Source-Tree Routing in Wireless Networks*

J.J. GARCIA-LUNA-ACEVES
Computer Engineering Department
Baskin School of Engineering
University of California
Santa Cruz, CA 95064, USA
jj@cse.ucsc.edu

MARCELO SPOHN
Nokia Wireless Routers
Mountain View, CA 94041, USA
marcelo@rooftop.com

Abstract

We present the source-tree adaptive routing (STAR) protocol and analyze its performance in wireless networks with broadcast radio links. Routers in STAR communicate to its neighbors their source routing trees either incrementally or in atomic updates. Source routing trees are specified by stating the link parameters of each link belonging to the paths used to reach every destination. Hence, a router disseminates link-state updates to its neighbors for only those links along paths used to reach destinations. Simulation results show that STAR is an order of magnitude more efficient than any topology-broadcast protocol, and four times more efficient than ALP, which was the most efficient table-driven routing protocol based on partial link-state information reported to date. The results also show that STAR is even more efficient than the Dynamic Source Routing (DSR) protocol, which has been shown to be one of the best performing on-demand routing protocols.

1. Introduction

Multi-hop packet-radio networks, or ad hoc networks, consist of mobile hosts interconnected by routers that can also move. Routing algorithms for ad hoc networks can be categorized according to the way in which routers obtain routing information, and according to the type of information they use to compute preferred paths. In terms of the way in which routers obtain information, routing protocols have been classified as table-driven and on-demand. In terms of the type of information used by routing protocols, routing protocols can be classified into link-state protocols and distance-vector protocols. Routers running a link-state protocol use topology information to make routing decisions; routers running a distance-vector protocol use distances and, in some cases, path information, to destinations to make routing decisions.

In an on-demand routing protocol, routers maintain path information for only those destinations that they need to contact as a source or relay of information. The basic approach consists of allowing a router that does not know how to reach a destination to send a flood-search message to obtain the path information it needs. The first routing protocol of this type was proposed to establish virtual circuits in the MSE network [25], and there are several more recent examples of this approach (e.g., AODV [2], ABR [1], DSR [10, 7], TORA [24], SSA [19], ZRP [26]). All of the on-demand routing protocols reported to date are based on distances to destinations, and there have been no on-demand link-state proposals to date. On-demand routing protocols differ on the specific mechanisms used to disseminate flood-search packets and their responses, cache the information heard from other nodes' searches, determine the cost of a link, and determine the existence of a neighbor.

In a table-driven algorithm, each router maintains path information for each known destination in the network and updates its routing-table entries as needed. Examples of table-driven algorithms based on distance vectors are the routing protocol of the DARPA packet-radio network [11], DSDV [3], WRP [21], WIRP [15], and least-resistance routing protocols [17]. Prior table-driven approaches to link-state routing in ad hoc networks are based on topology broadcast. However, disseminating complete link-state information to all routers incurs excessive communication overhead in an ad-hoc network because of the dynamics of the network and the small bandwidth available. Accordingly, all link-state routing approaches for ad hoc networks have been based on hierarchical routing schemes [20, 6, 18]. To date, the debate on whether a table-driven or an on-demand routing approach is best for ad hoc networks has assumed that table-driven routing necessarily has to provide shortest paths, when in fact on-demand routing protocols cannot ensure optimum or shortest paths.

We present the source-tree adaptive routing (STAR) protocol as an approach to obtaining efficient routing in ad hoc networks using link-state information. The key contributions of this paper consist of: (a) introducing the most bandwidth-efficient table-driven routing protocol for wireless networks to date, and (b) showing how a table-driven routing protocol can be more efficient than an on-demand routing protocol by exploiting link-state information and allowing paths taken to destinations to deviate from the optimum in order to save bandwidth.

In STAR, a router sends updates to its neighbors regarding the links in its preferred paths to destinations. The links along the preferred paths from a source to each desired destination constitute a *source tree* that implicitly specifies the complete paths from the source to each destination. Each router computes its source tree based on information about adjacent links and the source trees reported by its neighbors, and reports changes to its source tree to all its neighbors incrementally or atomically. The aggregation of adjacent links and source trees reported by neighbors constitutes the partial topology known by a router. Unlike any of the hierarchical link-state routing schemes proposed to date for packet-radio networks [18], STAR does not require backbones, the dissemina-

*This work was supported by the Defense Advanced Research Projects Agency (DARPA) under grant F30602-97-2-0338.

tion of complete cluster topology within a cluster, or the dissemination of the complete inter-cluster connectivity among clusters. Furthermore, STAR can be used with distributed hierarchical routing schemes proposed in the past for both distance-vector or link-state routing [16, 18, 22, 9].

Prior proposals for link-state routing using partial link-state data without clusters [13, 14] require routers to explicitly inform their neighbors which links they use and which links they stop using. In contrast, because STAR sends only changes to the structure of source trees, and because each destination has a single predecessor in a source tree, a router needs to send only updates for those links that are part of the tree and a single update entry for the root of any subtree of the source tree that becomes unreachable due to failures. Routers receiving a STAR update can infer correctly all the links that the sender has stopped using, without the need for explicit delete updates.

Section 2 introduces the network model assumed throughout the rest of the paper. Section 3 describes two different approaches that can be used to update routing information in wireless networks: the optimum routing approach (ORA) and the least-overhead routing approach (LORA), and elicit the reasons why STAR is the first table-driven routing protocol that can adopt LORA. Section 4 describes STAR and how it supports ORA and LORA. Section 5 compares STAR's performance against the performance of other table-driven and on-demand routing protocols. The simulation results show that STAR is four times more bandwidth-efficient than the best-performing link-state routing protocol previously proposed, an order of magnitude more bandwidth-efficient than topology broadcasting, and more bandwidth-efficient than DSR, which is a very efficient on-demand routing protocol [10].

2. Network Model

In STAR, routers maintain a partial topology map of their network. In this paper we focus on flat topologies only, i.e., there is no aggregation of topology information into areas or clusters.

To describe STAR, the topology of a network is modeled as a directed graph $G = (V, E)$, where V is the set of nodes and E is the set of edges connecting the nodes. Each node has a unique identifier and represents a router with input and output queues of unlimited capacity updated according to a FIFO policy. In a wireless network, a node can have connectivity with multiple nodes in a single physical radio link. For the purpose 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 link-level connectivity between A and B and A receives update messages from B reliably. Accordingly, we map a physical broadcast link connecting multiple nodes into multiple point-to-point bidirectional links defined for these nodes. A functional bidirectional link between two nodes is represented by a pair of edges, one in each direction and with a cost associated that can vary in time but is always positive.

All messages, changes in the cost of a link, link failures, and new-neighbor notifications are processed one at a time within a finite time and in the order in which they are detected. Routers are assumed to operate correctly, and information is assumed to be stored without errors.

3. Updating Routes in Wireless Networks

We can distinguish between two main approaches to updating routing information in the routing protocols that have been designed for wireless networks: the *optimum routing approach* (ORA) and the *least-overhead routing approach* (LORA). With ORA, the routing protocol attempts to update routing tables as quickly as possible to provide paths that are optimum with respect to a defined metric. In contrast, with LORA, the routing protocol attempts to provide viable paths, which need not be optimum, causing the least amount of control traffic.

