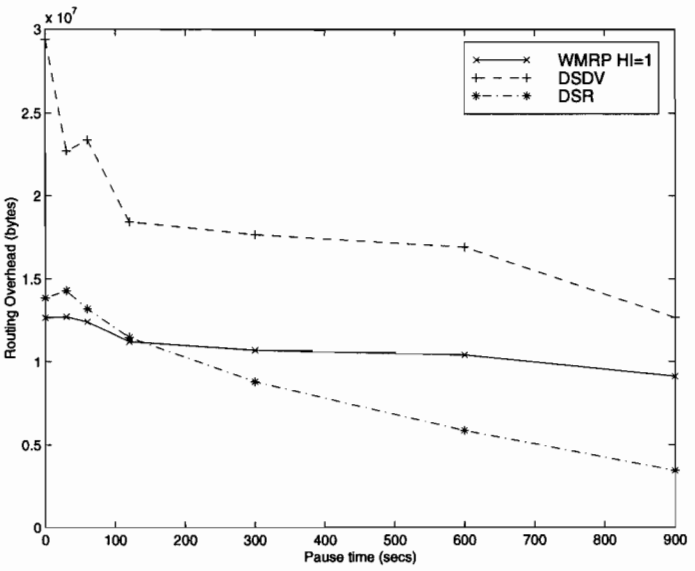


Figure 22: Comparing the routing overhead (bytes) for three routing protocols (Using 20 var. CBR sources, global topology configuration)

If we compare the routing overhead in Figure 22 (using a global topology configuration algorithm and four links) to Figure 16 (using a distributed topology configuration algorithm and sixteen links), we notice that :

1. DSDV's overhead dropped by a factor of four which is expected when the number of links drop by a factor of four. But that is also reflected on its throughput making it deliver less ratio of data packets.
2. DSR's overhead is pretty much the same because of the on demand ratio of DSR making it send more requests as the frequency topology changes increase.
3. DSR's overhead is slightly higher than WMRP's overhead at high mobility rates when using a global topology configuration algorithm and four links.

Figure 23 shows the a comparison in the average packet delay between DSR and WMRP using the global topology configuration algorithm and 20 CBR sources with variable source-destination pairs. The packet delay is measured from the time a packet is sent at the originator until it reaches its final destination.

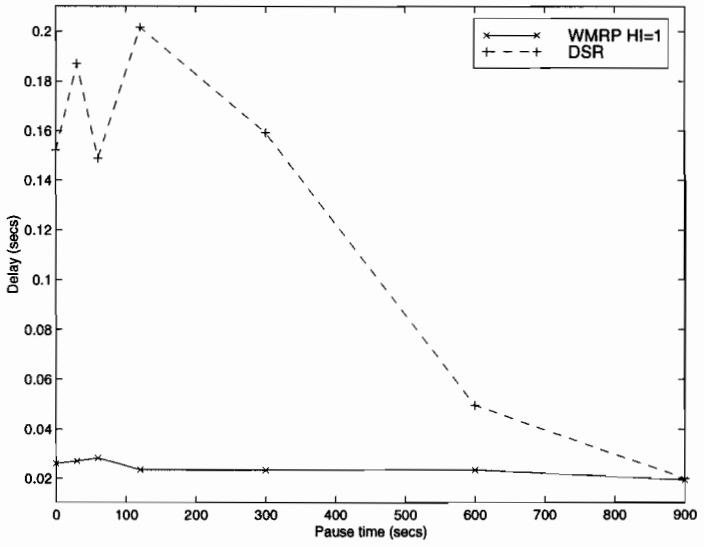


Figure 23: Comparing the time delay for WMRP and DSR (Using 20 var. CBR sources, global topology configuration)

As we can see from the figure, DSR has an order of magnitude more delay at high mobility rates which is due to the on demand nature of DSR where packets can be queued at the originator until a route is found to the packet's destination.

Chapter four gives a summary of the observations, conclusions and discusses some future work.

Chapter Four: Conclusions and Future Work

Conclusions

We have presented a wireless multi-path routing protocol (WMRP) that makes use of the non-shared wireless media and provides multiple paths per destination. The advantage of WMRP is better throughput and reliability. Routers can immediately switch to the next best path if the preferred path becomes unavailable. The information in the routing tables is similar to what is found in current distance vector (Bellman-Ford) algorithms, but includes a second-to last-hop information to prevent count-to-infinity problem. The protocol also enforces a hold-off time before new neighbors exchange their routing information to ensure reliability.

