SOURCE TREE ADAPTIVE ROUTING (STAR) PROTOCOL

The Source-Tree Adaptive Routing (STAR) protocol is intended for use by nodes (static and mobile) in an ad hoc network or an internet. A
router in STAR communicates to its neighbors the parameters of its source routing tree, which consists of each link that the router needs to reach every known destination (and address range) in the ad hoc network or internet. To conserve transmission bandwidth and energy, a router communicates changes to its source routing tree only when the router detects new destinations, the possibility of looping, or the possibility of node failures or network partitions.

Introduction

This document describes the Source-Tree Adaptive Routing (STAR) protocol [11, 12], a protocol developed in the SPARROW project part of the DARPA GloMo program.

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 or path information to destinations to make routing decisions. Our characterization of distance-vector routing protocols is broader than in other documents but is consistent with our prior publications.

In an on-demand routing protocol, routers maintain routing 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 [19], and there are several more recent examples of this approach (e.g., AODV [23], ABR [30], DSR [15], TORA [21], SSA [6], ZRP [13]). 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 [16i], DSDV [22], WRP [27], WIRP [9], and least-resistance routing protocols [24]. Prior table-driven approaches to link-state routing in packet-radio 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 packet-radio networks have been based on hierarchical routing schemes [25, 26, 29].

A key issue in deciding which type of routing protocol is best for ad hoc networks is the communication overhead incurred by the protocol. Because data and control traffic share the same communication bandwidth in the network, and because untethered routers use the same energy source to transmit data and control packets, computing minimum-cost (e.g., least interference) paths to all destinations at the expense of considerable routing-update traffic is not practical in ad hoc networks with untethered nodes and dynamic topologies. The routing protocol used in an ad hoc network should incur as little communication overhead as possible to preserve battery life at untethered routers and to leave as much bandwidth as possible to data traffic.

To date, the debate on whether a table-driven or an on-demand routing approach is best for wireless networks has assumed that table-driven routing necessarily has to provide optimum (e.g., shortest-path) routing, when in fact on-demand routing protocols cannot ensure optimum paths. STAR is the first example of a table-driven routing protocol that is as efficient as 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.

Assumptions and Terminology

We make very few assumptions for the efficient operation of STAR. STAR assumes that the ad hoc network is in fact a wireless internet in which IP hosts are connected through subnets or point-to-point links to routers. Currently, STAR operates on top of IP, just like an Internet routing protocol does.

STAR can operate in ad hoc networks based on very different medium access control (MAC) protocols, which need not permit promiscuous listening of transmissions.
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 updated according to a FIFO policy. In a wireless network, a node can have connectivity with multiple nodes over 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.

For brevity, an underlying protocol (which we call the neighbor protocol) is assumed to assure that a router detects within a finite time the existence of a new neighbor and the loss of connectivity with a neighbor. In practice, simple techniques can be used by a router to decide whether or not it maintains connectivity with a neighbor router by keeping track of packets received from neighboring nodes. In ad hoc networks with simple waveforms in which a node can eavesdrop on its neighbors, a router can keep track of transmissions from other neighbors to estimate who its neighbors are. Future specifications of STAR will describe examples of such techniques.

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.

A pseudocode description of STAR is presented in [12]. A subsequent version of this draft will specify STAR in more detail.

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 according to a given performance metric, which need not be optimum, to incur 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, maximum bandwidth, least amount of interference, maximum available battery life, or combinations of metrics. Multiple TOS can be supported in a routing protocol.

On-demand routing protocols 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. On-demand routing protocols can be applied to support multiple TOS; an obvious approach is to obtain paths of different TOS using separate flood searches. However, we assume that a single TOS is used in the network. ORA is not an attractive or even feasible approach in on-demand routing protocols, because flooding the network frequently while trying to optimize existing paths with respect to a cost metric of choice consumes the available bandwidth and can make the paths worse while trying to optimize them.

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 some form of information flooding 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 [16]; however, control traffic still has to flow periodically to update routing tables.

