# GSTAR Routing Details

GSTAR (*g*eneralized *st*orage-*a*ware *r*outing) is an intra-domain routing protocol built to overcome the challenges associated with mobile, wireless devices. It is suitable to run in a wireless and/or wired environment. At a high level, a node running GSTAR maintains time-sensitive information about links within its currently connected component (e.g., all nodes to which an instantaneous end-to-end path exists from the node in question) and time-insensitive information about general connection patterns between all nodes in the network. It attempts to use the time-sensitive information when possible, and fall back on the connection patterns when needed.

### Overview: Key Features

- The core of the protocol is link state – based on CNF / STAR routing
- It can respond to link quality fluctuations and congestion in the network by proactively storing message in the network downstream
- It has built in congestion control and congestion avoidance mechanisms by continuously monitoring available storage
- It is disruption-tolerant and can handle long and/or frequent disconnections

### Overview: Control Messages and Processing

GSTAR is fundamentally a link state protocol, with added DTN capabilities. There are three types of control messages: (1) link probes, (2) flooded link state advertisements (F-LSA), and (3) epidemically disseminated link state advertisements (D-LSA). Link probes allow nodes to obtain both time-sensitive ETT values for adjacent links as well as obtain a rough idea of the connectivity pattern of itself. F-LSAs allow nodes within the same partition as an advertiser to obtain short term ETT, long term ETT, and storage availability information about the advertiser and its adjacent links. D-LSAs allow all nodes, even those outside of an advertiser's partition, to obtain general connectivity information about the advertiser.

All nodes periodically probe for neighbors, making a note of which neighbors are currently available and what the ETT (directly computed) for the links are. Over time, they average the ETT values for a single link and compute a "long term ETT" value. All nodes in the partition periodically learn about the "current short term ETT", "current long term ETT", and "available storage" via periodically flooded F-LSA messages. They also learn about general connectivity patterns for the entire network via D-LSA messages. Therefore, two graphs are created: (1) the

intra-partition graph, where vertices are nodes within the partition and edges have both a short and long term ETT values associated with them, and (2) the inter-partition graph where vertices are nodes in the network and edge weights are a metric indicating the frequency or likelihood of two nodes being in contact.

## Overview: Data Processing

When a PDU arrives or is sourced, the router first checks the intra-partition graph to see if the destination ID is a valid vertex. If it is, it will solely use that graph to route the data. In this case, it will obtain the shortest path *with enough available space* using a combination of the short term link ETTs and long term ETTs. After obtaining a valid path, the router will then have to make a decision to forward to the next hop on that path, or store the data for later. This is done using a three-dimensional metric including: (1) short term ETT over the path, (2) long term ETT over the path, (3) exponentially weighted view of storage availability over the path. Proactive congestion control is built into the metric.

If the destination ID is not found in the intra-partition graph, then the DTN graph is consulted. The top MAX_DTN_PATHS non-overlapping shortest paths are computed. The goal is for a replica to make progress along each of these paths.

## Details: Control Messages and Processing

### Local Databases

Each node holds and periodically updates the following information:

1. List of current one-hop neighbors and the short-term and long-term link delays for each (referred to as the *1-hop neighbors*). It also holds the number of consecutive missed probe acknowledgements

*Single entry in database:*

| Neighbor | Short-term ETT | Long-term ETT | Last update time |
|---|---|---|---|

2. List of neighbors that occasionally respond to probes and a bitmap representing the amount of time the link is on and off (referred to as the *DTN neighbors*).

*Single entry in database:*

| Neighbor | 1 | 1 | 0 | 0 | 0 | 1 | 1 | … |
|---|---|---|---|---|---|---|---|---|

3. Intra-partition database/graph built from flooded link state advertisements (F-LSAs)

*Single entry in database:*

| Node Interface Address (IA) received F-LSA from |
|---|

| Sequence Number | | |
|---|---|---|
| Last Update Time | | |
| Current Disk Space Remaining | | |
| Neighbor 1 | Short-term ETT | Long-term ETT |
| Neighbor 2 | Short-term ETT | Long-term ETT |
| … | … | … |

4. DTN database/graph built from epidemically disseminated link state advertisements (D-LSAs)

*Single entry in database:*

| Node IA received D-LSA from | |
|---|---|
| Sequence Number | |
| Average Disk Space Available | |
| Neighbor 1 | Link Availability |
| Neighbor 2 | Link Availability |
| … | … |

5. Intra-partition Forwarding Table, built from running graph algorithms over the Intra-partition graph.

| Dest. 1 | Max Size 1 | Next Hop | SETT for path | LETT for path | Path storage metric |
|---|---|---|---|---|---|
| Dest. 1 | Max Size 2 | Next Hop | SETT for path | LETT for path | Path storage metric |
| Dest. 1 | … | … | … | … | … |
| Dest. 2 | Max Size 1 | Next Hop | SETT for path | LETT for path | Path storage metric |
| Dest. 2 | … | … | … | … | … |
| … | … | … | … | … | … |

How these databases are updated and utilized is described in the following sections.