We provided a realistic quantitative analysis that was used to find the optimal Hello interval, compare the behavior of the protocol using a variety of communication patterns, and compare the performance of the protocol to that of other wireless routing protocols. Simulation results show that WMRP is an excellent alternative for wireless networks providing very good throughput and a moderate overhead. When compared to other wireless routing protocols we found that:

1. WMRP provides better throughput and less overhead than DSDV at all mobility rates, and all frequencies of topology changes.

2. WMRP provides better throughput (and slightly higher overhead) than DSR at all mobility rates in a topology that provides multiple paths and less frequent changes.
3. DSR provides better throughput than WMRP in a more frequently changing topology at all mobility rates, and slightly higher overhead at high mobility rates.
4. WMRP provides an order of magnitude less packet delay time than DSR.

Also note that DSR is a source routing protocol and for security reasons too many routers do not allow source routing packet to flow through them; which makes a network using DSR disconnected from the rest of the world. This adds one more reason to use WMRP not DSR other than less delay and comparable throughput and overhead.

Future Work

The physical radio characteristics of each mobile node's network interface, such as antenna gain, transmit power, receiver sensitivity, propagation delay, and power attenuation, were chosen to approximate the Lucent WaveLAN radio operating in the 1-2GHz band. We need to characterize the RDRN channel in such terms, and use those figures to provide a more realistic RDRN Model.

We understand the fact that we need a better topology configuration algorithms and that lots of work is still to be done to figure out the best algorithm to be used in RDRN.

All the simulations used the same node density and the same average node speed. A more comprehensive evaluation would require creating the environment and running the simulations for different combinations of node density and node speed.

Last, but not least, an implementation of the protocol in a linux kernel should be an easy first step towards its practical realization.

References

1. 'Routing in the Internet', *Christian Huitema.* Prentice Hall, 1995.
2. 'Distributed Scalable Routing Based on Link State Vectors', *Jochen Behrens, J.J. Garcia-Luna-Aceves.*
3. 'Routing Basics', http://www.cisco.com/univercd/cc/td/doc/cisintwk/ito_doc/55171.htm.
4. 'RFC 1058: Routing Information Protocol', *C. Hedric.*
5. 'TCP/IP and related Protocols', Uyless Black. McGraw Hill, 1992.
6. 'TCP/IP Architecture, Protocols and Implementations'. Sidnie Feit. McGraw Hill, 1996.
7. 'An efficient Routing Protocol for Wireless Networks', *S. Murthy and J.J. Garcia-Luna-Aceves.*
8. 'Wireless Internet Gateways (WINGS)', *J.J. Garcia-Luna-Aceves et al.*
9. 'A Distributed Routing Algorithm for Mobile Wireless Networks', *M. S. Corson, A. Ephremides.* Wireless Networks Feb. 95
10. ' A Loop-free Path-Finding Algorithm: Specification, Verification and Complexity', *J.J. Garcia-Luna-Aceves, and S. Murthy.*
11. 'Distributed, Scalable Routing Based on Link-State Vectors', *S. Murthy and J.J. Garcia-Luna-Aceves.*
12. 'A Loop-Free Extended Bellman-Ford Routing Protocol without the Bouncing Effect', *C. Cheng, R. Riley, and J.J. Garcia-Luna-Aceves.* ACM Sigcom Sept. 89.

13. 'A New Responsive Distributed Shortest-Path Routing Algorithm', *B. Rajagopalan and M. Faiman.* ACM Sigcom 89.

14. 'Performance Comparison of Routing Protocols using MaRS: Distance-Vector versus Link-State', *A. U. Shankar et al.* Performance Evaluation Review, June 92.

15. 'An Internet MANET Encapsulation Protocol (IMEP) Specification', *M. S. Corson, S. Papademetriou, P. Papadopoulos, V. Park, A. Qayyam.* Internet-Draft, Aug. 98. Work in progress.

16. 'Temporally-Ordered Routing Algorithm (TORA) Version 1 Functional Specification'. *V. Park, S. Corson.* Internet-Draft. Nov. 97, Work in progress.

17. 'Ad Hoc On Demand Distance Vector (AODV) Routing', *C. Perkins, E. Royer.* Internet-Draft, Nov. 1998. Work in progress.

18. 'The Dynamic Source Routing Protocol for Mobile Ad Hoc Networks', *J. Broch, D. Johnson, D. Maltz.* Internet-Draft, March 1998. Work in Progress.

19. 'The Zone Routing Protocol (ZRP) for Ad Hoc Networks', *Z. Haas, M. Pearlman.* Internet-Draft, Nov, 97. Work in Progress.

20. 'Cluster Based Routing Protocol (CBRP) Functional Specification', *M. Jiang, J. Li, Y.C.Tay.* Internet-Draft. Work in Progress.