For the case of ORA, the routing protocol can provide paths that are optimum with respect to different types of service (TOS), such as minimum-delay paths or maximum-bandwidth paths. The rest of this paper, however, focuses on a *single* TOS to address the performance of routing protocols providing ORA, and uses shortest-path routing as the single TOS supported for ORA.

On-demand routing protocols such as DSR follow LORA, in that these protocols attempt to minimize control overhead by: (a) maintaining path information for only those destinations with which the router needs to communicate, and (b) using the paths found after a flood search as long as the paths are valid, even if the paths are not optimum.

We can view the flood search messages used in on-demand routing protocols as a form of polling of destinations by the sources. In contrast, in a table-driven routing protocol, it is the destinations who poll the sources, meaning that the sources obtain their paths to destinations as a result of update messages that first originate at the destinations. What is apparent is that flooding of information occurs in both approaches. Interestingly, all the table-driven routing protocols reported to date for ad-hoc networks adhere to ORA, and admittedly have been adaptations of routing protocols developed for wired networks. A consequence of adopting ORA in table-driven routing within a wireless network is that, if the topology of the network changes very frequently, the rate of update messages increases dramatically, consuming the bandwidth needed for user data. The two methods used to reduce the update rate in table-driven routing protocols are clustering and sending updates periodically. Clustering is attractive to reduce overhead due to network size; however, if the affiliations of nodes with clusters change too often, then clustering itself introduces unwanted overhead. Sending periodic updates after long timeouts reduces overhead, and it is a technique that has been used since the DARPA packet-radio network was designed [11]; however, control traffic still has to flow periodically to update routing tables.

Given that both on-demand and table-driven routing protocols incur flooding of information in one way or another, a table-driven routing protocol could be designed that incurs similar or less overhead than on-demand routing protocols by limiting the polling done by the destinations to be the same or less than the polling done by the sources in on-demand routing protocols.

There has been no prior description of a table-driven routing protocol that can truly adhere to LORA, i.e., one that has no need for periodic updates, uses no clustering, and remains quiet as long as the paths available at the routers are valid, even if they are not optimum. The reason why no prior table-driven protocols based on LORA have been reported is that, with the exception of WIRP and WRP, prior protocols have used either distances to destinations, topology maps, or subsets of the topology, to obtain paths to desti-

nations, and none of these types of information permits a router to discern whether the paths it uses are in conflict with the paths used by its neighbors. Accordingly, routers must send updates after they change their routing tables in order to avoid loops, and the best that can be done is to reduce the control traffic by sending such updates periodically.

4. STAR Description

4.1. Overview

In STAR, each router reports to its neighbors the characteristics of every link it uses to reach a destination. The set of links used by a router in its preferred path to destinations is called the *source tree* of the router. A router knows its adjacent links and the source trees reported by its neighbors; the aggregation of a router's adjacent links and the source trees reported by its neighbors constitute a partial *topology graph*. The links in the source tree and topology graph must be adjacent links or links reported by at least one neighbor. The router uses the topology graph to generate its own source tree. Each router derives a routing table specifying the successor to each destination by running a local *route-selection algorithm* on its source tree.

Under LORA, a router running STAR sends updates on its source tree to its neighbors only when it loses all paths to one ore more destinations, when it detects a new destination, when it determines that local changes to its source tree can potentially create long term routing loops, or when the change in the cost of a path to a destination exceeds a threshold. Because each router communicates its source tree to its neighbors, the deletion of a link no longer used to reach a destination is implicit with the addition of the new link used to reach the destination and need not be sent explicitly as an update; a router makes explicit reference to a failed link only when the deletion of a link causes the router to have no paths to one or more destinations, in which case the router cannot provide new links to make the deletion of the failed link implicit.

The basic update unit used in STAR to communicate changes to source trees is the link-state update (LSU). An LSU reports the characteristics of a link; an update message contains one or more LSUs. For a link between router u and router or destination v, router u is called the *head node* of the link in the direction from u to v. The head node of a link is the only router that can report changes in the parameters of that link. LSUs are validated using time stamps, and each router erases a link from its topology graph if the link is not present in the source trees of any of its neighbors.

Figures 1 and 2 specify the main procedures of STAR (for both LORA and ORA) used to update the routing table and the link-state database at a router i. For simplicity, the specification presented in these figures assumes the existence of an underlying protocol that: (a) assures that a router detects within a finite time the existence of a new neighbor and the loss of connectivity with a neighbor, and (b) delivers update messages reliably among neighbors.

4.2. Information Stored and Exchanged

We assume in the rest of the paper that a single parameter is used to characterize a link in one of its directions, which we will call the cost of the directed link. Furthermore, although any type of local route selection algorithm can be used in STAR, we describe STAR assuming that Dijkstra's shortest-path first (SPF) algorithm is used at each router to compute preferred paths.

An LSU for a link (u, v) in an update message is a tuple (u, v, l, t) reporting the characteristics of the link, where l represents the cost of the link and t is the time stamp assigned to the LSU.

A router i maintains a topology graph TG_i, a source tree ST_i, a routing table, the set of neighbors N_i, the source trees ST_x^i reported by each neighbor $x \in N_i$, the topology graphs TG_x^i reported by each neighbor $x \in N_i$, and the system time T_i used to generate time stamps for LSUs. The record entry for a link (u, v) in the topology graph of router i is denoted $TG_i(u, v)$ and is defined by the tuple (u, v, l, t, del), and an attribute p in the tuple is denoted by $TG_i(u, v).p$. The same notation applies to a link (u, v) in ST_i, ST_x^i, and TG_x^i. $TG_i(u, v).del$ is set to TRUE if the link is not in the source tree of any neighbor.

A vertex v in TG_i is denoted $TG_i(v)$. It contains a tuple $(d, pred, suc, d', d'', suc', nbr)$ whose values are used on the computation of the source tree. $TG_i(v).d$ reports the distance of the path $i \rightsquigarrow v, TG_i(v).pred$ is v's predecessor in $i \rightsquigarrow v, TG_i(v).suc$ is the next hop along the path towards v, suc' holds the address of the previous hop towards v, d' corresponds to the previous distance to v reported by suc', d'' is the cost of the path $i \rightsquigarrow v$ the last time the cost of the path changed by Δ, and nbr is a flag used to determine if an update message must be generated when the distance reported by the new successor towards v increases. The same notation applies to a vertex v in ST_i, ST_x^i, and TG_x^i.

The source tree ST_i is a subset of TG_i. The routing table contains record entries for destinations in ST_i, each entry consists of the destination address, the cost of the path to the destination, and the address of the next-hop towards the destination.

The topology graph TG_x^i contains the links in ST_x^i and the links reported by neighbor x in a message being processed by router i, after processing the message $TG_x^i \equiv ST_x^i$.

A router i running LORA also maintains the last reported source tree $ST_i{}'$.

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 the number of hops, or the addition of the latency over the link plus some constant bias.

We refer to an LSU that has a cost infinity as a RESET, $TG_i^i \equiv TG_i$, and $ST_i^i \equiv ST_i$.

4.3. Validating Updates

STAR uses time stamps to validate LSUs. A router either maintains a clock that does not reset when the router stops operating, or asks its neighbors for the oldest known time stamp after it initializes or reboots. Hence, for practical purposes, a time stamp based on 32 bits can be viewed as a monotonically increasing number.

A router receiving an LSU accepts the LSU as valid if the received LSU has a larger time stamp than the time stamp of the LSU stored from the same source, or if there is no entry for the link in the topology graph and the LSU is not reporting an infinite cost. Link-state information for failed links are the only LSUs erased from the topology graph due to aging (which is in the order of an hour after having processed the LSU). LSUs for operational links are erased from the topology graph when the links are erased from the source tree of all the neighbors.

```
NodeUp()
description
  Node i initializes itself
{
  TG_i ← ∅;
  ST_i ← ∅;
  ST'_i ← ∅;
  N_i ← ∅;
  M_i ← FALSE;
  NS_i ← FALSE;
}

NeighborUp(k)
description
  Neighbor protocol reports connectivity
  to neighbor k
{
  N_i ← N_i ∪ {k};
  TG^i_k ← ∅;
  ST^i_k ← ∅;
  sendST ← TRUE;

  if ( LORA and k ∈ TG_i and TG_i(k).pred ≠ null )
  {
    NS_i ← TRUE;
    sendST ← FALSE;
  }

  Update(i, (i, k, l^i_k, T_i));

  if ( sendST )
  {
    MSG_i ← ∅;

    for each ( link (u, v) ∈ ST_i )
      MSG_i ← MSG_i ∪ {( u, v, TG_i(u, v).l,
                         TG_i(u, v).t )};
  }

  Send();
}

NeighborDown(k)
description
  Neighbor protocol reports link
  failure to neighbor k
{
  N_i ← N_i − {k};
  TG^i_k ← ∅;
  ST^i_k ← ∅;

  Update(i, (i, k, ∞, T_i));

  Send();
}

LinkCostChange(k)
description
  Neighbor protocol reports link
  cost change to neighbor k
{
  Update(i, (i, k, l^i_k, T_i));

  Send();
}

Update(k, msg)
description
  Process update message msg
  sent by router k
{
  UpdateTopologyGraph(k, msg);

  if ( k ≠ i )
    BuildShortestPathTree(k);

  BuildShortestPathTree(i);
  UpdateRoutingTable();

  if ( k ≠ i )
    Send();
}
```

```
UpdateTopologyGraph(k, msg)
description
  Update TG_i and TG^i_k from LSUs in msg
{
  for each ( LSU (u, v, l, t) ∈ msg )
  {
    if ( l ≠ ∞ )
      ProcessAddUpdate(k, (u, v, l, t));
    else
      ProcessVoidUpdate(k, (u, v, l, t));
  }
}

ProcessAddUpdate(k, (u, v, l, t))
description
  Update topology graphs TG_i and TG^i_k
  from LSU (u, v, l, t)
{
  if ( (u, v) ∉ TG_i or t > TG_i(u, v).t )
  {
    if ( (u, v) ∉ TG_i )
      TG_i ← TG_i ∪ {(u, v, l, t)};
    else
    {
      TG_i(u, v).l ← l;  TG_i(u, v).t ← t;
    }
  }

  if ( k ≠ i )
  {
    if ( ∃ (r, s) ∈ TG^i_k | r ≠ u and s = v )
      TG^i_k ← TG^i_k − {(r, s)};

    if ( (u, v) ∉ TG^i_k )
      TG^i_k ← TG^i_k ∪ {(u, v, l, t)};
    else
    {
      TG^i_k(u, v).l ← l;  TG^i_k(u, v).t ← t;
    }
  }
  TG_i(u, v).del ← FALSE;
}

ProcessVoidUpdate(k, (u, v, l, t))
description
  Update topology graphs TG_i and TG^i_k from
  LSU (u, v, l, t) reporting link failure
{
  if ( (u, v) ∈ TG_i )
  {
    if ( t > TG_i(u, v).t )
    {
      TG_i(u, v).l ← l;  TG_i(u, v).t ← t;
    }

    if ( k ≠ i and (u, v) ∈ TG^i_k )
    {
      TG^i_k(u, v).l ← l;  TG^i_k(u, v).t ← t;
    }
    TG_i(u, v).del ← FALSE;
  }
}

Send()
{
  if ( MSG_i ≠ ∅ )
    Broadcast message MSG_i;
  MSG_i ← ∅;
}

InitializeSingleSource(k)
{
  for each ( vertex v ∈ TG^i_k )
  {
    TG^i_k(v).d ← ∞;
    TG^i_k(v).pred ← null;
    TG^i_k(v).suc' ← TG^i_k(v).suc;
    TG^i_k(v).suc ← null;
    TG^i_k(v).nbr ← null;
  }

  TG^i_k(k).d ← 0;
}
```

```
BuildShortestPathTree(k)
description
  Construct ST^i_k
{
  InitializeSingleSource(k);
  Q ← set of vertices in TG^i_k;
  u ← ExtractMin(Q);
  newST ← ∅;

  while ( u ≠ null and TG^i_k(u).d < ∞ )
  {
    if ( TG^i_k(u).pred ≠ null and TG^i_k(u).pred ∉ newST )
    {
      (r, s) ← TG^i_k(u).pred;
      newST ← newST ∪ (r, s);

      if ( LORA and k = i )
      {
        if ( i > TG_i(u).suc )
          if ( ∃ x ∈ N_i | TG^i_x(u).suc = i and TG_i(u).suc = x )
            M_i ← TRUE;   // LORA-3 rule
        if ( TG_i(u).suc ≠ TG_i(u).suc' and TG_i(u).suc > i )
          M_i ← TRUE;   // LORA-3 rule
        if ( ∄ (x, y) ∈ ST'_i | y = u )
          M_i ← TRUE;   // LORA-1 rule
        if ( TG_i(u).d'' ≠ ∞ and | TG_i(u).d − TG_i(u).d'' | > Δ )
        {
          M_i ← TRUE;   // LORA-2 rule
          TG_i(u).d'' ← TG_i(u).d;
        }
        w ← TG_i(u).suc;
        if ( w ≠ i )
          path_w_u_cost ← TG_i(u).d − TG_i(i, w).l;
        else
          path_w_u_cost ← 0;
        if ( path_w_u_cost > TG_i(u).d' )
        {
          if ( r = w or TG_i(r).nbr = i )
            TG_i(s).nbr ← i;
          if ( TG_i(s).nbr ≠ i )
            M_i ← TRUE;   // LORA-3 rule
        }
        TG_i(u).d' ← path_w_u_cost;
        TG_i(u).suc' ← TG_i(u).suc;
      }
    }

    for each ( vertex v ∈ adjacency list of TG^i_k(u)
          | TG^i_k(u, v).l ≠ ∞ and NOT TG_i(u, v).del )
    {
      if ( k = i )
      {
        if ( u = i )
          suc ← i;
        else if ( TG_i(u).suc = i )
          suc ← {x | x ∈ N_i and x = u};
        else
          suc ← TG_i(u).suc;
      }
      else
      {
        if ( u = k )
          if ( v = i )  suc ← i;
          else          suc ← k;
        else
          suc ← TG_i(u).suc;
      }
      RelaxEdge(k, u, v, Q, suc);
    }
    if ( Q ≠ ∅ )  u ← ExtractMin(Q);
    else          u ← null;
  }

  UpdateNeighborTree(k, newST);

  if ( k = i )
  {
    if ( LORA and M_i )
    {
      ReportChanges(ST'_i, newST);
      ST'_i ← newST;  NS_i ← FALSE;
    }
    else if ( ORA )
      ReportChanges(ST_i, newST);
    if ( ORA or ( LORA and M_i ) )
      for each ( link (u, v) ∈ TG_i | TG_i(u, v).del = TRUE )
        TG_i ← TG_i − {(u, v)};
    M_i ← FALSE;
  }
  ST^i_k ← newST;  newST ← ∅;
}
```

Figure 1. STAR Specification

We note that, because LSUs for operational links never age out, there is no need for the head node of a link to send periodic LSUs to update the time stamp of the link. This is very important, because it means that STAR does not need to disseminate LSUs for a given link periodically to each router that uses the link in its source tree. This is in contrast to other routing protocols based on sequence numbers or time stamps, together with aging, which age out LSUs and must, therefore, flood LSUs periodically.

```
RelaxEdge(k, u, v, Q, suc)
{
  if ( TG_k^i(v).d > TG_k^i(u).d + TG_k^i(u,v).l or
     ( k = i and TG_k^i(v).d = TG_k^i(u).d + TG_k^i(u,v).l and
       (u,v) ∈ ST_i ))
  {
    TG_k^i(v).d ← TG_k^i(u).d + TG_k^i(u,v).l;
    TG_k^i(v).pred ← TG_k^i(u,v);
    TG_k^i(v).suc ← suc;

    if ( LORA and k = i and TG_i(v).suc' = null )
    {
      // v was an unknown destination
      TG_i(v).suc' ← suc;
      TG_i(v).d'' ← TG_i(v).d;

      if ( suc ≠ i )
        TG_i(v).d' ← TG_i(v).d - TG_i(i,suc).l;
      else
        TG_i(v).d' ← 0;
    }

    Insert(Q, v);
  }
}

ReportChanges(oldST, newST)
description
  Generate LSUs for new links in the router's source tree
{
  for each ( link (u, v) ∈ newST )
    if ( (u,v) ∉ oldST or newST(u,v).t ≠ oldST(u,v).t or NS_i )
      MSG_i ← MSG_i ∪ {(u, v, TG_i(u,v).l, TG_i(u,v).t)};
}

UpdateNeighborTree(k, newST)
description
  Delete links from TG_k^i and report failed links
{
  for each ( link (u, v) ∈ ST_k^i )
  {
    if ( (u,v) ∉ newST )
    {
      // k Has removed (u,v) from its source tree

      if ( LORA and TG_k^i(v).pred = null )
      {
        // LORA-2 rule: k has no path to destination v
        M_i ← TRUE;

        if ( k = i )
          for each ( link (r, s) ∈ TG_i | s = v )
            if ( TG_i(r,s).l = ∞ )
              MSG_i ← MSG_i ∪ {(r, s, TG_i(r,s).l, TG_i(r,s).t)};
      }

      if ( ORA and k = i and ( u = i or TG_i(v).pred = null ))
      {
        // i has no path to destination v or i is the head node

        if ( TG_i(v).pred = null )
          for each ( link (r, s) ∈ TG_i | s = v )
            if ( TG_i(r,s).l = ∞ )
              MSG_i ← MSG_i ∪ {(r, s, TG_i(r,s).l, TG_i(r,s).t)};
        else if ( TG_i(u,v).l = ∞ )
          // i Needs to report failed link
          MSG_i ← MSG_i ∪ {(u, v, TG_i(u,v).l, TG_i(u,v).t)};
      }

      if ( LORA and k = i and TG_i(v).pred = null )
      {
        TG_i(v).d' ← ∞;
        TG_i(v).suc' ← null;
      }

      if ( NOT ( k = i and u = i ))
      {
        if ( (u,v) ∈ TG_k^i )
          TG_k^i ← TG_k^i - {(u,v)};

        if ( TG_i(u,v).l ≠ ∞ and ∄ x ∈ N_i | (u,v) ∈ TG_x^i )
          TG_i(u,v).del ← TRUE;
      }
    }
  }
}
```

Figure 2. STAR Specification (cont.)

4.4. Exchanging Update Messages

How update messages are exchanged depends on the routing approach used (ORA or LORA) and the services provided by the link layer. In this section, we assume that the link layer provides collision-free broadcasts.

For ORA to be supported in STAR, the only rule needed for sending update messages consists of a router sending an update message every time its source tree changes or when a neighbor sends an outdated LSU for a link known to the receiving router.

In an on-demand routing protocol, a router can keep using a path found as long as the path leads to the destination, even if the path does not have optimum cost. A similar approach can be used in STAR, because each router has a complete path to every destination as part of its source tree. To support LORA, router i running STAR reports updates to its source trees in the event of unreachable destinations, new destinations, the possibility of permanent routing loops, and cost of paths exceeding a given threshold. Router i accomplishes this by comparing its source tree against the source trees it has received from its neighbors after any input event, and by sending the updates to its source tree according to the following three rules.

LORA-1: Router i sends a source-tree update when it finds a new destination, or any of its neighbors reports a new destination.

Whenever a router hears from a new neighbor that is also a new destination, it sends an update message that includes the new LSUs in its source tree. Obviously, when a router is first initialized or after a reboot, the router itself is a new destination and should send an update message to its neighbors. Link-level support should be used for the router to know its neighbors within a short time, and then report its links to those neighbors with LSUs sent in an update message. Else, a simple way to implement an initialization action consists of requiring the router to listen for some time for neighbor traffic, so that it can detect the existence of links to neighbors.

LORA-2: Router i sends a source-tree update when the change in the cost of the path to at least one destination exceeds a threshold Δ for router i or any of its neighbors.

In this paper, we assume $\Delta = \infty$, i.e., routers force source-tree updates when destinations become unreachable. When a router processes an input event (e.g., a link fails, an update message is received) that causes *all* its paths through all its neighbors to one or more destination to be severed, the router sends an update message that includes an LSU specifying an infinite cost for the link connecting to the head of each subtree of the source tree that becomes unreachable. The update message does not have to include an LSU for each node in an unreachable subtree, because a neighbor receiving the update message has the sending node's source tree and can therefore infer that all nodes below the root of the subtree are also unreachable, unless LSUs are sent for new links used to reach some of the nodes in the subtree.

LORA-3: Router i sends a source-tree update when:

1. A path implied in the source tree of router i leads to a loop.
2. The new successor chosen to a given destination has an address larger than the address of router i.
3. The reported distance from the new chosen successor n to a destination j is longer than the reported distance from the previous successor to the same destination. However, if the link $(i,\ j)$ fails and n is a neighbor of j, no update message is needed regarding j or any destination whose path from i involves j.

Each time a router processes an update message from a neighbor, it updates that neighbor's source tree and traverses that tree to determine for which destinations its neighbor uses the router processing the update as a relay in its preferred paths. The router then determines if it is using the same neighbor as a relay for any of those destinations. A routing loop is detected if the router and neighbor use each other in the path to any destination, in which case the loop must be broken and the router must send an update message with the corresponding changes.

To explain the need for the second part of LORA-3, we observe that, in any routing loop among routers with unique addresses, one of the routers must have the smallest address in the loop; therefore, if a router is forced to send an update message when it chooses a successor whose address is larger than its own, then it is not possible for all routers in a routing loop to remain quiet after choosing one another, because at least one of them is forced to send an update message, which causes the loop to break when routers update their source trees.

The last part of LORA-3 is needed when link costs can assume different values in different directions, in which case the second part of LORA-3 may not suffice to break loops because the node with the smallest address in the loop may not have to change successors when the loop is formed. The following example illustrates this scenario.

Consider the six-node wireless network shown in Figure 3 and assume that the third part of LORA-3 is not in effect at the routers running STAR. In this example, nodes are given identifiers that are lexicographically ordered, i.e., a is the smallest identifier and f is the largest identifier in the graph. All links and nodes are assumed to have the same propagation delays, and all the links but links (a, b) and (b, c) have unit cost. Figures 3(b) through 3(d) show the source trees according to STAR at the routers indicated with filled circles for the network topology depicted in Figure 3(a). Arrowheads on solid lines indicate the direction of the links stored in the router's source tree. Figure 3(e) shows c's new source tree after processing the failure of link (c, d); we note that c does not generate an update message, because $c > b$ by assumption. Suppose link (b, e) fails immediately after the failure of (c, d), node b computes its new source tree shown in Figure 3(f) without reporting changes to it because a is its new successor to destinations d, e, and f, and $a < b$. A permanent loop forms among nodes a, b, and c. Figure 4 depicts the sequence of events triggered by the execution of the third part of LORA-3 in the same example introduced in Figures 3, after the failures of links (c, d) and (b, e). The figure shows the LSUs generated by the node with filled circle transmitted in an update message to the neighbors, and shows such LSUs in parentheses. The third element in an LSU corresponds to the cost of the link (a RESET has cost $infinity$). Unlike in the previous example, node c transmits an update message after processing the failure of link (c, d) because of the third part of LORA-3; the distance from the new successor b to d and f is longer than from the previous successor d. When link (b, e) fails, node b realizes that the destinations d, e, and f are unreachable and generates an update message reporting the failure of the link connecting to the head of the subtree of the source tree that becomes unreachable. The update message from b triggers the update messages that allow nodes a, b, and c to realize that there are no paths to d, e, and f. A similar sequence of events takes place at the other side of the network partition.

The example shown in Figure 5 illustrates the scenario in which

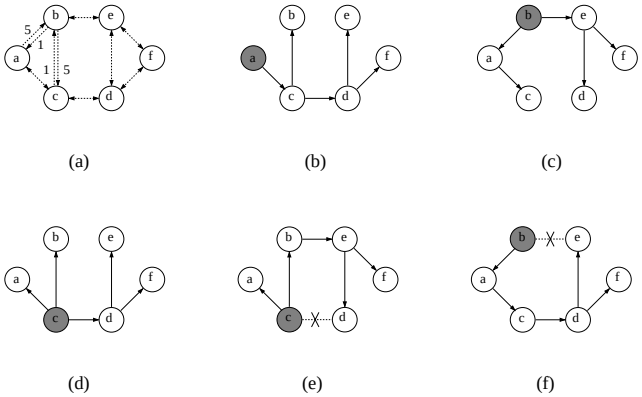


Figure 3. An example of a six node wireless network with routers running STAR without the third part of LORA-3 being in effect.

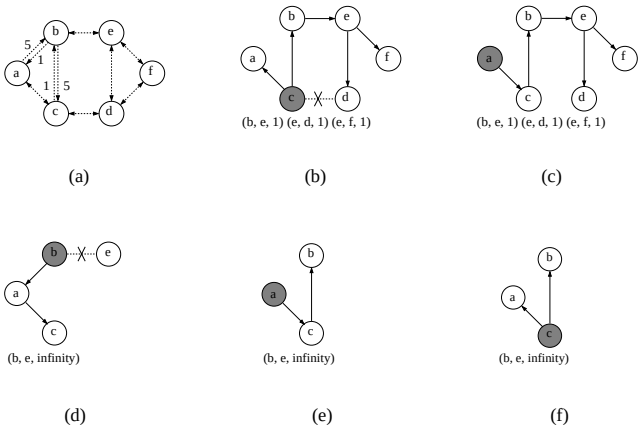


Figure 4. Example of a six-node wireless network with routers running STAR with the third part of LORA-3 being in effect.

Figure 5. The third part of LORA-3 not always triggers the generation of an update message: (a) network topology, and (b) the new source tree of node c after processing the failure of link (c, b).

a router that chooses a new successor to a destination with a larger distance to it does not need to send an update message. Figure 5(b) shows the new source tree of node c after the failure of link (c, b). In this case, c does not need to send an update message because the parent node of the subtree headed by b is a neighbor of c and therefore no permanent loop can be formed.

To ensure that the above rules work with incremental updates

specifying only changes to a source tree, a router must remember the source tree that was last notified to its neighbors. If any of LORA-1 to LORA-3 are satisfied, the router must do one of two things:

- If the new source tree includes new neighbors than those present in the source tree that was last updated, then the router must send its entire source tree in its update, so that new neighbors learn about all the destinations the router knows.
- If the two source trees imply the same neighbors, the router sends only the updates needed to obtain the new tree from the old one.

To ensure that STAR stops sending update messages, a simple rule can be used to determine which router must stop using its neighbor as a relay, such a rule can be, for example, "the router with the smaller address must change its path."

The above rules are sufficient to ensure that every router obtains loop-less paths to all known destinations, without the routers having to send updates periodically. In addition to the ability for a router to detect loops in STAR, the two key features that enable STAR to adopt LORA are: (a) validating LSUs without the need of periodic updates, and (b) the ability to either listen to neighbors' packets or use a neighbor protocol at the link layer to determine who the neighbors of a router are.

4.5 Impact of The Link Layer in LORA

The rules for update-message exchange stated in the previous section assume that an update message is sent reliably to all the neighbors of a router. If the link layer provides efficient reliable broadcast of network-level packets, then STAR can rely on sending an update message only once to all neighbors, with the update message specifying only incremental changes to the router's source tree. The link layer will retransmit the packet as needed to reach all neighbors, so that it can guarantee that a neighbor receives the packet unless the link is broke. An alternative way to provide a reliable exchange of update messages consists of providing collision-free broadcasts of update messages at the medium access control (MAC) layer and implementing the retransmission strategy for update messages as part of STAR itself.

A reliable broadcast service at the link layer can be implemented very efficiently at the link layer or in STAR itself if the MAC protocol used guarantees collision-free transmissions of broadcast packets. A typical example of A MAC protocol that can support collision-free broadcasts is TDMA, and there are several recent proposals that need not rely on static assignments of resources (e.g., FPRP [4], CARTS [27]).

Unfortunately, reliable broadcasting from a node to all its neighbors is not supported in the collision-avoidance MAC protocols that have been proposed [5, 12, 8] or implemented in commercial products for ad hoc networks operating in ISM bands. Furthermore, any link-level or network-level strategy for reliable exchange of broadcast update messages over a contention-based MAC protocol will require substantial retransmissions under high-load conditions and rapid changes to the connectivity of nodes. Therefore, if the underlying MAC protocol does not provide collision-free broadcasts, then STAR (and any table-driven routing protocol for that matter) is better off relying on the approach adopted in the past in the DARPA packet-radio network, whereby a router broadcasts unreliably its update messages to its neighbors, and each update message contains the entire source tree. For STAR to operate correctly with this approach under LORA, routers must prevent the case in which permanent loops are created because an update message is not received by a neighbor due to channel errors or hidden-terminal interference.

When the routers transmit updates over a MAC protocol that does not provide collision-free broadcasting, the following additional mechanisms are needed in STAR: (a) the data packets must record the route traversed, and (b) four additional rules are used to send an update messages. These added rules are used to provide persistence in the exchange of updates, probe neighbor routers for updates when paths to a destination are not known, and break loops detected by the traversal of data packets.

LORA-4: Router i sends its update message as a reliable unicast to the neighbor that makes router i send its update, and all neighbors of i process the update message.

LORA-5: A router sends periodic updates in intervals of 60 seconds while at least one of its neighbors does not report having a path to a destination known to the router; otherwise, periodic updates are transmitted in intervals of 600 seconds or longer.

LORA-6: When router i has a data packet to send to a destination j for which it has no paths, it sends an update message to its neighbors reporting the absence of a path to j. This message acts as a query, because any neighbor with a path to j receiving the message will generate an update message and send it reliably to router i. While router i has no path to j, it retransmits its update message in intervals of 600 milliseconds, 6 seconds, and 60 seconds, and then backs off to periodic updates transmitted in intervals of 600 seconds or longer.

LORA-7: Router i receives a data packet to destination j and one of the routers in the traversed path is in i's path to the destination, the data packet is discarded and a ROUTE-REPAIR update message is generated to break the loop. A ROUTE-REPAIR contains the complete source tree of the sender's router and the *route repair path*, and is transmitted reliably to the router in the head of the route repair path. The *route repair path* corresponds to the path $i \rightsquigarrow x$, where x is the last router in the data packet's traversed path that is first found in the path $i \rightsquigarrow j \in ST_i$. When a router receives a ROUTE-REPAIR update it removes itself from the route repair path and transmits a ROUTE-REPAIR with its source tree to the head of the route repair path. When a router detects a loop it will only transmit a ROUTE-REPAIR update to neighbor k if 30 seconds have elapsed since the last time a ROUTE-REPAIR was sent to k.

5. Performance Evaluation

STAR has the same communication, storage, and time complexity than ALP [14] and efficient table-driven distance-vector routing protocols proposed to date (e.g., WRP [21]). However, worst-case performance is not truly indicative of STAR's performance; accordingly, we ran a number of simulation experiments to compare STAR's average performance against the performance of table-driven and on-demand routing protocols. The simulation study was

conducted in the C++ Protocol Toolkit (CPT) simulator environment, in which the protocol stack implementation in the simulation runs the very same code used in a real embedded wireless router and IP (Internet Protocol) is used as the network protocol.

The link layer implements a medium access control (MAC) protocol similar to the IEEE 802.11 standard and the physical layer is based on a direct sequence spread spectrum radio with a link bandwidth of 1 Mbit/sec. The neighbor protocol is configured to report loss of connectivity to a neighbor if the probation of the link fails in a period of about 10 seconds.

STAR based on ORA was compared against two other table-driven routing protocols, and STAR based on LORA was compared with an on-demand routing protocol. The simulation experiments use 20 nodes forming an ad-hoc network, moving over a flat space (5000m x 7000m), and initially randomly distributed at a density of one node per square kilometer. Nodes move in the simulation according to the "random waypoint" model [10]. Each node begins the simulation by remaining stationary for $pause\ time$ seconds. It then selects a random destination and moves to that destination at a speed of 20 meters per second for a period of time uniformly distributed between 5 and 11 seconds. Upon reaching the destination, the node pauses again for $pause\ time$ seconds, selects another destination, and proceeds there as previously described, repeating this behavior for the duration of the simulation.

5.1. Comparison with Table-Driven Protocols

We chose to compare STAR against the traditional link-state approach and ALP [14]. The traditional link-state approach (denoted by TOB for topology broadcast) corresponds to the flooding of link states in a network, or within clusters coupled with flooding of inter-cluster connectivity among clusters. ALP is a routing protocol based on partial link-state information that we have previously shown to outperform prior table-driven distance-vector and link-state protocols. ALP's efficiency is derived from the fact that a router running ALP does not report to its neighbors the deletion of a link from its preferred paths if the cost of the link has not increased; however, the router may be forced to report the deletion of such a link subsequently if the cost of the link increases or the link becomes unreachable.

For these simulations STAR uses ORA, because both ALP and TOB attempt to provide paths that are optimum with respect to a defined metric. The three protocols rely on the reliable delivery of broadcast packets by the link layer. We ran our simulations with movement patterns generated for five different pause times: 0, 30, 45, 60, and 90 seconds. A pause time of 0 seconds corresponds to continuous motion. The simulation time in all the simulation scenarios is of 900 seconds.

Table 1 summarizes the behavior of the three protocols according to the $pause\ time$ of the nodes. The table shows the number of link connectivity changes and the total number of update packets generated by the routing protocols; ALP generates on average more than four times the number of update packets generated by STAR, and TOB generates more than 10 times the number of packets generated by STAR. The performance of ALP and TOB for $pause\ time$ 0 could not be assessed because the amount of update packets generated by the routers lead to congestion at the link layer.

Because STAR can be used in combination with any clustering scheme proposed in the past for packet-radio networks, it is clear from this study that STAR should be used instead of ALP and topology broadcast for the provision of QoS routing in packet radio networks, given that any overhead traffic associated with clustering would be equivalent for STAR, ALP, and topology broadcast.

Pause Time	Connectivity Changes	Packets Generated		
		STAR	ALP	TOB
0	1090	2542	–	–
30	154	411	1765	5577
45	102	262	1304	3908
60	90	239	1144	2502
90	50	138	623	1811

Table 1. Average performance of STAR, ALP, and TOB.

5.2. Comparison with On-Demand Protocols

We compare STAR using LORA with DSR, because DSR has been shown to produce the smallest number of update messages among on-demand routing protocols [10]. Our simulation experiments use the same methodology used recently to evaluate DSR and other on-demand routing protocols [10]. To run DSR in our simulation environment, we ported the ns2 code available from [23] into the CPT simulator. There are only two differences in our DSR implementation with respect to that used in [10]: (1) in the embedded wireless routers and simulated protocol stack we used there is no access to the MAC layer and cannot reschedule packets already scheduled for transmission over a link (however, this is the case for all the protocols we simulate); and (2) routers cannot operate their network interfaces in *promiscuous mode* because the MAC protocol operates over multiple channels and a router does not know on which channels its neighbors are transmitting, unless the packets are meant for the router. Both STAR and DSR can buffer 20 packets that are awaiting discovery of a route through the network. Routers running STAR exchange update messages according to LORA (rules LORA-1 to LORA-7) because the underlying MAC protocol used in the simulations does not provide collision-free broadcasts.

The overall goal of the simulation experiments was to measure the ability of the routing protocols to react to changes in the network topology while delivering data packets to their destinations. To do this, we applied three different communication patterns to the simulated network, corresponding to 8, 14, and 20 data flows. The total workload in the three scenarios was the same and consisted of 32 data packets/sec. Each continuous-bit-rate (CBR) source generated four packets/sec in the scenario with eight flows. Each CBR source generated 1.6 packets/sec in the scenario with 20 sources. In the scenario with 14 flows, there were 7 flows from distinct CBR sources to the same destination D, generating an aggregate of four packets/sec and seven flows having D as the CBR source and the other seven sources of data as destinations. In each scenario, the number of unique destinations was eight and the packet size was 64 bytes. The data flows were started at times uniformly distributed between 20 and 120 seconds (we chose to start the flows after 20 seconds of simulated time to give some time to the Link Layer for determining the set of nodes that are neighbors of the routers).

The protocol evaluations are based on the simulation of 20 wireless nodes with movement patterns generated for five different pause times: 0, 15, 30, 45, and 60 seconds. The simulated time is of 900 seconds and 1800 seconds for the simulation scenarios with pause time 0, and 900 seconds for pause times other than 0.

Pause Time	Num. Flows	Update Pkts Sent		Data Pkts Delivered		Data Pkts Generated
		STAR	DSR	STAR	DSR	
0	8	908	791	15110	14740	24100
	14	930	1460	15845	10975	25917
	20	916	3122	13689	6830	23718
15	8	615	460	19544	20831	24396
	14	636	702	23027	23210	25989
	20	686	1535	17254	10129	23649
30	8	559	350	20180	20492	24160
	14	551	464	23086	23228	25892
	20	580	763	19929	18341	23716
45	8	517	280	21685	22683	24100
	14	526	2352	23776	20481	25917
	20	507	1880	20749	19898	23731
60	8	522	482	22536	19102	24100
	14	507	1357	24473	23436	25917
	20	493	744	22218	21899	23775

Table 2. Performance of STAR and DSR (900 sec).

Pause Time	Num. Flows	Update Pkts Sent		Data Pkts Delivered		Data Pkts Generated
		STAR	DSR	STAR	DSR	
0	8	1583	1963	33068	32650	52900
	14	1582	3249	21085	21830	54716
	20	1609	5199	27040	21755	52518

Table 3. Performance of STAR and DSR (1800 sec).

Number of Flows	Protocol	Number of Hops					
		1	2	3	4	5	6
8	STAR	94.0	4.1	1.9			
	DSR	64.9	31.2	2.6	1.3		
14	STAR	76.0	16.4	4.2	3.0	0.4	
	DSR	64.1	26.9	4.0	4.5	0.5	
20	STAR	90.8	6.4	1.6	1.1	0.1	
	DSR	61.9	32.4	5.1	0.3		0.3

Table 4. Number of hops traversed by data packets (900 sec, pause time 0).

Pause Time	Connectivity Changes
0	1461
15	605
30	424
45	350
60	322

Table 5. Changes in link connectivity (900 sec).

Tables 2 and 3 summarize the behavior of STAR and DSR according to the simulated time. The tables show the total number of update packets transmitted by the nodes and the total number of data packets delivered to the applications for the three simulated workloads. Table 4 shows the number of hops traversed by data packets during 900 seconds of simulated time when nodes are in continuous motion. The total number of update packets transmitted by routers running STAR varies with the number of changes in link connectivity while DSR generates control packets based on both variation of changes in connectivity and the type of workload inserted in the network. Routers running STAR generated fewer update packets than DSR in most of the simulated scenarios for 900 seconds of simulated time, the difference increased significantly when the number of flows in the network was 20 (routers running DSR sent three times more control packets than STAR when nodes were in continuous motion). Both STAR and DSR were able to deliver about the same number of data packets to the applications in the simulated scenarios with 8 and 14 flows. When we increased the number of sources of data from 8 to 20 nodes, while inserting the same number of data packets in the network (32 packets/sec), we observed that STAR was able to deliver as much as twice the amount of data packets delivered by DSR when nodes were in continuous motion. The changes in the network topology during 1800 seconds of simulated time made DSR to send more control packets than STAR for all workloads.

The MAC layer discards all packets scheduled for transmission to a neighbor when the link to the neighbor fails, which contributes to the high loss of data packets seen by nodes. In DSR, each packet header carries the complete ordered list of routers through which the packet must pass and may be updated by nodes along the path towards the destination. The low throughput achieved by DSR for the case of 20 sources of data is due to the poor choice of source routes the routers make, leading to a significant increase in the number of ROUTE ERROR packets generated. Data packets are also discarded due to lack of routes to the destinations because the network may become temporarily partitioned or because the routing tables have not converged in the highly dynamic topology we simulate.

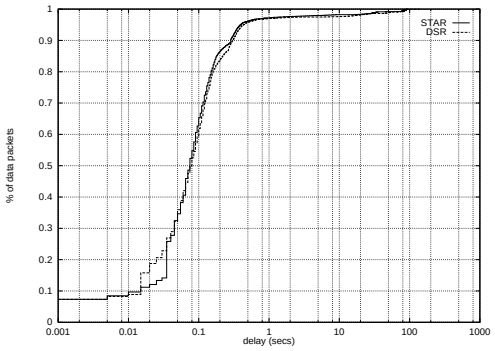
Figures 6(a) through 6(c) show the cumulative distribution of packet delay experienced by data packets when nodes were in continuous motion during 900 seconds of simulated time, for a workload of 8, 14, and 20 flows respectively. We note that the distribution of the latency is about the same for both STAR and DSR.

The number of destinations was set to just 40% of the number of nodes in the network in all the scenarios, which favors DSR by limiting the number of flood searches needed from each node. For the cases in which all network nodes receive data, STAR would introduce no extra overhead while DSR would be severely penalized. It is also important to note the low ratio of update messages generated by STAR compared to the number of changes in link connectivity (Table 5), when we run the simulations for 1800 seconds of simulated time there were 2792 changes in link connectivity.

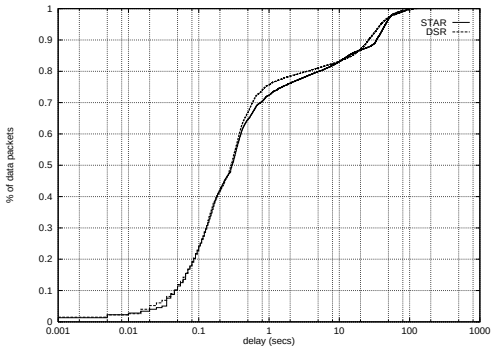
We note that, in cases where routers fail or the network becomes partitioned for extended time periods, the bandwidth consumed by STAR is much the same as in scenarios in which no router fails, because all that must happen is for updates about the failed links to unreachable destinations to propagate across the network. In contrast, DSR and several other on-demand routing protocols would continue to send flood-search messages trying to reach the failed destination, which would cause a worst-case bandwidth utilization for DSR. To illustrate the impact the failure of a single destination has in DSR we have re-run the simulation scenario with 8 flows present in the network for 1800 seconds making one of the destinations fail after 900 seconds of simulated time, routers running STAR sent 1823 update packets while routers running DSR sent 3043 update packets. The existence of a single flow of data to a destination that was unreachable for 900 seconds made DSR generate 55% more update packets while STAR experienced an increase of 15% (see Table 3).

6. Conclusions

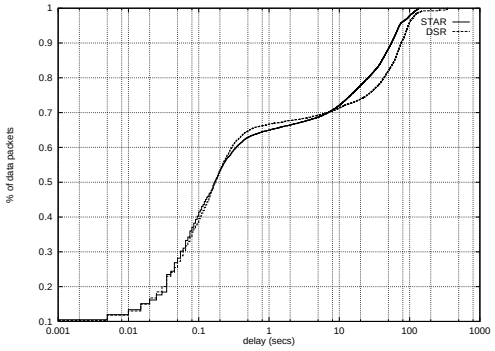
We have presented STAR, a link-state protocol that incurs the smallest communication overhead of *any* prior table-driven routing protocol, and also incurs on average less overhead than on-demand routing protocols. STAR accomplishes its bandwidth efficiency by: (a) disseminating only that link-state data needed for routers to reach destinations; (b) exploiting that information to ascertain when update messages must be transmitted to detect new destinations, unreachable destinations, and loops; and (c) allowing paths to deviate from the ideal optimum while not creating permanent



(a)



(b)



(c)

Figure 6. Cumulative distribution of packet delay experienced by data packets for a workload of (a) 8 flows, (b) 14 flows, and (c) 20 flows (900 sec, pause time 0).

loops. Because STAR can be used with any clustering mechanism proposed to date, these results clearly indicate that STAR is a very attractive approach for routing in packet-radio networks. Perhaps more importantly, the approach we have introduced in STAR for least-overhead routing opens up many research avenues, such as developing similar protocols based on distance vectors and determining how route aggregation and multicasting work under LORA.

7. References

[1] C-K. Toh. *Wireless ATM & Ad-Hoc Networks*. Kluwer, 1996.

[2] C. Perkins. *Ad-Hoc On Demand Distance Vector (AODV) Routing*. draft-ietf-manet-aodv-00.txt, 1997.

[3] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *Proc. ACM SIGCOMM 94*, October 1994.

[4] C. Zhu and S. Corson. A five phase reservation protocol (FPRP) for mobile ad-hoc networks. *Proc. IEEE INFOCOM 98*, 1998.

[5] C.L. Fullmer and J.J. Garcia-Luna-Aceves. Solutions to hidden terminal problems in wireless networks. *Proc. ACM SIGCOMM 97*, September 1997.

[6] C.V. Ramamoorthy and W. Tsai. An adaptive hierarchical routing algorithm. *Proceedings of IEEE COMPSAC '83*, pages 93–104, 1983.

[7] D. Johnson and D. Maltz. Protocols for adaptive wireless and mobile networking. *IEEE Pers. Commun.*, 3(1), February 1996.

[8] IEEE. *P802.11–Unapproved Draft: Wireless LAN Medium Access Control (MAC) and Physical Specifications*. IEEE, 1996.

[9] J. Behrens and J.J. Garcia-Luna-Aceves. Hierarchical routing using link vectors. *Proc. IEEE INFOCOM 98*, April 1998.

[10] J. Broch et al. A performance comparison of multi-hop wireless ad hoc network routing protocols. *Proc. ACM MOBICOM 98*, October 1998.

[11] J. Jubin and J. Tornow. The DARPA packet radio network protocols. *Proceedings of the IEEE*, 75(1), January 1987.

[12] J.J. Garcia-Luna-Aceves and A. Tzamaloukas. Reversing the collision avoidance handshake in wireless networks. *Proc. ACM/IEEE Mobicom 99*, August 1999.

[13] J.J. Garcia-Luna-Aceves and J. Behrens. Distributed, scalable routing based on vectors of link states. *IEEE Journal on Selected Areas in Communications*, 13(8), 1995.

[14] J.J. Garcia-Luna-Aceves and M. Spohn. Scalable link-state internet routing. *Proc. IEEE International Conference on Network Protocols (ICNP 98)*, October 1998.

[15] J.J. Garcia-Luna-Aceves et al. Wireless internet gateways (WINGS). *Proc. IEEE MILCOM'97*, November 1997.

[16] L. Kleinrock and F. Kamoun. Hierarchical routing for large networks: Performance evaluation and optimization. *Computer Networks*, 1:155–174, 1977.

[17] M. Pursley and H.B. Russell. Routing in frequency-hop packet radio networks with partial-band jamming. *IEEE Trans. Commun.*, 41(7):1117–1124, 1993.

[18] M. Steenstrup (Ed.). *Routing in Communication Networks*. Prentice-Hall, 1995.

[19] R. Dube et al. Signal stability-based adaptive routing (SSA) for ad-hoc mobile networks. *IEEE Pers. Commun.*, February 1997.

[20] R. Ramanathan and M. Steenstrup. Hierarchically-organized, multihop mobile wireless networks for quality-of-service support. *ACM Mobile Networks and Applications*, 3(1):101–119, 1998.

[21] S. Murthy and J.J. Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *CM Mobile Networks and Applications Journal*, 1996.

[22] S. Murthy and J.J. Garcia-Luna-Aceves. Loop-free internet routing using hierarchical routing trees. *Proc. IEEE INFOCOM 97*, April 1997.

[23] The CMU Monarch Project. *Wireless and Mobility Extensions to ns-2 - Snapshot 1.0.0-beta*. URL: http://www.monarch.cs.cmu.edu/cmu-ns.html, 1998.

[24] V. Park and M. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. *Proc. IEEE INFOCOM 97*, April 1997.

[25] V.O.K. Li and R. Chang. Proposed routing algorithms for the us army mobile subscriber equipment (MSE) network. *Proc. IEEE MILCOM'86*, October 1986.

[26] Z. Haas and M. Pearlman. The zone routing protocol for highly reconfigurable ad-hoc networks. *Proc. ACM SIGCOMM 98*, August 1998.

[27] Z. Tang and J.J. Garcia-Luna-Aceves. A protocol for topology-dependent transmission scheduling. *Proc. IEEE Wireless Communications and Networking Conference 1999 (WCNC 99)*, September 1999.