Baltzer Journals

An Efficient Routing Protocol for Wireless Networks

Shree Murthy and J.J. Garcia-Luna-Aceves
Computer Engineering, University of California,
Santa Cruz, CA 95064

We present the wireless routing protocol (WRP). In WRP, routing nodes communicate the distance and second-to-last hop for each destination. WRP reduces the number of cases in which a temporary routing loop can occur, which accounts for its fast convergence properties. A detailed proof of correctness is presented and its performance is compared by simulation with the performance of the distributed Bellman-Ford algorithm (DBF), DUAL (a loop-free distance-vector algorithm) and an ideal link-state algorithm (ILS), which represent the state of the art of internet routing. The simulation results indicate that WRP is the most efficient of the alternatives analyzed.

Keywords: routing, packet radio, wireless, distance-vector, link-state, shortest-path

1 – Introduction

The routing protocols used in multihop packet-radio networks implemented in the past [2, 3, 11] were based on shortest-path routing algorithms that have been typically based on the distributed Bellman-Ford algorithm (DBF) [4]. According to DBF, a routing node knows the length of the shortest path from each neighbor to every network destination and this information is used to compute the shortest path and successor in the path to each destination. An update message contains a vector of one or more entries, each of which specifies as a minimum, the distance to a given destination. A major performance problem with DBF is that it takes a very long time to update the routing tables of network nodes after network partitions, node failures, or increase in network congestion. This performance problem of DBF stems from the fact that it has no inherent mechanism to determine when a network node should stop incrementing its distance to a given destination. This problem is usually called the *counting-to-infinity* problem.

The counting-to-infinity problem is overcome in one of three ways in existing

This work was supported in part by the Advanced Research Projects Agency (ARPA) under contract F19628-93-C-0175 and by the Office of Naval Research under Contract No. N-00014-92-J-1807.

internet routing protocols. OSPF [12] relies on broadcasting complete topology information among routers, and organizes an internet hierarchically to cope with the overhead incurred with topology broadcast. BGP [16] exchanges distance vectors that specify complete paths to destinations. EIGRP [1] uses a loop-free routing algorithm called DUAL [8], which is based on internodal coordination that can span multiple hops; DUAL also eliminates temporary routing loops.

However, there are significant differences between wireless networks and wired internets in which internet routing protocols are used. A wired internet has relatively high bandwidth and topology that changes infrequently; in contrast, wireless networks have mobile nodes and have limited bandwidth for network control. Accordingly, flooding, multihop internodal synchronization and the specification of complete path information would incur too much overhead in a multihop radio network with a dynamic topology. On the other hand, the routing protocols based on DBF or modifications of DBF would take a long time to converge and the frequent topology changes in a wireless network with mobile nodes make the looping problem of DBF unacceptable. Therefore, there is a need for a new routing protocol which is devoid of all these drawbacks.

In the recent past, a number of efforts have been made to address the limitation of DBF and topology broadcast in mobile wireless networks. One such effort is the DSDV protocol [14]. In this protocol, each mobile host, which is a specialized router that periodically advertises its view of the interconnection topology with other mobile hosts within the network to maintain up to date information about the status of the network. Unfortunately, in DSDV a node has to wait until it receives the next update message originated by the destination in order to update its distance-table entry for that destination. This implicit destination-centered synchronization suffers from the same latency problems of DUAL and similar algorithms based on explicit synchronization. Also, DSDV uses both periodic and triggered updates for updating routing information, which could cause excessive communication overhead.

A distributed routing algorithm for mobile wireless networks based on diffusing computations has been proposed by Corson and Ephremides [6]. This protocol relies on the exchange of short control packets forming a *query-reply* process. It also has the ability to maintain multiple paths to a given destination. This is a destination-oriented protocol in which separate versions of the algorithm run independently for each destination. Routing is source-initiated, which means that routes are maintained by those sources which actually desire routes. Even though this algorithm provides multiple paths to the destination, because of the query-based synchronization approach to achieve loop-free paths, the communication complexity could be high.

Recently, a number of distributed shortest-path algorithms have been proposed [5, 7, 9, 10, 15] that utilize information regarding the length and second-to-last hop (predecessor) of the shortest path to each destination to eliminate the counting-to-infinity problem of DBF. We call this type of algorithms as *path-finding algorithms.* According to these algorithms, each node maintains the shortest-path spanning tree reported by its neighbors. A node uses this information along with the cost of adjacent links to generate its own shortest-path spanning tree. An update message exchanged among neighbors consists of a vector of entries that report updates to the sender's spanning tree; each update entry contains a destination identifier, the distance to the destination, and the second-to-last hop of the shortest path to the destination.

Path-finding algorithms are an attractive approach for wireless networks, because they eliminate counting-to-infinity problem. However, these algorithms can still incur temporary loops in the paths specified by the predecessor before they converge; without proper precautions, this can lead to slow convergence, or incur substantial processing if a node is required to update its entire routing table for each input event. To address these problems, we have proposed a path-finding algorithm, PFA, which substantially reduces temporary looping situations [13], and which limits routing table updates to include only that entries affected by a network change.

The rest of this paper describes a wireless routing protocol (WRP) for a packet radio network based on PFA, illustrating the key aspects of the protocol's operation. The following sections show that the protocol is correct (i.e., that it produces correct routing tables within a finite time after topology changes) and compares its performance with that of DBF, DUAL and an ideal link state algorithm (ILS) which uses Dijkstra's shortest path algorithm.

ILS consists of ideal flooding of link-state updates in order to replicate the topology of the network at each router; ideal flooding means that infinite sequence numbers can be used to validate link-state updates, and that all such updates are successfully delivered at every router.

2 – Wireless Routing Protocol

2.1 – Overview

To describe WRP, we model a 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 and is a computing unit involving a processor, local memory and input and output queues with unlimited capacity. In a wireless network, a node has radio connectivity with multiple nodes and a single physical radio link connects a node with many other nodes. However, for the purposes of routing-table updating, a node A can consider another node B to be adjacent (we call such a node a "neighbor") if there is radio connectivity between A and B and A receives update messages from B. Accordingly, we map a physical broadcast link connecting multiple nodes into multiple point-to-point functional links defined for these node paths that consider to be neighbors of each other.

Then, a functional bidirectional link connecting the nodes is assigned a positive weight in each direction. All messages received (transmitted) by a node are put in an input (output) queue and are processed in FIFO order. The communication links in the network are such that all update messages transmitted over an operational link are received in the order in which they were transmitted within a finite time.

A link is assumed to exist between two nodes only if there is radio connectivity between the two nodes and they can exchange update messages reliably with a certain probability of success. When a link fails, the corresponding distance entries in a node's distance and routing tables are marked as infinity. A node failure is modeled as all links incident on that node failing at the same time.

WRP is designed to run on top of the medium-access control protocol of a wireless network. Update messages may be lost or corrupted due to changes in radio connectivity or jamming. Reliable transmission of update messages is implemented by means of retransmissions. After receiving an update message free of errors, a node is required to send a positive acknowledgment (ACK) indicating that it has a good radio connectivity and has processed the update message. Because of the broadcast nature of the radio channel, a node can send a single update message to inform all its neighbors about changes in its routing table; however, each such neighbor sends an ACK to the originator node.

In addition to ACKs, the connectivity can also be ascertained with the receipt of any message from a neighbor (which need not be an update message). To ensure that connectivity with a neighbor still exists when there are no recent transmissions of routing table updates or ACKs, periodic update messages without any routing table changes (null update messages) are sent to the neighbors. The time interval between two such null update messages is the *HelloInterval*.

If a node fails to receive any type of message from a neighbor for a specified amount of time (e.g., three or four times the HelloInterval known as the *RouterDeadInterval*), the node must assume that connectivity with that neighbor has been lost.

2.2 – Information Maintained at Each Node

For the purpose of routing, each node maintains a *distance table*, a *routing table*, a *link-cost table* and a *message retransmission list*.

The distance table of node i is a matrix containing, for each destination j and each neighbor of i (say k), the distance to j (D^i_{jk}) and the predecessor (p^i_{jk}) reported by k.