**Link Probe Message (LPM)**

Every node will *broadcast* a LPM every LP_PERIOD seconds, meant to be seen by its one-hop neighbors. This probe contains a timestamp that will be to obtain the link delay, as well as the source IA.

| **LPM** | |
|---|---|
| xx bits | Source IA |
| xx bits | Timestamp (ms) |

Each node receiving this broadcast must immediately respond with an LPM ACK. This is the only action taken by the receiving node; no local databases are updated. The LPM ACK contains the

original source IA as well as the responding node IA. Furthermore, it contains the same timestamp as the LPM it is acknowledging.

| LPM ACK | |
|---|---|
| xx bits | Orig. Source IA |
| xx bits | Responder IA |
| xx bits | Timestamp (ms) |

Upon receiving an LPM ACK, the receiving node checks to see if it is the *Orig. Source IA* in the message. If not, it drops the message. If so, it computes the link delay as (current time – timestamp) / 2. It then updates its *1-hop neighbor* database with that delay, and inserts a *1* into the corresponding bitmap in the *DTN neighbor* database. If a sent probe was not acknowledged by a neighbor already in the *1-hop neighbor* database, then the *number of missed probes* field for that node is incremented. If that number exceeds MAX_MISSED_PROBES, then the entry is deleted from the database. If a sent probe was not acknowledged by a neighbor already in the *DTN neighbor* database, then a *0* is inserted into the corresponding bitmap.

**Flooded Link State Advertisement (F-LSA)**

Every node will *flood* an F-LSA every F_LSA_PERIOD seconds. This will contain time-sensitive information about the node's current 1-hop neighborhood and allows all nodes in the connected component to obtain fine-grained topology information. An F-LSA message size can vary, depending on how many 1-hop neighbors the source node has. Sequence numbers are added to determine age, and hence the Source IA and Sequence Number together can uniquely identify an F-LSA.

| F-LSA | |
|---|---|
| xx bits | Source IA |
| xx bits | Sequence Number |
| xx bits | Disk Space Remaining |
| xx bits | Neighbor 1 IA |
| xx bits | Short-term ETT for 1 |
| xx bits | Long-term ETT for 1 |
| xx bits | Neighbor 2 IA |
| xx bits | Short-term ETT for 2 |
| xx bits | Long-term ETT for 2 |
| .. | … |

When a node receives an F-LSA, it first checks to see if an entry exists in its *Intra-Partition Graph* containing the same Source IA and a Sequence Number equal to or greater than the

Sequence Number in the F-LSA. If so, the message is silently dropped. If not, it performs two tasks: (1) updates its local database and (2) rebroadcasts the F-LSA.

First, the *Intra-Partition Graph* is edited for the Source IA entry (or created if the entry does not exist). All ETT fields are appropriately copied. Next, the Sequence Number in the F-LSA is copied into the Sequence Number field in the entry, and the Disk Space field is copied over. Finally, the Last Update Time is set to the current local time. After the database is updated, the node must then rebroadcast the received F-LSA.

**The Path Storage Metric (Optional - Currently Being Explored)**

The path storage metric is a way of proactively controlling storage-based congestion. The goal is to not immediately fill up the buffers close to popular destinations, but rather start to store messages before they cause congestion. By keeping congestion-point buffers from getting too full, some storage will be more readily available for messages that have very high short-term ETT / long term ETT ratios, which should improve overall metrics.