A nice feature of such routing protocols as DSR [15] and WIRP [9] is

Garcia-Luna-Aceves, Spohn, Beyer

[Page 5]
that these protocols remain quiet when no new update information has
to be exchanged; they have no need for periodic updates. Both proto-
cols take advantage of promiscuous listening of any packets sent by
router’s neighbors to determine the neighborhood of the router.

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 des-
tinations to be the same or less than the polling done by the sources
in on-demand routing protocols. However, 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 routing protocols have been reported based on LORA
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 destinations, 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. Accord-
ingly, routers must send updates after they change their routing
tables in order to avoid long-term routing loops, and the best that
can be done is to reduce the control traffic by sending such updates
periodically. STAR is the first table-driven routing protocol that
implements LORA.

STAR 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 adja-
cent 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 neigh-
bor. 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.

Depending on the bandwidth available in an ad hoc network, the ORA or
LORA approach can be used to updating routing information. STAR
supports both.

Under ORA, updates are sent when source trees change. Under LORA, a router running STAR sends updates on its source tree to its neighbors only when it loses all paths to one or more destinations, when it detects a new destination, or when it determines that local changes to its source tree can potentially create long term routing loops. 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 headnode 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 sequence numbers, 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. The head of a link does not periodically send LSUs for the link, because link-state information never ages out.

Unlike any of the hierarchical link-state routing schemes proposed to date for packet-radio networks [29], STAR does not require backbones, the dissemination 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 [18, 29, 28, 2].

Prior proposals for link-state routing using partial link-state data without clusters [8, 10] 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.
Conceptual Data Structures

To execute STAR, a router maintains a topology graph, a source tree, a routing table, the set of neighbors, the source trees reported by each neighbor, and the topology graphs reported by each neighbor.

The record entry for a link from u to v in the topology graph of router i is denoted $TG_{i(u, v)}$ and is defined by the tuple $(u, v, l, sn, del)$, 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 $ST_{i}$, $ST_{i x}$, and $TG_{i x}$. $TG_{i(u, v)}.del$ 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 $(d, pred, suc, suc', nbr)$ whose values are used on the computation of the source tree. $TG_{i(v)}.d$ reports the distance of the path $i$'s predecessor in $i$'s 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 $suc'$, 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 $ST_{i}$, $ST_{i x}$, and $TG_{i x}$.

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 $TG_{i x}$ contains the links in $ST_{i x}$ and the links reported by neighbor $x$ in a message being processed by router $i$, after processing the message $TG_{i x} = ST_{i x}$.

A router $i$ running LORA also maintains the last reported source tree $ST_{i}^\prime$.

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, $TG_{i}$ sub $i$, and $ST_{i}$ sub $i = ST_{i}$.
Validating Updates in STAR

Because of delays in the routers and links of an internetwork, update messages sent by a router may propagate at different speeds along different paths. Therefore, a given router may receive an LSU from a neighbor with stale link-state information, and a distributed termination-detection mechanism is necessary for a router to ascertain when a given LSU is valid and avoid the possibility of LSUs circulating forever. STAR uses sequence numbers to validate LSUs. A sequence number associated with a link consists of a counter that can be incremented only by the head node of the link. For convenience, a router $i$ needs to keep only a counter $SN_{i}$ for all the links for which it is the head node, which simply means that the sequence number a router gives to a link for which it is the head node can be incremented by more than one each time the link parameters change value.

A router receiving an LSU accepts the LSU as valid if the received LSU has a larger sequence number than the sequence number 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.

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 sequence number of the link. This is very important, because it means that STAR does not need periodic update messages to validate link-state information like OSPF [20] and every single routing protocol based on sequence numbers or time stamps does!

To simplify our description, the specification in the rest of this document describes STAR as if unbounded counters were available to keep track of sequence numbers. Future specifications of STAR will be more precise.

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. The rest of this section describes how LORA and ORA can be supported in STAR and describes the impact of the link layer on the way in which...
Supporting LORA and ORA in STAR

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 should send update messages according to the following three rules, which inform routers of unreachable destinations, new destinations, and update topology information to prevent permanent routing loops. Router $i$ implements these rules by comparing its source tree against the source trees it has received from its neighbors.