Procedure Init1
when router i initializes itself
do begin
  set a link state table with costs of adjacent links;
  $N \leftarrow i$; $N_i \leftarrow x \mid l_x^i < \infty$;
  for each $(x \in N_i)$
  do begin
    $N_i \leftarrow N \cup x$; $tag_x^i \leftarrow null$;
    $s_x^i \leftarrow null$; $p_x^i \leftarrow null$; $D_x^i \leftarrow \infty$
  end
  $D_i^i \leftarrow 0$; $s_i^i \leftarrow null$; $p_i^i \leftarrow null$; $tag_i^i \leftarrow correct$
  for each $j \in N$ call **Init2**(x, j)
  for each $(n \in N_i)$ **do** add $(0, i, 0, i)$ to $LIST_i(n)$
  $x \leftarrow$ retransmission time; $y \leftarrow$ hello count;
  $z \leftarrow$ retransmission count;
  call **Send**
end

Procedure Init2(x, j)
begin
  $D_{jx}^i \leftarrow \infty$; $p_{jx}^i \leftarrow null$; $s_{jx}^i \leftarrow null$; $seqno_{jx}^i \leftarrow 0$
end

Procedure Send
begin
  for each $(n \in N_i)$
  do begin
    if $(LIST_i(n)$ is not empty$)$
    then send messages with $LIST_i(n)$ to n
    empty $LIST_i(n)$
  end
end

Procedure Message
when router i receives a message on link (i, k)
begin
  if $(k \notin N_i)$ **do**
  begin
    $N_i \leftarrow N_i \cup k$;
    $l_k^i \leftarrow$ cost of new link;
    if $(k \notin N)$ **begin**
      $N \leftarrow N \cup k$; $tag_k^i \leftarrow null$;
      $D_k^i \leftarrow \infty$; $p_k^i \leftarrow null$; $s_k^i \leftarrow null$;
      for each $x \in N_i$ **do** call **Init2**(x, k)
    end
    for each (i, k, l_k^i) **do**
      send update$(0, k, D_k^i, p_k^i)$
  end
  reset HelloTimer;
  for each entry $(u_j^k, j, RD_j^k, rp_j^k) \mid i \neq j$
  do begin
    if $(j \notin N)$
    then begin
      if $(RD_j^k = \infty)$ **then** delete entry
      else begin
        $N \leftarrow N \cup j$;
        for each entry $x \in N_i$ call **Init2**(x, j)
        $tag_j^i \leftarrow null$; call **DT**
      end
    end
    else begin
      $tag_j^i \leftarrow null$;
    end
  end
  for each entry $(u_j^k, j, RD_j^k, rp_j^k)$ left $\mid i \neq j$
  do case of u_j^k
    0: call **Update**(j, k)
    1: call **ACK**(j, k)
  end
  call **Send**
end

Procedure Create_RList$(seqno)$
begin
  $seqno \leftarrow seqno + 1$; $NeighborSet \leftarrow N_i$
  $bitmap[] \leftarrow 0$; $RetransmissionTimer \leftarrow x$
  add updates to RList
end

Procedure Delete_RList$(seqno)$
begin
  set $bitmap[seqno] \leftarrow 1$; $delete \leftarrow 1$
  for all $n \in N_i$ **begin**
    if $(bitmap[seqno] = 0)$ $delete \leftarrow 0$;
  end
  if $(delete = 1)$ delete RList[seqno] **end**

Procedure Update_RList$(seqno)$
begin
  reset RetransmissionTimer
  send update $RList[seqno]$;
end

Procedure Clean_RList $(seqno)$
begin
  for all entries in $RList$
    delete $RList[seqno]$;
end

Procedure Connectivity
when HelloTimer expires
begin
  $HelloCount[k] \leftarrow HelloCount[k] + 1$;
  if $(HelloCount[k] < y)$ **then**
    reset HelloTimer;
  else begin
    $N_i \leftarrow N_i - k$
    call **Delete_RList**(k)
    $l_k^i \leftarrow \infty$
    $tag_k^i \leftarrow null$
    delete column for k in distance table
    update routing table
  end
end

Procedure TimeOut(i, k)
when RetransmissionTimer expires
begin
  RetransmissionCounter $\leftarrow$ RetransmissionCounter - 1;
  if (RetransmissionCounter $<$ z)
    call **Update_RList**(k)
  else begin
    $N_i \leftarrow N_i - k$
    call **Delete_RList**(k)
    $l_k^i \leftarrow \infty$
    $tag_k^i \leftarrow null$
    delete column for k in distance table
    update routing table
  end
end

Procedure DT
when distance table update has to be done
begin
  $D_{jk}^i \leftarrow l_k^i + D_j^k$; $p_{jk}^i \leftarrow p_j^k$;
(2) **for all** neighbors b
  do begin
    if k is in the path from i to j in
      the distance table through neighbor b
    then $D_{jb}^i \leftarrow D_{kb}^i + D_j^k$; $p_{jb}^i \leftarrow p_j^k$
  end
end

Figure 1: Protocol Specification

```
Procedure ACK(n)
when router i receives an ACK on link (i, k)
begin
        call Delete_RList(n);
        RetransmissionCounter ← z;
end

Procedure Update(i, k)
when router i receives an update on link (i, k)
begin
        send ACK to neighbor k
        RetransmissionCounter ← z;
        RetransmissionTimer ← x;
(0)     begin
              update=0;
              RTEMP^i ← φ;
              DTEMP^{i,b} ← φ for all neighbors b
(1)           for each triplet (j, D^k_j, p^k_j) in V^{k,i}, j ≠ i do
                   call procedure DT
(3)           begin
                   if there are b and j such that
                      (D^i_jb < D^i_j) or ((D^i_jb > D^i_j) and (b = s^i_j))
                   then call RT_Update
              end
(4)           begin if (RTEMP^i ≠ φ) then
                   for each neighbor b do begin
                        for each triplet t = (j, D^i_j, p^i_j) in RTEMP^i
                        do begin
                             if b is not in the path from i to j
                             then DTEMP^{i,b} ← DTEMP^{i,b} ∪ t;
                        end
                        send DTEMP^{i,b} to neighbor b;
                   end
        end
end
```

```
Procedure RT_Update
when routing table has to be updated
begin
   find minimum of the distance entries DT_min
   if (D^i_j s^i_j = DT_min) then ns ← s^i_j
   else ns ← b | {b ∈ N_i and D^i_jb = DT_min};
   x ← j;
   while (D^i_x ns = Min{D^i_xb  ∀  b ∈ N_i}
           and D^i_xns < ∞ and tag^i_x = null)
   do x ← p^i_x ns;
   if (p^i_x ns = i or tag^i_x = correct)
   then tag^i_j ← correct  else tag^i_j ← error
   if (tag^i_j = correct) then begin
     if (D^i_j ≠ DT_min or p^i_j ≠ p^i_j ns) then begin
       seqno ← seqno + 1;
       add (0, j, DT_min, p^i_j ns, seqno) to LIST_i(x)  ∀x ∈ N_i;
       call Clean_RList(seqno)
       call Create_RList(seqno)
     end
     D^i_j ← DT_min; p^i_j ← p^i_j ns; s^i_j ← ns
   end
   else begin
     if(D^i_j < ∞) then begin
       seqno ← seqno + 1;
       add (0, j, ∞, null, seqno) to LIST_i(x)  ∀x ∈ N_i;
       call Clean_RList(seqno)
       call Create_RList(seqno)
     end
     D^i_j ← ∞; p^i_j ← null; s^i_j ← null
   end
end
```

Figure 2: Protocol Specification (Cont..)

The routing table of a node i is a vector with an entry for each known destination j which specifies:

- The destination's identifier
- The distance to the destination (D^i_j)
- The predecessor of the chosen shortest path to j (p^i_j)
- The successor (s^i_j) of the chosen shortest path to j
- A marker (tag^i_j) used to update routing table; it specifies whether the entry corresponds to a simple path (tag^i_j = correct), a loop (tag^i_j = error) or a destination that has not been marked (tag^i_j = null).

The link-cost table of node i lists the cost of relaying information through each neighbor k, and the number of periodic update periods that have elapsed since node i received any error-free messages from k.

The cost of a failed link is considered to be infinity. The way in which costs are assigned to links is beyond the scope of this specification. As an example, the cost of a link could simply be 1 reflecting the hop count, or the addition of the latency over the link plus some constant bias. The cost of the link from i to k (i, k) is denoted by l^i_k.

The message retransmission list (MRL) specifies one or more retransmission entries, where the m^{th} entry consists of the following:

- The sequence number of an update message
- A retransmission counter that is decremented every time node i sends a new update message
- An *ack-required* flag (denoted by a^i_{km}) that specifies whether node k has sent an ACK to the update message represented by the retransmission entry
- The list of updates sent in the update message

The above information permits node i to know which updates of an update message (each update message contains a list of updates) have to be retransmitted and which neighbors should be requested to acknowledge such retransmission. Node i retransmits the list of updates in an update message when the retransmission counter of the corresponding entry in the MRL reaches zero. The retransmission counter of a new entry in the MRL is set equal to a small number (e.g., 3 or 4).

2.3 – Information Exchanged among Nodes

In WRP, nodes exchange routing-table update messages (which we call "update messages" for brevity) that propagate only from a node to its neighbors. An update message contains the following information:

- The identifier of the sending node.
- A sequence number assigned by the sending node.
- An *update list* of zero or more updates or ACKs to update messages. An update entry specifies a destination, a distance to the destination, and a predecessor to the destination. An ACK entry specifies the source and sequence number of the update message being acknowledged.
- A *response list* of zero or more nodes that should send an ACK to the update message.

In the event that the message space is not large enough to contain all the updates and ACKs that a node wants to report, they are sent in multiple update messages. An example of this event can be the case in which a node identifies a new neighbor and sends its entire routing table.

The response list of the update message is used to avoid the situation in which a neighbor is asked to send multiple ACKs to the same update message, simply because some other neighbor of the node sending the update did not acknowledge.

The first transmission of an update message must ask all neighbors to send an ACK, of course, and this is accomplished by specifying the "all-neighbors address," which consists of all 1's.

When the update message reports no updates, the "empty address" is specified; this address consists of all 0's and instructs the receiving nodes not to send an ACK in return. This type of update message is used as a "hello message" from a node to allow its neighbors to know that they maintain connectivity, even if no user messages or routing-table updates are exchanged.

As we explain subsequently, an ACK entry refers to an entire update message, not an update entry in an update message, in order to conserve bandwidth.

2.4 – Routing-Table Updating

Figures 1 and 2 specify important procedures of WRP used to update the routing and distance tables.

A node can decide to update its routing table after either receiving an update message from a neighbor, or detecting a change in the status of a link to a neighbor. When a node i receives an update message from its neighbor k, it processes each update and ACK entry of the update message in order.

In WRP, a node checks the consistency of predecessor information reported by *all* its neighbors each time it processes an event involving a neighbor k. In contrast, all previous path-finding algorithms [5, 10, 15] check the consistency of the predecessor only for the neighbor associated with the input event. This unique feature of WRP accounts for its fast convergence after a single resource failure or recovery as it eliminates more temporary looping situations than previous path-finding algorithms.

Processing an Update: To process an update from neighbor k regarding destination j, the distance and the predecessor entries in the distance table are updated. A flag (tag) is set to specify that this entry in the table has been changed. A unique feature of WRP is that node i also determines if the path to destination j through any of its other neighbors $\{b \in N_i | b \neq k\}$ includes node k. If the path implied by the predecessor information reported by node b includes node k, then the distance entry of that path is also updated as $D^i_{jb} = D^i_{kb} + D^k_j$ and the predecessor is updated as $p^i_{jb} = p^k_j$. Thus, a node can determine whether or not an update received from k affects its other distance and routing table entries.

To update its distance and predecessor for destination j (procedure RT_Update), node i chooses a neighbor p that has reported routing information such that:

- The path from p to j (which is implied by the predecessor information reported by p) does not include node i
- $D^i_{jp} \leq D^i_{jx}$ for any other neighbor x, and $D^i_{yp} \leq D^i_{yx}$ for any other neighbor x and for every node y in the path from i to j.

The above means that node i chooses node p as its successor to a destination j if that neighbor appears to offer a smallest-cost loop-free path to j and all the intermediate nodes in the path to j.

When node i sends an update message, it updates its message retransmission list. For each destination j for whom there is an update being reported, node i sets the ack-required flag for all its neighbors. It also adds an entry in the message-retransmission list containing the sequence number given to the update message, and starts the retransmission timer for that entry.

Sending New and Retransmitted Update Messages: Node i sends a new update message after processing updates from its neighbors or detecting a change in a link to a neighbor. Whenever node i sends a new update message, it must

- Decrement the retransmission counter of all the existing entries in the MRL
- Delete the updates in existing entries in the MRL that are included in the new update message
- Add an entry in the MRL for the new update message

When the list of updates of a MRL entry is emptied by the transmission of a new update message, node i erases that entry from the MRL.

When the retransmission counter for a retransmission entry m in the MRL expires, node i sends an update message with a new sequence number, an update list containing the list of updates of the retransmission entry, and a response list specifying those neighbors who did not acknowledge the update message earlier (i.e., every neighbor k for whom $a^i_{km} = 1$). The retransmission counter of existing entries in the MRL is not modified.

Note that, based on the above retransmission strategy, there is no limit on the number of times node i would retransmit an update message to an existing neighbor. However, as we discuss below, node i stops considering node k as its neighbor after it fails to communicate with it for some finite amount of time.

Processing an ACK: An ACK entry in an update message refers to another entire update message, i.e., it acknowledges all the updates included in the update message bearing the referenced sequence number. Therefore, it is up to the node whose update message is being acknowledged to ascertain which updates are implied by a received ACK.

To process an ACK from neighbor k, node i scans its MRL for the sequence number matching the sequence number specified in the ACK received. Whenever a match is found, node i resets the ack-required flag for neighbor k; if $a^i_{pm} = 0$ for entry m and every neighbor p of node i, the retransmission entry is deleted. This scheme obtains short ACKs at the expense of additional processing.

Node i may receive an ACK for an update message whose retransmission entry has been erased after sending a more recent update message for the same destinations. In that case, node i simply ignores the ACK.

Handling Topology and Link-Cost Changes: To ensure that nodes know that they have connectivity even when they do not transmit user messages or routing-table updates for some time, every node i must periodically send an update message reporting no changes (hello messages). Acknowledgments are not required for such update messages, and they can be very short (e.g., a byte for control information and a byte for the node identifier, since the control information can imply that there is no sequence number, update list, or response list in the message). Alternatively, a node may retransmit an update message if it is not too long. When a node k comes up, it transmits a hello message.

Given that short periodic update messages are transmitted by every node, the failure of a link to a neighbor is detected by the lack of any user or update messages being received from that neighbor over a period of time equal to a few update-message transmission periods. Similarly, new links and nodes are detected by means of update messages or user messages.

When node i receives an update or user message from node k and node k is not listed in its routing table or distance table, it adds the corresponding entry to its distance or routing table for destination k. An infinite distance to all destinations through node k is assumed, with the exception of node k itself and those destinations reported in node k's updates, if the message received from k was an update message. In addition, node i notifies node k of the information in its routing table. This information can be transmitted in one or multiple update messages that only node k needs to acknowledge.

When a link fails or a link-cost changes, node i recomputes the distances and predecessors to all affected destinations, and sends to all its neighbors an update message for all destinations whose distance or predecessor have changed.

2.5 – Example

The following example illustrates the working of WRP. Consider a four node network shown in Figure 3(a). All links and nodes are assumed to have the same propagation delays. Link-costs are as indicated in the figure. Node i is the source node, j is the destination node and nodes k and b are the neighbors of node i. The arrows next to links indicate the direction of updates messages and the label in

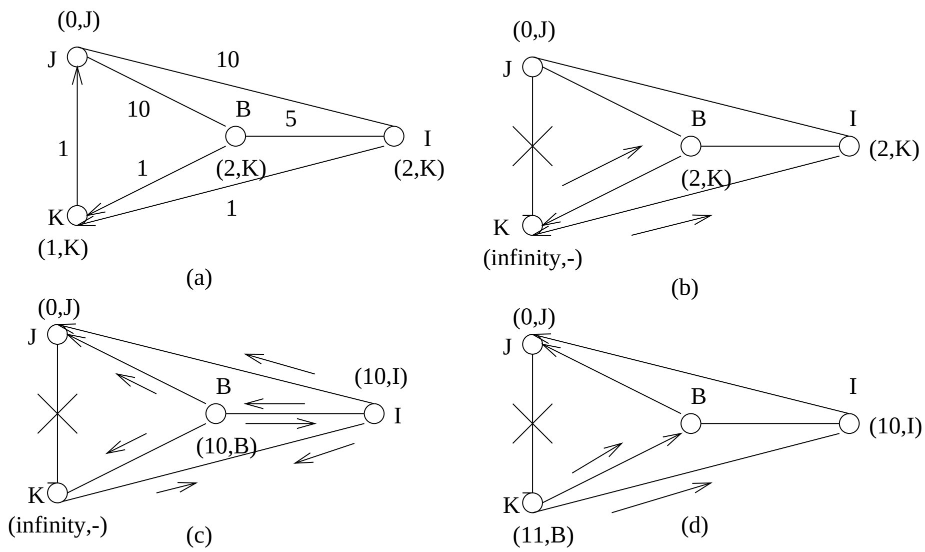


Figure 3: Example of the algorithm's operation

parentheses gives the distance and the predecessor to destination j. Each update will be acknowledged by an ACK message from the neighbor. ACKs are not shown in the figure. The figure focuses on update messages to destination j only.