21. 'Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers', *C. Perkins, and P. Bhagwat.* Sigcom ' 94 Conference on Communications Architectures.

22. 'A Performance Comparison of Multi-Hop Wireless Ad Hoc Network Routing Protocols', *J. Broch, D. Maltz, D. Johnson, Y. Hu, J. Jetcheva.* ACM/IEEE

International Conference on Mobile computing and Networking (Mobicom'98) Oct. 98.

23. 'Graph Theory with Applications to Engineering and Computer Science', *Deo, Narsingh*. Prentice-Hall, Englewood Cliffs, NJ, 1974.
24. 'RFC 2328: OSPF Version 2', *J. Moy*. April 1998.
25. 'Information Technology, Telecommunication and Information change between systems, Intermediate System-to-Intermediate System Routing Information Exchange Protocol for Use in Conjunction With ISO 8473', ISO 10589, 1990.
26. 'ns notes and documentation', *Kevin Fall and Kannan Varadhan*. The VINT project, November 1997. http://www-mash.cs.berkeley.edu/ns/.

Appendix A: Algorithm

```
Program main
    do
        wait HI until
            ConnectMsg (k)
                Connect-To (k)
            LinkDownMsg (k)
                Lost-Link (k)
            Recv-Packet (p)
                Recv (p)
            Timeout-Event (e)
                Timeout-Handler (e)
        tiaw
    forever
end

Procedure Timeout-Handler (e)
    for each k in Bx
        if (B^k_x.timeout == e)
            Bx = Bx - k
            return
        fi
    rof
    for each k in Nx
        if (N^k_x.timeout == e)
            Lost-Link (k)
            return
        fi
    rof
    Send B_HELLO   //Broadcast
    Send-Updates ()
end

Procedure Forward-Packet (p)
    Rxj = Find in Table (p.dst)
    min = infinity
    for each k in Rxj
        if(k(R^k_xj) == p.src ) continue
        if(D(R^k_xj) < min)
            min = D(R^k_xj)
            n   = k(R^k_xj)
        fi
    rof
    send p to n
end

Procedure Recv (p)
    if (p.port == ROUTER_PORT)
        Process-Update (p)
    else
        Forward-Packet(p)
    fi
end

Procedure Lost-Link (k)
    for each destination j
        Update (R^k_xj, infinity)
    rof
end

Procedure Process-Update (p)
    switch(p.type )
        case B_HELLO :
          if( k ∉ Bx) return
          B^k_x.ctr ++
          if(B^k_x.ctr == 3)
            Add-Nbr (k)
            Send-Summary (k)
          fi
        return
        esac
        case N_HELLO :
          if( k ∉ Nx )
            Add-Nbr (k)
            Send-Summary (k)
          else
            cancel (N^k_x.timeout )
            N^k_x.timeout = schedule (3*HI)
          fi
          break
        esac
        case SMRY :
          if( k ∉ Nx )
            Add-Nbr (k)
            Send-Summary (k)
          else
            if ( now - N^k_x.addtime > HI )
                Send-Summary (k)
            fi
          fi
        esac
```

```
    hctiws
    for each U^k_j in p
        Verify-Update (U^k_j)
    rof
end

Procedure Verify-Update (U^k_j)
    if (U^k_j D == INF )
        Update (R^k_xj , U^k_j)
        return
    fi
    if ( (U^k_j D == 0 && k ∈ N_x) ||
          U^k_j .D == R^k_xp.D )
        Update (R^k_xj , U^k_j)
        return
    else
        U^k_j .D = INF
        Update (R^k_xj , U^k_j)
        return
    fi
end

Procedure Update (R^k_xj , U^k_j )
    for each n in N_x
        min_before[n]=Find-Min (R_xj ,n)
    rof
    R^k_xj.D = U^k_j.D + 1
    R^k_xj.P = U^k_j.P
    for each k in N_x
        min_after = Find-Min (R_xj ,n)
        if(min_after≠min_before[n] ||
            min_before[n]==k)
                N_xj = N_xj + k
        fi
    rof
end

Procedure Send-Summary (k)
    new UPDATE (U^k)
    for each destination j
        U^k_j =FindMin(R_xj ,k)
        Add U^k_j to U^k
    rof
    sort  U^k
    send U^k to k
end

Procedure Send-Updates ()
    for each k in N_x
        new UPDATE (U^k)
        for each destination j
            if(k ∈ N_xj)
                U^k_j = FindMin (R_xj ,k)
                Add U^k_j to U^k
                N_xj = N_xj – k
            fi
        rof
        sort  U^k
        Send U^k  to k
    rof
```