[LORA-1:]
Router $i$ 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:]
At least one destination becomes unreachable to router $i$ or any of its neighbors.

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:
This rule has three parts:

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 as a relay in its preferred paths. The router then determines if it is using the same neighbor as a relay for any of the same destinations. A routing loop is detected if the router and neighbor use each other as relay 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.

Examples of why these rules are used are presented in [11, 12]. The next version of this document will include detailed examples as well.
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:

1. 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.

2. 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 loopless 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.

If ORA is 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.

The rules for update-message exchange stated above assume that an update message is sent reliably to all the neighbors of a router. This is a very realistic assumption, because STAR working under LORA generates far fewer update messages than the topology changes that occur in the network. However, if preserving bandwidth is of utmost importance and the underlying link protocol is contention-based, additional provisions must be taken, which we describe next.

Impact of The Link Layer

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.

A reliable broadcast service at the link layer can be implemented very efficiently if the MAC protocol being used guarantees collision-free transmissions of broadcast packets. A typical example of MAC protocols 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 [31]).

Unfortunately, reliable broadcasting from a node to all its neighbors is not supported in the collision-avoidance MAC protocols that have been proposed and implemented for ad hoc networks [1, 7, 14, 17]. 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 over which efficient reliable broadcasting can be built, 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. For STAR this means that 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. A simple example is a two-node loop between two neighbor routers, $A$ and $B$, in which the neighbor with the smaller address $A$ sends an update to its neighbor $B$ specifying that $A$ is using $B$ to get to at least one destination $D$, but the message does not reach $B$, which then starts using $A$ to reach $D$.

An additional simple rule to send an update message can be used to eliminate permanent looping due to lost packets using unreliable broadcasting:

[LORA-4:

A data packet is received from a neighbor who, according to its source tree, is in the path to the destination specified in the data packet. This rule is needed to eliminate permanent looping under unreliable broadcasting.]
Details on The Processing of Input Events

The next version of this draft will provide a detailed account of how input events (topology changes and update messages) are processed by a router running STAR. Details can be found in [11, 12].

The neighbor set of the router is empty initially, and the sequence number counter is set to zero.

If the neighbor protocol reports a new link to a neighbor $k$ (or the mechanisms used instead of the neighbor protocol based on the reception of neighbor packets), the router then sends an update message to its neighbor.

Router $i$ updates its source tree when router $i$ receives an update message from neighbor $k$ or when the parameters of an outgoing link have changed. First, the topology graphs $TG_{i,k}$ and $TG_{i,i}$ are updated, then the source trees $ST_{i,k}$ and $ST_{i,i}$ are updated, which may cause the router to update its routing table and to send its own update message.

The state of a link in the topology graph $TG_{i,i}$ is updated with the new parameters for the link if the link-state update in the received message is valid, i.e., if the LSU has a larger sequence number than the sequence number of the link stored in $TG_{i,i}$.

The parameters of a link in $TG_{i,k}$ are always updated when processing an LSU sent by a neighbor $k$, even if the link-state information is outdated, because they report changes to the source tree of the neighbor. A node in a source tree $ST_{i,k}$ can have only one link incident to it. Hence, when $i$ receives an LSU for link $(u, v)$ from $k$ the current incident link $(u', v)$ to $v$, with $u'$ being different than $u$, is deleted from $TG_{i,k}$.

The information of an LSU reporting the failure of a link is discarded if the link is not in the topology graph of the router.

A shortest-path algorithm (SPF) based on Dijkstra’s SPF is run on the updated topology graph $TG_{i,k}$ to construct a new source tree $ST_{i,k}$, and then run on the topology graph $TG_{i,i}$ to construct a new source tree $ST_{i,i}$.