The goal is to compute the available storage over a path, with higher weights on closer nodes. If the metric is low compared to the size of the PDU being sent, then the router should consider storing the data when it may have otherwise forwarded it along. If the number of hops was infinite, and the storage at the *i*th hop was *S_i*, then the path storage metric would be:

$$PSM = \frac{1}{2} \cdot S_1 + \frac{1}{4} \cdot S_2 + \frac{1}{8} \cdot S_3 + \ldots$$

since the weights add to 1. Consider a finite length path of *h* hops. Then the PSM would be:

$$PSM = \frac{2^{h-1}}{2^h - 1} \cdot S_1 + \frac{2^{h-2}}{2^h - 1} \cdot S_2 + \frac{2^{h-3}}{2^h - 1} \cdot S_3 + \ldots + \frac{2^{h-h}}{2^h - 1} \cdot S_h$$

**Epidemically Disseminated Link State Advertisement (D-LSA)**

Every node will *epidemically disseminate* a D-LSA when necessary. This need not be periodic, and instead should be sourced when a significant change occurs in the *DTN neighbors*. D-LSA messages allow nodes to obtain course-grain topological information. Sequence numbers are added to determine age, and hence the Source IA and Sequence Number together can uniquely identify a D-LSA. Average Disk Space Available is simply the expected disk space availability the node will have at some point in the future.

| D-LSA | |
|---|---|
| xx bits | Source IA |
| xx bits | Sequence Number |
| xx bits | Average Disk Space Available |
| xx bits | DTN Neighbor 1 IA |
| xx bits | Link Availability for 1 |
| xx bits | Neighbor 2 IA |
| xx bits | Link Availability for 2 |
| … | … |

For each entry in the *DTN neighbors* table, a bitmap exists corresponding to the time slots the link was available. It is currently an active research question about how to best characterize that bitmap as a single metric. For now, the following "average availability" metric can be used:

$$AA = \#1's / (\#1's + \#0s)$$

Note that if only 0's have been seen for MAX_DTN_LINK_TIME, then the bitmap can be dropped and the DTN link is discarded.

D-LSAs are disseminated using epidemic dissemination, and hence a copy the most recent unique (Source IA, Sequence Number) D-LSAs for every node are stored at every node. These messages are synched using standard epidemic techniques. To process a newly obtained D-LSA, the information from the message is simply copied into the *DTN graph*.

**Intra-Partition Graph Aging**

The *Intra-Partition Graph* must periodically be aged, since the information contained in it is time-sensitive. If the (current time - Last Update Time) is ever greater than MAX_IPGRAPH_LINK_TIME, the corresponding entry is deleted from the *Intra-Partition Graph*.

**Path Selection - Updating the Intra-Partition Forwarding Table**

The Intra-Partition forwarding table must be updated every time there is a change in the Intra-Partition graph. There are multiple ways to select a path, each of which must be explored in more detail. Some options are:

1) Use SETT as weights and run Dijkstra's algorithm
2) Use a combination of SETT and LETT (which more weight towards LETT if the link is further away from the source) and run Dijkstra's algorithm
3) Use the *delay-based selection algorithm* found in the supplementary document "Path Selection in Storage-Aware Ad-hoc Routing"

Each destination will have multiple "max size" fields associated with it, indicating the size of the PDU used as a reference for the path computation. Dijkstra's algorithm can then be used to compute the shortest path for a particular <destination, max size> pair, after eliminating all nodes from the graph that have available storage less than "max size". The result is used to fill in the "next hop" field. Finally, the short term ETT, long term ETT, and path storage metric (described in the following section) are filled in for that entry.

## Details: Data Processing

### PDU Header Format

PDUs have the following header format:

| Source GUID | Source IA | Destination GUID | Destination IA |
|---|---|---|---|
| PDU Size | Service ID | | |

Nodes have the freedom to re-bind the GUID to a IA by using the getGUID() interface from the GNRS. If there is no IA attached to the PDU, then the node must either perform this operation or transmit the PDU as is to a gateway or default router.

### Buffering Incoming PDUs

There are two primary buffers that can be used by the router: (1) transit buffer and (2) hold store. The transit buffer is a single priority queue, where PDUs coming into the router are placed in the appropriate slot in the transit buffer. There are three primary priorities, as follows:

1. PDU's service ID indicates real-time data, and the PDU is small in size
2. PDUs whose destination is found in the *Intra-Partition Graph*
    a. Further ordering is done based on (short-term ETT / long-term ETT) value for the PDU
3. PDUs whose destination is found only in the *DTN Graph*

### Transmitting a PDU

When a PDU traverses through the transit buffer and is ready to be sent, GSTAR notes two properties of the PDU: (1) destination ID and (2) PDU size.

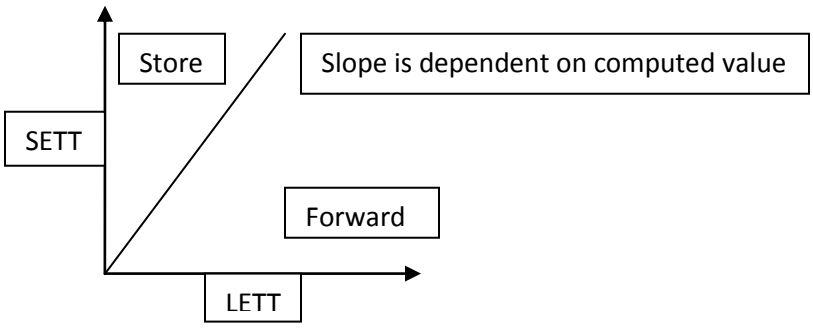
*If the destination is found in the intra-partition graph*:
The forwarding table entry corresponding to the PDU's destination ID and size is consulted to obtain the next hop, SETT, and LETT. If the SETT is high compared to the LETT, then the link is abnormally good, and hence immediate forwarding is desired. If the LETT is high compared to

the SETT, then the link is abnormally bad, and hence the PDU should be placed in the hold store instead of being transmitted. It is important to note here that GSTAR assumes that there is no transport layer congestion control, such as TCP. If there were, then the decision to store would be wrongfully viewed as congestion and data rates unnecessarily throttled.

*If the path storage metric is being used:*
The decision to store or forward can be pictorially illustrated by the following graph, where the slope is defined by:

$$Slope = \frac{PathStorageMetric}{2 \cdot PDUSize}$$

In this case, if there is a lot of storage available, the slope will be relatively high, encouraging forwarding. If the amount of storage is equal to the PDU size, and hence very constrained, the slope is 0.5, which heavily encourages storing.




*If the path storage metric is not being used:*
The slope is simply set to 1.1. Therefore, the following algorithm indicates whether to store or forward:

*If SETT > 1.1 x LETT then store*
*Else forward*

*If the destination is found on the DTN graph:*
The weights of the DTN graph give information about the connectivity patterns in the network. The goal is to make progress along the DTN graph until the point when the destination node falls within the local partition. The following approach can be taken to make progress:

1. The top MAX_DTN_PATHS are computed using Djikstra's algorithm in the following manner.

2. The node transmits a replica to the next hop for each path according to the DTN graph. Note that the next hop may not be immediately available and the router may have to temporarily store the message.

*If the destination is not found on either graph,* then node then has the option to place the PDU in the hold store or drop it.

**Possible Extensions – Partial Source Routing in DTN Graph for Loop Prevention**

In order to better prevent routing loops due to varying views of the DTN connectivity graph, one possible extension is the following. When a node receives a message and the destination of that message is not found on the intra-partition graph, it normally consults the DTN graph and transmits to one or more neighbors of that graph. It may be possible to instead, for each replication path, determine the furthest node on that path such that it is still within its intra-partition graph and has enough available storage. Replicas then can be transmitted to these nodes using the intra-partition graph technique, and then stored there until the appropriate next hop is met.

Therefore, the DTN steps will be changed to the following. The header will also be changed as follows:

| | | | |
|---|---|---|---|
| Source GUID | Source IA | Destination GUID | Destination IA |
| PDU Size | DTN Flag | Intermediate IA | DTN Next IA |
| Service ID | | | |

1. The top MAX_DTN_PATHS are computed using Djikstra's algorithm in the following manner. The shortest path is computed and the node furthest away on that path that is still within the partition and has adequate space is marked. Furthermore, the node 1-hop further than the marked node is noted as a "DTN next hop" node. This node is then removed from the graph and the source path computation is redone, this time obtaining a new "furthest" node, which is marked. This process is repeated until MAX_DTN_PATHS nodes are marked.
2. A replica of the PDU is sent to each of the marked nodes, with the replica setting the *Intermediate IA* to the IA of the appropriate marked node. Furthermore, the noted "DTN next hop" node is set in the *DTN next IA*, and the *DTN Flag* is set. These replicas can be sent using the *Intra-Partition Graph*. After they are transmitted, the node can delete its local copy of the PDU.
3. When the marked nodes each receive their corresponding replicas, they must do one of three actions: transmit the replica to the node set in the *DTN next IA* field, store the replica until the *DTN next IA* becomes available, or drop the replica. This is to force the partition to be bridged and hence help mitigate routing loops.

4. When the *DTN next IA* receives the message, it can clear all intermediate fields and treat the PDU as normal. Furthermore, it can flood a local acknowledgement to the partition it was in, causing the other replicas to be deleted. This is similar in nature to BBN's *Endemic* work.