When link (j, k) fails, nodes j and k send update messages to their neighboring nodes as shown in Figure 3(b). In this example, node k is forced to report an infinite distance to j as nodes b and i have reported node k as part of their path to destination j. Node b processes node k's update and selects link (b, j) to destination j. This is because of step(2) of WRP which forces node b to purge any path to node j involving node k. Also, when i gets node k's update message, i updates its distance table entry through neighbor k and checks for the possible paths to destination j through any other neighboring nodes. Thus, a node examines the available paths through its other neighboring nodes and updates the distance and the routing table entries accordingly. This results in the selection of the link (i, j) to the destination j (Figure 3(c)). When node i receives neighbor b's update reporting an infinite distance, node i does not have to update its routing table as it already has correct path information (Figure 3(d)). Similarly, updates sent by node k reporting a distance of 11 to destination j will not affect the path information of nodes i and b. This illustrates how the method used in WRP to update a node's distance table (Step (2) in Procedure DT) helps in the reduction of the formation of temporary loops in the explicit paths.

3 – Correctness of WRP

In this section, we show that the basic routing algorithm used in WRP is correct. The following assumptions are made on the behavior of links and routers for the working of WRP.

1. Messages are transmitted reliably. A lower-level protocol is responsible for maintaining the status of the link.
2. Messages are sent by a router over a link only when the link is perceived as being up.
3. A router that is not functional cannot receive or send any messages.
4. All routers are initially down.
5. Update messages received by a router are processed in the order of their arrival (FIFO).
6. Link lengths are always positive and a failed link has an infinite length.
7. Time T is defined such that between the time interval 0 and T links and routers go up and down and the cost of the link changes; at time T, links have the same status at both ends and there are no changes after time T.

For simplicity, the following proof assumes that all update messages sent over an operational link are received correctly. In practice, WRP handles errors by means of retransmissions. In terms of the correctness proof, the effect of retransmissions is that of added delay in the delivery of an update message to a neighbor, and a link fails when a given number of retransmissions have been attempted. In essence, this proof shows that the path-finding algorithm (PFA) on which WRP is based is correct.

Definition 1
The link weight d_{ij} for link (i, j) is extracted from the distance table D_v at a router v if there is a column k in D_v such that $d_{ij} = D^v_{jk} - D^v_{ik}$ and $p^v_{jk} = i$. Similarly, the link weight for link (i, j) can be extracted from the routing table if d_{ij} agrees $D^v_j - D^v_i$, where $p^v_j = i$.

Lemma 1
If a routing table is generated by PFA based on the distance table, any link weight that can be extracted from this routing table can be extracted from a column in the distance table.

Proof
Let N_v denote the set of neighbors of router v. Let d_{ij} be the link weight extracted from the routing table of node v. By definition 1, the cost of the link can be extracted from the routing table as $(D_j^v - D_i^v)$, the predecessor $p_j^v = i$ and the successor $s_i^v = k$, from the function *RT_Update*. The procedure *RT_Update* requires each distance in the routing table to be the minimum among the rows corresponding to the same destination in the distance table entry of the router as defined by the distributed Bellman-Ford algorithm. Therefore, Lemma 1 is true. $\square$

Lemma 2
When a node comes up and initializes its distance table, the link weight that can be extracted from any of its distance table entries is the weight of the link.

Proof
The recovery of a router can be viewed as all the links connected to that router coming up. Initially, when the router is down, its distance table entries will have infinite distance. A link coming up will be recorded as a single entry in the distance vector, which is nothing but the weight of the link. Therefore, the link weight extracted from any column in the distance table is the weight of that link. $\square$

Lemma 3
The link cost change of a link will be reflected in the distance and the routing tables of a neighboring router after a finite time T.

Proof
The change in the link cost can be due to the link coming up, the link going down, or the cost of the link changing.

When a link comes up, a new column entry will be added to the distance table and the new link cost will be assigned to the corresponding entry in the distance table. Procedure RT_Update will be called, which eventually updates the routing table entry.

When a link goes down, the column entry will be deleted and the distance entries in the distance table will be set to ∞. The procedure RT_Update again updates the routing table entries accordingly.

When the link cost changes, the distance entry in the distance table is updated to reflect the new link cost (Step (1) and Step (2)). These changes will be updated in the routing table again by the procedure RT_Update.

From assumption (8) we have, any change that occurs in the time interval $(0, T)$ will be updated by time T. This implies that the link cost changes will be reflected in the distance and routing tables of nodes adjacent to the links within a finite time T. □

Property 1
After a finite time interval T, the routing table structures at all routers will form the final shortest path.

Proof
The proof consists of the following two parts:

1. The old topology information present in the router's routing and distance tables is updated.

2. The shortest-path trees are eventually computed.

Let the initial time be $T(0) = T$. Let $T(K)$ be the time by which all messages that are in transit at time $T(K-1), K \geq 0$, have arrived at their destination and have been processed. The proof is done by induction on K. At time $K = 0$, the property holds true. Assume that the property is true for $0 \leq K \leq M$.

A path of $M+1$ links can be a concatenation of an adjacent link and a path with M links, or a concatenation of next 2 and $M-1$ links, or next 3 and $M-2$ links and so on. This we can generalize as the concatenation of l and $M-l+1$ links. From the assumptions, by time $T(M+1)$, the routing trees at time $T(M)$ of the routers have all been communicated to their neighboring routers. This is true for any $M = M - l$. By Lemma 3, these link cost changes will be updated in the router's distance and routing tables within a finite time T. This proves first part of the property.

The change in the link cost will result in a routing table update (in procedure RT_Update) as required. When a router has to select a new path, the minimum in row entry for that destination router will be chosen from the distance table entries resulting in the shortest path in the final graph all along the way. This implies that the routing table structures at all routers form the final shortest path. □

Theorem 1
If the distance entries in the distance and routing tables are finite, then a path can be extracted from the distance and routing table entries and this extracted path is loop-free.

Proof
Let $T(K = 0)$ be the initial time when the algorithm begins execution. The theorem is true for $K = 0$ since no link exists between routers at time $t = 0$.

Assume that the property is true for $T(M), 0 \leq M \leq K-1$. By time $T(K)$, all the routing changes at time $T(K-1)$ would have been communicated to all routers (assumption). No router will be marked as undetermined as all the distance entries are finite.

When a router recovers, within a time $T(K-1)$, the information about the change in the link cost will be communicated within a finite time (by Lemma 3). As all the entries in the table are finite, a path can be extracted from any router i to any other router j by traversing through the distance and the routing tables.

When a particular link is selected as a path from i to j, the loop freeness of the path is checked in step 2 and RT_Update. An update message about the link cost change will be sent to the neighbor. The loop-freeness of the update messages can be verified by traversing from destination router to the source router using predecessor information present in each entry of the distance and routing tables.

Therefore, the paths in the final graph are loop-free. □

The following theorems prove that PFA terminates in such a way that the distance to any other router maintained in the routing table in each router is the shortest distance of the final graph and the distance to any unreachable router is marked as undetermined.

Property 2

If router j is not connected to router i in the final topology, then the distance between the two routers is equal to infinity for all time after $T(H(i,\infty)+1)$.

Proof

If a router i does not have a path to router j, the distance entries in the router i's tables will be set to ∞ (from the algorithm description). Let $H(i,d)$ be the maximum number of links in the path from i whose distance to any other router is less than or equal to d in the final topology. This implies $H(i,d)$ is a finite quantity.

By Property 1, all the paths with links less than or equal to $H(i,\infty)$ will have their final length by time $T(H(i,\infty)+1)$. This proves Property 2. □

Theorem 2

PFA terminates within a finite time after the last topological change happened.

Proof

Assume that the algorithm does not terminate. This implies that there must be an infinite number of messages sent after the last topological change. These infinite messages must have finite distances since from Property 2 if the distance between the two routers is equal to infinity, the algorithm converges. Moreover, from Theorem 1, the path extracted from the distance table must be a simple path. Thus, there must be some neighbor b that sends finite distances an infinite number of times to node i for node i to send messages without stopping.

Each time router i sends a message, it can be due to any one of the following reasons

1. It receives D^b_j from b and $D^i_j = D^b_j + d_{ib}$ where d_{ib} is the link weight
2. D^i_j has been in router i's distance table when it receives a message from b
3. Neighbor b is in the path from i to j through another neighbor $k(\neq b)$ and $D^i_{jk} = D^i_{bk} + D^b_j$ (Step (2))

If the first case happens infinite times, router b sends D^b_j infinite times and $D^b_j = D^i_j - d_{ib} < D^i_j$ because $d_{ib} > 0$.

The second case can happen in a situation where D^i_j is not stable. This means that D^i_j is changed forever, which is similar to the first case in that there must be a neighbor b' such that b' sends $D^{b'}_j$ infinite times and $D^{b'}_j = D^i_j - d_{ib'} < D^i_j$, because $d_{ib'} > 0$. Else, if D^i_j becomes stable, then there must be an infinite number of times in which router i receives a distance that is shorter than D^i_j.

For the third case to happen an infinite number of times, D^i_{jk} must be changed forever. This in turn means that a neighbor has to send the distance vector D^b_j infinite times. This reduces to case (2) and eventually to case (1).

Consequently, there must be a neighbor b' sending $D^{b''}_j$ infinite times and $D^{b''}_j = D^i_{jb''} - -d_{ib''} < D^i_j - d_{ib''} < D^i_j$ because $d_{ib''} > 0$.

Therefore, in all of the cases, there must be a router that will infinitely generate messages with a distance at least w less than D^i_j, where w is the minimum distance of the final graph. This will consequently contradict that all the distances are positive by recursively applying the above argument. □

Property 3

When PFA terminates, the link weights maintained in the distance table must be in the final graph.

Proof

This proof is by induction. When a router comes up, its distance entries in the distance and routing tables are maintained correctly by Lemmas 1 and 2 and Property 1. If a link is not in the final graph, it implies that router must have detected a link failure that caused it to delete the corresponding column entry from the distance table entry of the router and the distance is marked as infinity. If the distance in the final graph d_{ij} is different from the earlier distance, the router

i must have been notified about this link-cost change by its neighbor. Thus, the correct distance entries are maintained in the final graph for all adjacent routers.

Assume that the result is true for nodes that are k hops away from i.

We will show by induction that the result is true for routers that are $k+1$ hops away from i. Let j be a router that is $k+1$ hops away from router i and a be a router that is k hops away from i. Since all routers that are k hops away from i maintain the distance entries correctly, the distance entry is correct for router a. The distance from j to i is the sum of d_{ja} and D_i^a (step 1 of the algorithm). This is nothing but the minimum of the distances from i to j and hence is the shortest-path from i to j. Therefore, this distance entry will be present in the final graph unless the link has gone down before the algorithm terminates in which case, an infinite distance will be maintained. This proves the property. □

Theorem 3

When PFA terminates, the distance for any router i to any other router j in the routing table of router i is the shortest distance from i to j in the final graph and the successor will be maintained correctly; furthermore, the distance from router i to any unreachable router is marked as undetermined.

Proof

We prove the theorem by induction.

¿From Lemma 3, the weight of any link must be maintained by its adjacent node. When a link comes up, the cost of the link will be assigned to the distance table entry (neighbor router) and the predecessor will be initialized to be the source router itself. A check is made to see whether the distance table entry is smaller than the routing table entry and the routing table will be updated according to the procedure RT_Update with the successor and the predecessor entries properly set. If the link is in the path to the destination through any other neighboring routers, then the distance and the routing table entries will also be updated.

Assume that the result is true for a node j that is k hops away from router i.

We will show by induction that the result is true for a router $k+1$ hops away from i. Consider a router j that is $k+1$ hops away from i. There must be a neighbor b of router j that is k hops away from i and that maintains correct distance and routing table entries. Let d_{jb} denote the distance between router j and its neighbor router b which are $k+1$ and k hops away from i respectively. Let D_b^i be the distance from i to b. D_b^i is the shortest path from i to b as D_b^i is the minimum in row of distance table for router b and each distance table entry represents an existent path,

Since b is a neighbor of j, $D_j^i = D_b^i + d_{bj}$ is the shortest path from i to j, with d_{bj} being the minimum in row entry. The predecessor path will also be maintained

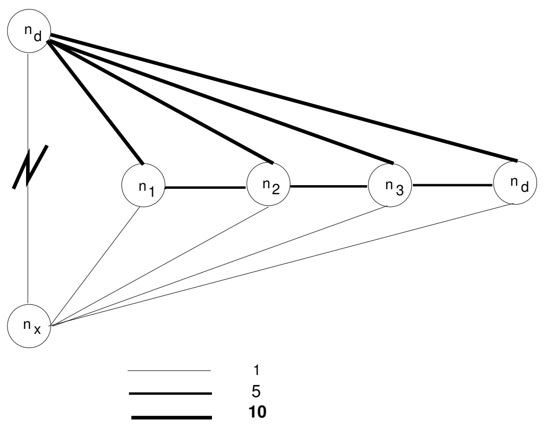


Figure 4: Complexity

correctly (from step 1). Furthermore, any router x in the shortest path from i to j must also have the subpath from x to i as the shortest path because it is the minimum in the row of x.

The RT_Update procedure is called in the update routine after updating all the distance table entries of that router. This routine picks up a minimum entry through one of its neighbors and will have a successful trace for the destination router j and thus will have $D^i_j = D^i_{ji'} = D^{i'}_j + d_{ii'} = D^i_j$ and $s^i_j = b$. □

4 – Complexity Analysis

WRP's time complexity is $O(h)$ in the worst-case, where h is the height of the routing tree. Theorem 4 below proves this result. *Time complexity* is defined as the largest time that can elapse between the moment T when the last topology change occurs and the moment at which all the routers have final shortest path and distances to all other routers. *Communication complexity* is defined as the maximum number of node identities exchanged (messages) after time T before the final graph is reached.

Consider Figure 4. The weight of the links are as indicated. Assume n_d is the destination router. Routers n_1, n_2, n_3 and n_4 will have the shortest path router n_x before link (n_d,n_x) fails. After the link failure, routers n_1, n_2, n_3 and n_4 immediately identify that the only possible way to reach the destination router n_d is through the link (n_i,n_d) for i = 1,2,3,4 upon receiving an update message from router n_x about the link failure, instead of going through an intermediate step of selecting the path through routers n_2, n_3 and n_4 respectively as in the case of any other path-finding algorithm. That is, the routers need not have to wait for an update message from the neighbor n_2, n_3 and n_4 before arriving at the final graph. This reduces the number of update messages required.

Theorem 4
The time complexity for a single failure/change for WRP is $O(h)$ in the worst-case, where h is the maximum height of the routing tree experienced during the computation.

Proof
Consider a source router i and a destination router j. Let the changed link be (n, m) and node m is a router downstream to router n. There are four possible situations involving the shortest path from i to j.

1. (n, m) is not on the shortest path and its length does not change enough to change the shortest path.
2. (n, m) is not on the shortest path and its length decreases enough that it becomes part of the shortest path.
3. (n, m) is on the shortest path and its length does not change enough to modify the shortest path (although the length of the shortest path changes).
4. (n, m) is on the shortest path and its length increases enough that the shortest path changes.

A router with the initial shortest path not going through the changed link (Case (1)) does not change its routing table since the original shortest path is not changed and the change in the link cost has resulted only in the increase in the path length through other routes.

In Case (2), router is aware of the change in the link cost along the shortest path after a delay not exceeding the number of links on the new shortest path. In Case (3) the change will be noticed in the worst case after a delay of at most the number of links in the shortest path.

For Case (4), let router n_k with the original shortest path through the changed link be k hops away from router n on the initial shortest path. When a link distance changes or a link fails, the node containing the failed link selects a new neighbor (changes the successor) for a path to a destination j. This changes the routing table entry at router n and the routing vector generated due to link failure will be sent to all its neighbors. Each of these neighbors will update their table entries and the change in the link cost propagates. This process continues until a stable router which does not change its successor is encountered. The tables are updated either on the receipt of an update message or if the distance update received from a router's neighbor has any effect on the router's other distance table entries. The distance of the stable node found in the path from i to j in the new shortest path is bounded by h, the height of the tree. Therefore, in the worst case, the number of steps required for a router to converge to its correct distance is $O(h)$. □

5 – Simulation Results

To gain insight into the average-case performance of WRP in a dynamic environment, we have simulated its operation using an actor-based, discrete-event simulation language called *Drama* [17], together with a network simulation library. The library provides a standard input syntax and a framework for constructing simulations consisting of routers attached to each other via links. Drama itself is an extension to C.

The network simulation library treats both routers and links as *actors*. Routers send packets over links by using the function-call interface to the link's actors, but they receive packets by responding to messages delivered from the input queue. Link failures and recoveries are simulated by sending link status message to the nodes at the end points of the appropriate links. In the link models used in the simulation, each link responds to an update packet by encapsulating the packet as a message and sending the message to the link itself. Node failures can be treated as all links connecting to that node going down at the same time and the link cost changes can be treated as a link failing and recovering with a new link cost.

The connectivity of a mobile node is said to be lost when a node does not hear from a mobile node for a certain period of time. The connectivity with a node will be reestablished when a node hears from a mobile node again. Mobility is modeled as an arbitrary set of failures and recoveries of a mobile node at random points in time. All simulations are done assuming unit propagation time and zero packet processing time at each node. If a mobile node fails when the packets are in transit, the packets are assumed to get dropped.

Our goal is to compare the performance of WRP against the performance of routing protocols based on DBF, DUAL, and ILS. To reduce the complexity of the simulation, we have eliminated those features of the protocols that were common to all; these features concern the reliable transmission of updates over unreliable links, and the identification of neighbors. Accordingly, our simulation assumed that, for any of the protocols simulated, any update message sent over an operational link is received correctly, and that a node always receives enough user messages to know that it continues to have connectivity with a neighbor. According to these assumptions, there is no need to account for acknowledgments, retransmissions of updates, or periodic transmissions of update messages.

However, our intent in running the simulations was to obtain insight on the

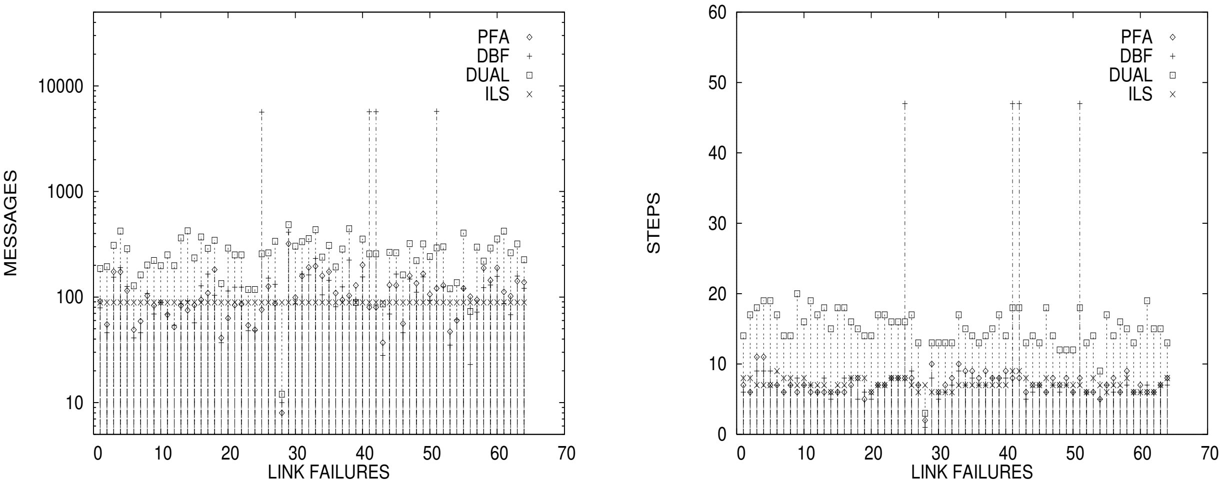


Figure 5: ARPANET Link Failure

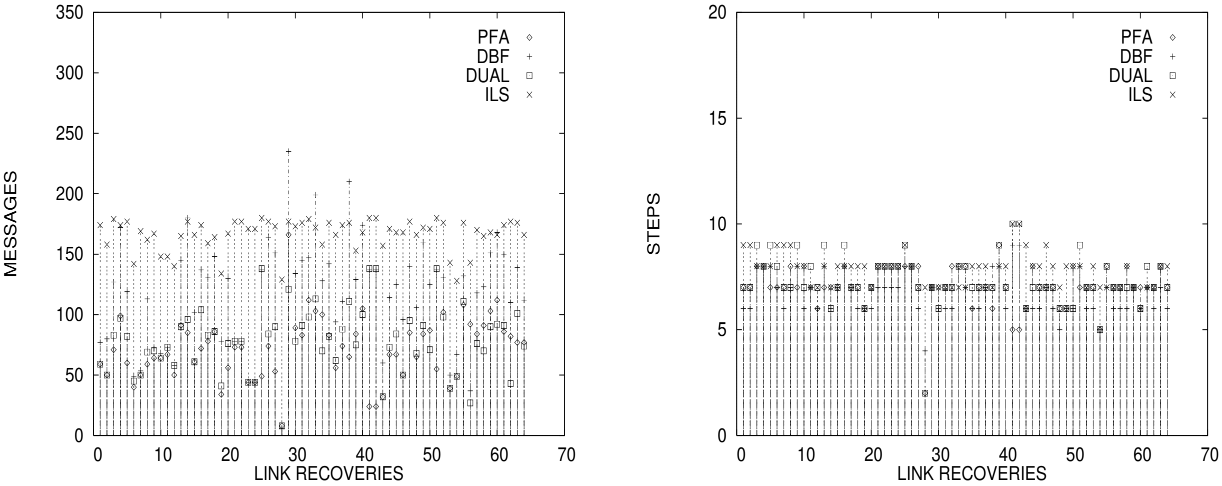
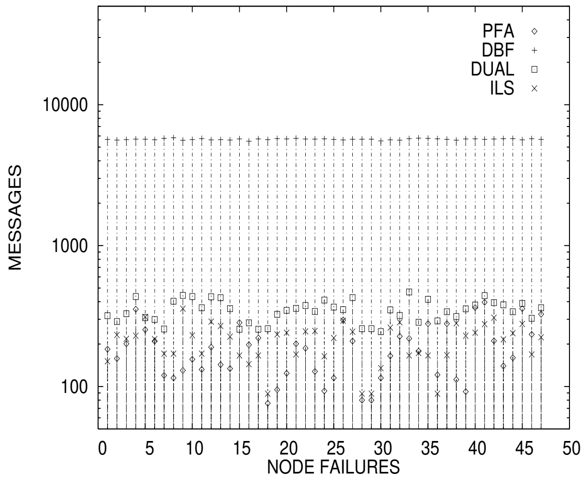


Figure 6: ARPANET Link Recovery



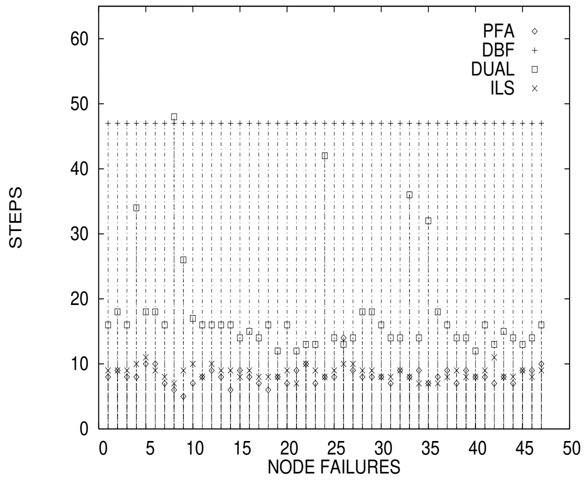


Figure 7: ARPANET Node Failure

comparative overhead of different protocols that necessarily require the transmission of acknowledgments to update messages. We approached this problem in the following manner: In a wireless packet radio network, the same update messages sent by a node is received by all its neighbors, i.e., each update message is broadcast to a node's neighbors. However, to guarantee the reliable transmission of updates, each neighbor must send an acknowledgment to the sender of the update. Therefore, under the assumption that no errors or collisions occur in the network channel, counting the number of acknowledgments received for a single update broadcast to all neighbors is much the same as counting the number of updates sent by a node to its neighbors on a point-to-point basis and with no acknowledgments—the two counts differ only by one. Accordingly, we simulated the routing protocols' operation in a wireless network using the same point-to-point links typical of wireline networks. The message count obtained from the simulation runs is not the exact number of updates and acknowledgments sent by each protocol, but accurately reflects the relative differences among protocols.

The resulting simplified version of WRP we simulated is simply the path finding algorithm (PFA), and is the same basic algorithm first described in [13]. Similarly, ILS, DBF, and DUAL correspond to the ideal case of the best protocols that could be designed based on these algorithms.

To simulate the routing algorithm, a node receives a packet and responds to it by running the routing algorithm, queueing the outgoing packets and processing the updates one at a time in the order in which they arrive. Drama's internals ensure that all the packets at a given time are processed before new updates are generated.

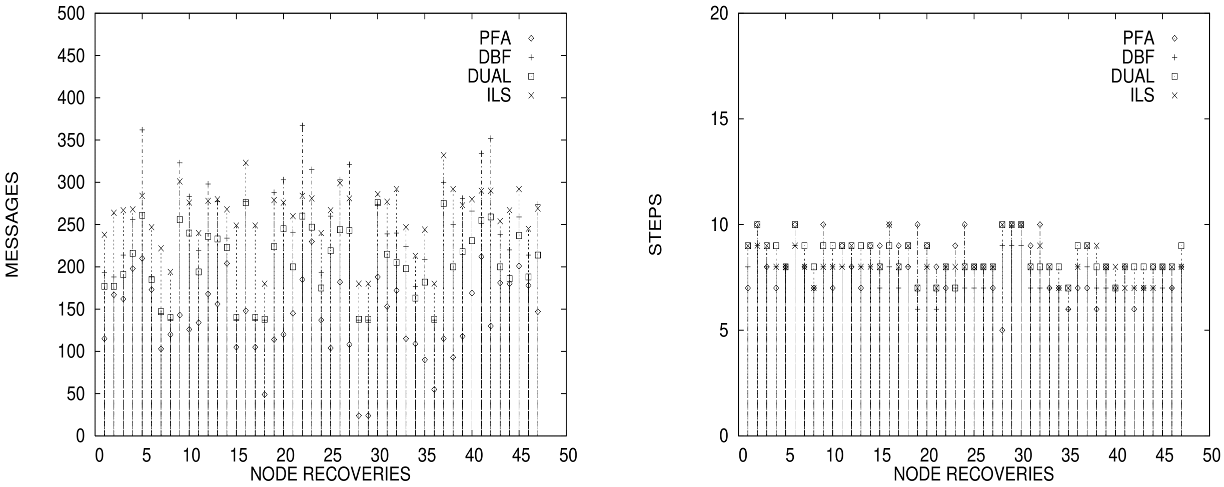


Figure 8: ARPANET Node Recovery

The simulations were run on several network topologies such as *Los-Nettos*, *Nsfnet* and *Arpanet*. We chose these topologies to compare the performance of routing algorithms for well-known cases given that we cannot sample a large enough number of networks to make statistically justifiable statements about how an algorithm scales with network parameters. The los-nettos topology has 11 nodes, a diameter of 4 hops, and each node has at most four neighbors. The Nsfnet topology has 13 nodes, a diameter of 4 hops, and each node has at most 4 neighbors. The ARPANET topology has 57 nodes, a diameter of 8 hops, and each node has a maximum of four neighbors.

For the routing algorithms under consideration, there is only one shortest path between a source and a destination pair and we do not consider null paths from a node to itself. Data are collected for a large number of topology changes to determine statistical distribution. The statistics has been collected after each failure and recovery of a link. To obtain the average figures, we make each link (or node) in the network fail and count the number of steps and messages required for each algorithm to converge. Then the same link (node) is made to recover and the process is repeated. The average is taken over all failures and recoveries. Again, this message count is not exact, but the relative difference from one protocol to another is accurate.

5.1 – Total Response to a Single Resource Change

The graphs in Figures 5 and 6 depict the number of messages exchanged and the number of steps required before PFA, DBF, DUAL, and ILS converge for every link failing and recovering in the ARPANET topology. We focus more on the results for the ARPANET topology, because of its larger size. Similar graphs for every node failing and recovering are given in Figures 7 and 8 respectively. All topology changes are performed one at a time and the algorithms were allowed to converge after each such change before the next resource change occurs. The ordinates of the graphs represent the identifiers of the links and the nodes while the data points show the number of messages exchanged after each resource change (graphs on the left hand side) and the number of steps needed for convergence (graphs on the right hand side) in each of these figures.

For a single resource failure, PFA outperforms DUAL. This is because, PFA does not use an internodal coordination mechanism that spans several hops to achieve loop freedom. The performance of PFA is comparable to that of ILS after resource failures. The performance of PFA and DUAL is much better than that of ILS after resource recoveries. The counting-to-infinity problem of DBF can be clearly seen in both resource failures and resource recoveries. Given that both resource recoveries and failures will occur in the WRP, PFA offers the best total response to single topology changes, in terms of both update messages and time required to obtain correct routing tables after a topology change.

5.2 – Dynamics with Mobile Nodes

We modeled mobility in the simulation by making the links fail and come back up arbitrarily at random points in time. The network is assumed to be fully connected with potential links. At startup, the topology is initialized to some well known topology, such as *los-nettos*, *Nsfnet* or *ARPANET*. After initialization, to simulate the movement of a node, a node is assumed to have failed at its previous location and reappear in its new location. Node failure is simulated as all the links associated with that node going down at the same time. The gradual movement of a node from one location to another is simulated by means of link failures and additions. When a link fails, it can be assumed that a node is no longer in the neighborhood of its previous neighbor. The addition of a new link is viewed as a movement of a node wherein, a node reappears in the new neighborhood.

The links are chosen at random from the set of all the existing links in the fully connected network. Selecting any particular link is equally likely. The probability of a link failing or recovering is also equally likely. We also have imposed an additional condition in our simulations that a node at any given time cannot have more than x neighbors. Here, x indicates the degree of the node. This condition is imposed in order to make sure that all the links pertaining to one node alone will not be active. This helps in simulating the mobility more closely. This, of course, is only an approximation of the more gradual topology changes that would be experienced in a real mobile network.

The average number of messages and the average message length for each of

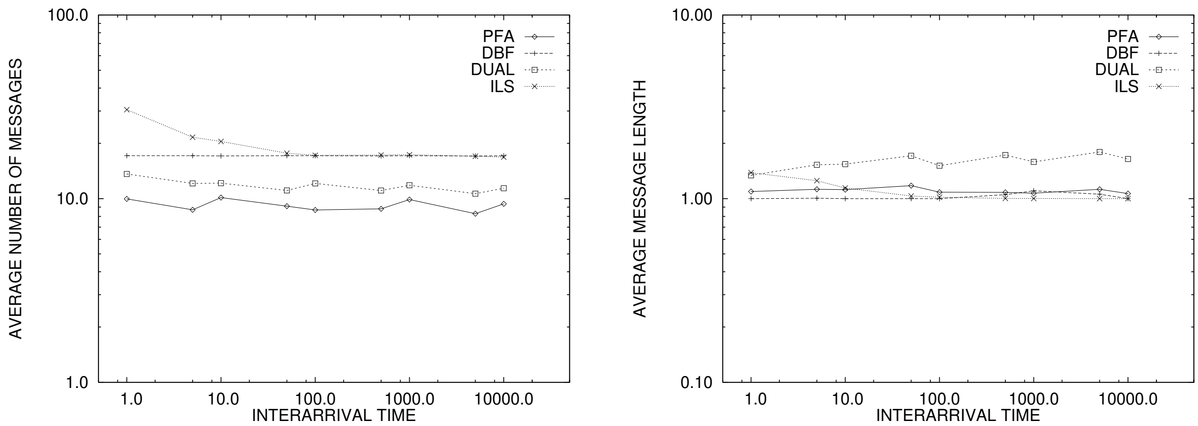


Figure 9: Los-Nettos

these algorithms are obtained by varying the interarrival time between two events (Figures 9–11). An event can be either a link failure or a link recovery. For the purpose of event generation, we consider a fully connected topology and start off with a given initial topology. Since any random link can fail or recover at any time, our model simulates mobility closely.

The above results indicate that the routing algorithm of WRP outperforms all other algorithms which we have simulated, namely, DBF, DUAL and ILS. We were not able to simulate ILS for the ARPANET topology due to limited resources. The statistics about the average number of messages and the average message length have been collected for all the above mentioned topologies for all the four algorithms by varying the interarrival time between events (failures and recoveries).

In all cases, the average number of messages for DBF and DUAL are more than that of WRP. This is because, DBF suffers from counting-to-infinity problem and DUAL uses an interneighbor coordination mechanism to achieve loop-freedom and this synchronization mechanism spans the entire diameter of the network. ILS sends maximum number of messages since the complete topology information has to be exchanged between neighbors every time the topology changes.

The average length of each message is the highest in DUAL as compared to all other algorithms. The average message length in case of ILS is almost constant since it always sends the complete topology information. Even though we do not have simulation results for ILS in case of ARPANET topology, we can extrapolate the results from the other two network topologies and can expect similar behavior for ARPANET topology also.

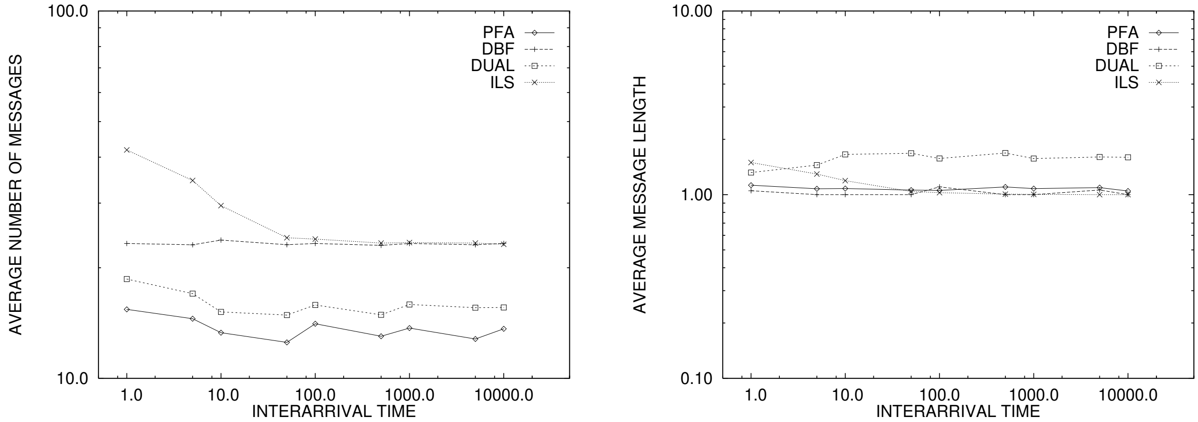


Figure 10: Nsfnet

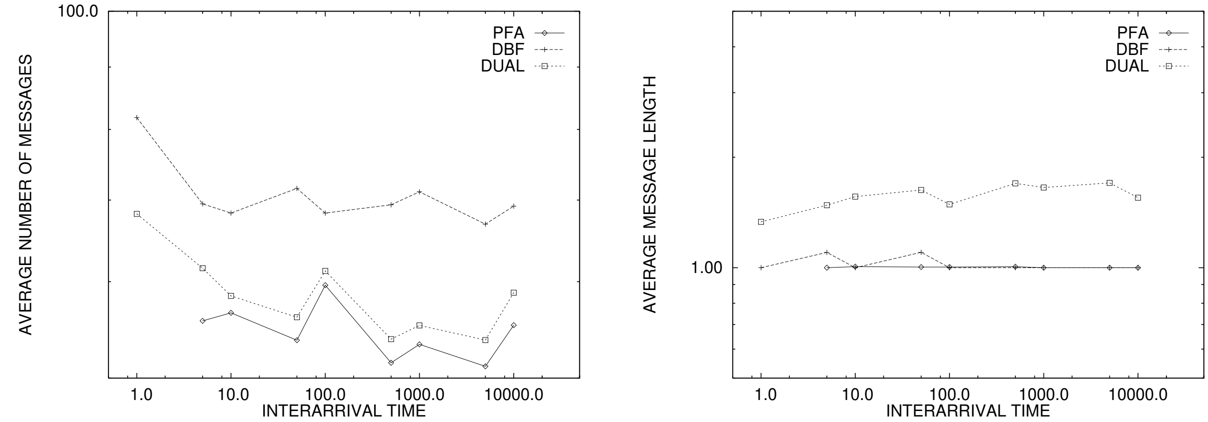


Figure 11: ARPANET

6 – Conclusion

A new routing protocol, WRP, for a wireless network has been presented. This protocol is based on a path-finding algorithm which substantially reduces the number of cases in which routing loops can occur. A mechanism has been proposed for the reliable exchange of update messages as part of WRP. The basic algorithm used in WRP has been proved to be correct and WRP's complexity has been analyzed. The performance of the routing algorithm in WRP has been compared with that of an ideal topology broadcast algorithm (ILS), DUAL and DBF for highly dynamic environment through simulations. The simulation results show that WRP provides about 50% improvement in the convergence time as compared to DUAL. The results indicate that WRP is an excellent alternative for routing in wireless networks.

References

[1] R. Albringhtson, J.J. Garcia-Luna-Aceves and J. Boyle, "EIGRP–A Fast Routing Protocol Based on Distance Vectors", *Proceedings Networld/Interop 94*, Las Vegas, Nevada, May 1994.

[2] D. Beyer *et. al.*, "Packet Radio Network Research, Development and Application", *Proc. SHAPE Conference on Packet Radio*, Amsterdam, 1989.

[3] D. Beyer, "Accomplishments of the DARPA SURAN Program", *Proc. IEEE MILCOM 90*, Monterey, California, Dec. 1990, pp.855–862.

[4] D. Bertsekas and R. Gallager, *Data Networks*, Second Ed. Prentice Hall, Inc. 1992.

[5] C. Cheng, R. Reley, S. P. R Kumar and J. J. Garcia-Luna-Aceves, "A Loop-Free Extended Bellman-Ford Routing Protocol without Bouncing Effect", *ACM Computer Communications Review*, Vol.19, NO.4, 1989, pp.224–236.

[6] M. Scott Corson and Anthony Ephremides, "A Distributed Routing Algorithm for Mobile Wireless Networks", *ACM J. of Wireless Networks*, Jan. 1995, pp. 61–81.

[7] J.J. Garcia-Luna-Aceves, "A Fail-Safe Routing Algorithm for Multihop Packet-Radio Networks", *IEEE INFOCOM* April, 1986.

[8] J.J. Garcia-Luna-Aceves, "Loop-Free Routing Using Diffusing Computations", *IEEE/ACM Trans. Networking*, Vol.1, No. 1, Feb. 1993, pp.130–141.

[9] J. Hagouel, "Issues in Routing for Large and Dynamic Networks", *IBM Research Report*, RC 9942 (No. 44055), April 1983.

[10] P.A. Humblet, "Another Adaptive Shortest-Path Algorithm", *IEEE Trans. Comm.*, Vol.39, No.6, June 1991, pp.995–1003.

[11] B.M. Leiner, D.L. Nielson and F.A. Tobagi, *Proc. IEEE*, Packet Radio Networks Special Issue, Jan. 1987.

[12] J. Moy, "OSPF Version 2", *RFC 1583*, March 1994.

[13] Shree Murthy and J.J. Garcia-Luna-Aceves, "A More Efficient Path-Finding Algorithm", 28^{th} *Asilomar Conference*, Pacific Groove, CA, Nov. 1994, pp. 229-233.

[14] Charles E. Perkins and Pravin Bhagwat, "Highly Dynamic Destination-Sequenced Distance-Vector Routing (DSDV) for Mobile Computers'", *ACM SIGCOMM*, Oct. 1994, pp.234–244.

[15] B. Rajagopalan and M. Faiman, "A Responsive Distributed Shortest-Path Routing Algorithm within Autonomous Systems," *Journal of Internetworking: Research and Experience*, Vol. 2, No. 1, March 1991, pp. 51-69.

[16] Y. Rekhter and T. Li, "A Border Gateway Protocol 4 (BGP-4)", *Network Working Group Internet Draft*, Jan. 1994.

[17] W. T. Zaumen, "Simulations in Drama", *Network Information System Center, SRI International*, Menlo Park, California, Jan. 1991.