The incident link to a node $v$ in router’s new source tree is different from the link in the current source tree $ST_{i,i}$ only if the cost of the path to $v$ has decreased or if the incident link in $ST_{i,i}$ was deleted from the source trees of all neighbors.
A new source tree $\textit{newST}$ for a neighbor $k$, including the router’s new source tree, is then compared to the current source tree $\textit{ST sup i sub k}$, and the links that are in $\textit{ST sup i sub k}$ but not in $\textit{newST}$ are deleted from $\textit{TG sup i sub k}$. After deleting a link $(u, v)$ from $\textit{TG sup i sub k}$ the router sets $\textit{TG sub i (u, v).del}$ to TRUE if the link is not present in the topology graphs $\textit{TG sup i sub x}$ for all $x$ in $\textit{N sub i}$.

If a destination $v$ becomes unreachable, i.e., there is no path to $v$ in the new source tree $\textit{newST}$, then LSUs will be broadcast to the neighbors for each link in the topology graph $\textit{TG sub i}$ that have $v$ as the tail node of the link and a link cost infinity.

The new router’s source tree $\textit{newST}$ is compared to the last reported source tree ($\textit{(ST sub i)' for LORA and ST sub i for ORA}$), and an update message that will be broadcast to the neighbors is constructed from the differences of the two trees. An LSU is generated if the link is in the new source tree but not in the current source tree, or if the parameters of the link have changed. For the case of a router running LORA, the source trees are only compared with each other if at least one of the three conditions (LORA-1, LORA-2, and LORA-3) is met, i.e., $\textit{M sub i = TRUE}$.

If the new router’s source tree was compared against the last reported source tree then the router removes from the topology graph all the links that are no longer used by any neighbor in their source trees.

Finally, the current shortest-path tree $\textit{ST sup i sub k}$ is discarded and the new one becomes the current source tree. The router’s source tree is then used to compute the new routing table, using for example a depth-first search in the shortest-path tree.

Packet Formats

An update message in STAR has the following format:
The header specifies control information for the proper processing of acknowledgments to prior updates and LSUs contained in the update message. Acknowledgments are specified in the acknowledgment list and LSUs are specified in the LSU list of the message.

There are four types of packets in STAR: GUM, PUM, TUM, RUM, and SUM. A SUM packet (Start Update Message) is broadcast or unicast and indicates that the router has restarted operation. A GUM packet (Goodbye Update Message) is broadcast and indicates that the router will be out of reach for some time and should not be used as a neighbor. A PUM packet (Partial Update Message) is broadcast or unicast and contains a partial update. A TUM packet (Total Update Message) is broadcast or unicast and contains an atomic update. A RUM packet (Reset Update Message) is broadcast and informs neighbors to disregard the sequence numbers used in prior update messages.

Packet Headers

The next version of this document will provide more details on the packet formats used in STAR, and the REQUIRED and the OPTIONAL information it uses.

The packet header for packet types: GUM, PUM, TUM, and RUM is the following:
The packet header for packet type SUM is the following:

```
| Version | Type  | Num. of ACKs | Sequence Number |
+---------+-------+-------------+----------------|
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
```

The current version is Protocol version 1.

SUM (0x1) : Start Update Message (broadcast/unicast)
GUM (0x2) : Goodbye Update Message (broadcast)
PUM (0x3) : Partial Update Message (broadcast/unicast)
TUM (0x4) : Total Update Message (broadcast/unicast)
RUM (0x5) : Reset Update Message (broadcast)

The Number of ACKs denotes the number of entries in the Acknowledgment List.

The Sequence Number has values in the range [0, 2^15 - 1]. The Most Significant Bit is used to request acknowledgments when set in the sequence numbers specified at the Acknowledgment List.

The Router ID:

The first packet sent to a new neighbor (SUM) must carry the router ID so that the neighbor get to know how to uniquely
identify the router.

Acknowledgment List Entry

<table>
<thead>
<tr>
<th>Neighbor’s Address</th>
<th>R</th>
<th>Expected Sequence Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</td>
<td>R</td>
<td>0 1 2 3 4 5 6 7 8 9 0 1</td>
</tr>
</tbody>
</table>

Neighbor’s Address

The address of the neighbor we are acknowledging receipt of packet(s).

R (The request for acknowledgment bit - BAR)

This bit is set when the router is requesting an acknowledgment from the neighbor with address specified in the Acknowledgment List Entry.

Expected Sequence Number

Corresponds to the sequence number of the next expected packet from the neighbor. It acknowledges the receipt of all the packets with smaller sequence number.

LSU List Entry
H (Head ID bit - HIB)

This bit is set to 1 if the field "ID of the Head of the Link" is present in the LSU, it is set to 0 otherwise.

Head Prefix

Number of 1’s in the network mask associated with the "IP address of the Head of the Link".

T (Tail ID bit - TIB)

This bit is set to 1 if the field "ID of the Tail of the Link" is present in the LSU, it is set to 0 otherwise.

Tail Prefix

Number of 1’s in the network mask associated with the "IP address of the Tail of the Link".
Link Type

- LSU_BROADCAST (0x01): Broadcast
- LSU_P2P (0x02): Point-to-Point with subnet
- LSU_P2P_UN (0x03): Point-to-Point unnumbered
- LSU_P2P_BR (0x04): Point-to-Point broadcast
- LSU_ROUTER_TO_HOST (0x05): Link to a host
- LSU_LINK_EXTERNAL (0x06): Link to EXTERIOR ROUTING information

TOS Bit Vector

The iTH bit in the vector is set to 1 if the link is present in the source tree of the TOS associated with the iTH bit, otherwise the iTH bit is set to 0.

Time Stamp

Used to determine if the LSU carries up-to-date information. The time stamp corresponds to the number of seconds elapsed from January 1st, 1990, to the moment the head of the link generates the LSU reporting changes in the parameters of the link. A router maintains a clock that does not reset when the router stops operating. A time stamp based on 32 bits requires resetting after 136 years of operation.

IP address of the Head of the Link

Corresponds to the IP address of the network interface through which the head of the link has a bidirectional link with the router at the tail of the link.

IP address of the Tail of the Link

Corresponds to the IP address of the network interface through which the tail of the link has a bidirectional link with the router at the head of the link. This field can be a vector of IP addresses if the "Link Type" is LSU_ROUTER_TO_HOST (the number of instances is then determined by "Number of Links").

ID of the Head of the Link (OPTIONAL)

This field is required if the head of the link has multiple
network interfaces with different IP address. It specifies an ID that uniquely identifies the head of the link in the network, and should be chosen among the set of IP addresses assigned to the router.

ID of the Tail of the Link (OPTIONAL)

This field is required if the tail of the link has multiple network interfaces with different IP address. It specifies an ID that uniquely identifies the tail of the link in the network, and should be chosen among the set of IP addresses assigned to the router.

Parameters of the Link

```
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Num. of Metrics | Metric 1 Type | Metric 2 Type | Metric 3 Type |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Metric 1 Value |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Metric 2 Value |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Metric 3 Value | Padding |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

References


Implementation Status

We have implemented STAR running in gated. For convenience, in this implementation, STAR runs on top of UDP, just like RIP. We have demonstrated STAR several times at DARPA PI meetings running in testbed wireless internetworks consisting of wireless links and wired segments.

The gated code for STAR can be made available to the MANET community upon request.

We have verified the correctness of STAR, and the proof of its correctness is presented in a forthcoming journal publication.

The exact same code used in the gated implementation of STAR has been used in simulations comparing its performance against on-demand routing approaches. Part of these results appear in [11, 12].

An OPNET model of STAR is almost complete at the time of this writing, and we will start a public-domain simulation of STAR in the Parsec (GloMoSIM) package developed by UCLA.

The design and development work on STAR has been sponsored by the DARPA GloMo Program (Rob Ruth, Program Manager).

Chair’s Address

The Working Group can be contacted via its current chairs:
M. Scott Corson
Institute for Systems Research
University of Maryland
College Park, MD 20742
USA

Phone: +1 301 405-6630
Email: corson@isr.umd.edu

Joseph Macker
Information Technology Division
Naval Research Laboratory
Washington, DC 20375
USA

Phone: +1 202 767-2001
Email: macker@itd.nrl.navy.mil

Authors’ Addresses

Questions about this document can also be directed to: