Intra-domain Generalized Storage-Aware Routing (GSTAR)

Samuel Nelson

Working Document

Rutgers University, WINLAB

1. Introduction

The *MobilityFirst Future Internet Architecture* project is geared around the principle that mobile devices and their associated applications must be treated as first-class Internet citizens. Traditionally, the challenges associated with mobility and wireless communication were partitioned from the core Internet and handled as a "last hop" problem. However, with the prevalence of hand held and other wireless devices, many acting as transit points for routing purposes, these challenges will have to be address throughout the entire access network and, to some degree, in the core itself. In fact, we envision a future Internet where networks are not strictly characterized (e.g., core vs. access, wired vs. wireless, ad-hoc vs. managed) but are instead fluid and highly heterogeneous.

Since most of the interesting mobility challenges will occur relatively close to end users, it is critical that *MobilityFirst* presents a flexible, robust, and unified means for exchanging data in a local area through many different types of environments. To this end, this paper presents *GSTAR*, a generalized storage-aware intra-domain routing protocol capable of high performance in a variety of mobility-driven environments, including wired, wireless mesh, wireless ad-hoc, and DTN. GSTAR is fundamentally a link-state protocol that incorporates readily available router storage to seamlessly bridge different environmental challenges.

At a high level, 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. In this way, it can be thought of as a MANET+DTN protocol that is easily extended to more stable, perhaps wired, environments. GSTAR routes on flat identifiers, namely *GUIDs*, and proactively transmit topology information about GUID adjacencies.

At a high level, GSTAR can be summarized as follows:

- The core of the protocol is link state (based on CNF / STAR routing, with DTN additions), utilizes readily-available routing storage, and relies on hop-by-hop data transfer.
- It can respond to link quality fluctuations and congestion in the network by proactively storing message in the network downstream
- It is disruption-tolerant and can handle long and/or frequent disconnections

The rest of this report is presented as follows. First, the core principles and concepts of GSTAR are described. This is followed by a detailed description of the protocol, including path selection, storage-aware forwarding decisions, and DTN fallback mechanisms. An ns3 and CLICK implementation is then described and preliminary results analyzed. Next, a brief discussion on open problems and possible extensions is presented. Finally, future work is presented and the report is concluded.

2. Motivation

GSTAR is motivated primarily by supporting highly heterogeneous networks in a wide range of mobility-driven environments. A single network may include desktop computers, laptops, smartphones, and vehicles, which naturally brings a multitude of mobility-driven environmental challenges, such as those found in ad-hoc networks and DTNs (see Figure 1)

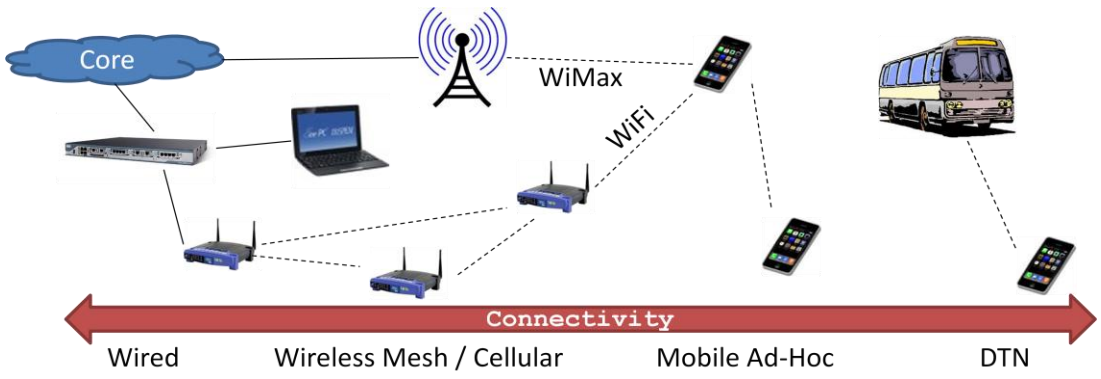


Figure 1: A heterogenous network and the mobility-driven challenges associated with it

To better understand the goals and guiding principles of GSTAR, it is important to visit the overall *MobilityFirst* routing challenges and associated principles. The major challenges found in a mobility-centric Internet are as follows.

1. *Host Mobility* – Mobile hosts must be handled as core to the network design and not rely on added infrastructure and possibly suboptimal routing techniques such as MobileIP.
2. *Varying levels of link quality* – Transport protocols, such as TCP, perform poorly in mobile environments where link qualities fluctuate. Instead of relying on end-to-end acknowledgements as congestion feedback, bulk data transfer in a hop-by-hop fashion and the ability to temporarily store data are useful.
3. *Varying levels of connectivity* – Due to the heterogeneous, mobile nature of future networks, complete disconnections and partitions should be expected. Since traditional

ad-hoc protocol fail in these environments, techniques from the DTN community must be used to help facilitate data transfer.

4. *Multi-homing* – Most devices have or will have multiple radios, and hence the potential for multiple network attachment points. The future Internet should therefore proactively utilize and take advantage of this.
5. *Context*-aware routing paradigms – Group-based communication, such as contacting any or all nodes that meet some characteristic is becoming increasingly important in many different networks such as disaster recovery networks, sensor networks, and mobile social networks.

Each of these challenges, to some degree, carries over to and must be address locally. Furthermore, these five challenges bring to light four main design principles of *MobilityFirst* routing.

1. Separation of naming and addressing
2. Late binding
3. In-network storage utilization
4. Conditional routing behavior

Since GUIDs are, by definition, unique, GSTAR uses these as a naming primitive for flat routing within a GSTAR network. Therefore, there is no need for name resolution when routing to and from the same GSTAR network, as all necessary information is propagated via the routing control plane.

3. GSTAR Protocol

MobilityFirst uses a two pronged approach for intra-domain routing that is capable of quickly responding to link quality changes for nearby nodes as well as remaining robust in the face of disconnectivity and partitioning. At a high level, individual routers maintain two types of topology information, one useful for responding to fine-grained changes to links and nodes *within* the router's current partition, and one useful for responding to course-grain changes to connection probability for all nodes in the network.

3.1 Protocol Overview

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.

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.

If the destination ID is not found in the intra-partition graph, then the DTN graph is consulted, and the next hop is determined based on progress that can be made along that graph.

3.2 Protocol Details

In this subsection, we describe the details for both the control and data plane.

3.2.1 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:

Source GUID		
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:

Source GUID	
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	Store/Forward Flag
Dest. 1	Max Size 2	Next Hop	SETT for path	LETT for path	Store/Forward Flag
Dest. 1	…	…	…	…	…
Dest. 2	Max Size 1	Next Hop	SETT for path	LETT for path	Store/Forward Flag
Dest. 2	…	…	…	…	…
…	…	…	…	…	…

3.2.2 Control Plane – Proactive Message Dissemination

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 GUID.

LPM
Source GUID 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 GIUD as well as the responding node GUID. Furthermore, it contains the same timestamp as the LPM it is acknowledging.

LPM ACK
Orig. Source GUID Responder GUID Timestamp (ms)

Upon receiving an LPM ACK, the receiving node checks to see if it is the *Orig. Source GUID* 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.

Note that if the router in question is serving end clients, then the router must *automatically* consider them neighbors and compute and store their link delay and DTN average availability without the use of link probes, since the end clients do not run GSTAR. These end clients are then added to zero or more of the F-LSA and D-LSA messages that the router sends.

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. Additionally, a flag is set if the node desires to be a gateway router. If this flag is set, then the node is promising that it has sufficient connectivity to the rest of the Internet to ensure timely delivery of external packets. 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 GUID and Sequence Number together can uniquely identify an F-LSA.

F-LSA
Source GUID
Sequence Number
Gateway Flag
Disk Space Remaining
Neighbor 1 GUID
Short-term ETT for 1
Long-term ETT for 1
Neighbor 2 GUID
Short-term ETT for 2
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 GUID 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 GUID 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. This 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*.

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 GUID 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
Source GUID
Sequence Number
Gateway Flag
Average Disk Space Available

DTN Neighbor 1 GUID Link Availability for 1 Neighbor 2 GUID 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:

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

Note that if only 0's have been seen for an extended period of 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 GUID, 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*.

3.2.3 Control Plane – Intra-Partition Path Selection and Forwarding Table Computation

This subsection first describes a generalized storage-aware metric for ad-hoc networks that captures the end-to-end latency associated with transferring a large block through the network. This metric is then instantiated for use as a path selection metric in GSTAR. Note that this is for the intra-partition graph only.

Generalized Storage-Aware Metric

The following are assumptions made:

- Blocks are transferred in a hop-by-hop fashion
- Routers have storage that they allow other nodes in the network to utilize
- The network layer has access to some type of link quality information for the links adjacent to it
- The network is generally connected; e.g., most of the time, instantaneous end-to-end paths exist

As a block traverses a path, each node along the path will perform the following steps, each of which increases the total end-to-end latency of the block:

1. Store the block until ready to send
2. Attempt to gain access to the channel
3. Transmit the block to the next node along the path

Assume nodes along an arbitrary path are labeled *1,2,…,k* where *1* is the source and *k* is the destination. Let $BACK_i$ be the expected amount time a block will spend in storage at node *i* due to backpressure from the hop-by-hop transport. Let $HOLD_i$ be the expected amount time a block will spend in storage at node I due to a routing decision to store not related to forced backpressure. Let P_i be the probability that a routing decision to store (not due to backpressure) is made by node *i*. Let $CHAN_i$ be the expected amount of time a block will wait while node *i* gains access to the channel to send the block. Let T_i be the expected amount of time a block will be in transit from node *i* to node *i + 1*.

The total end-to-end latency for a block over a path is:

$$delay_{1,2,\ldots,k} = \sum_{i=1}^{k} \left(P_i \cdot HOLD_i + BACK_i + CHAN_i + T_i \right)$$

Storage time due to backpressure at node *i* will directly depend on the amount of data at node *i* with higher priority that the block in question that is also going to node *i + 1*, in addition to how long it takes for space to open at node *i + 1*. Making the simplifying assumption that blocks are served on a first-come first-serve basis:

$$BACK_i = \frac{D_i}{tx_{i,i+1}} + TWS_{i+1}(D_i)$$

where D_i is the amount of data being held at node *i* due to backpressure, $tx_{i,i+1}$ is the transmission rate between node *i* and node *i+1*, and $TWS_{i+1}(D_i)$ is the time node *i* waits on node *i+1* to clear out D_i space.

The transmission time, T_i, is:

$$T_i = \frac{blockSize}{tx_{i,i+1}}$$

Practical Instantiation for GSTAR

One approach taken by CNF is to hold data close to the source until the SETT path metric is reasonable in relation to the LETT path metric. The following simplifying assumptions to the model are for a particular path *p*:

- Blocks are held at the source, and after they are released they will not be held (except for queuing due to backpressure) at any intermediate node. Therefore, $P_i = 0$ for $2 <= i <= k$.
- $TWS_i = 0$ for $1 <= i <= k$, which is a result of holding the block at the source until the path becomes of normal quality.
- Channel access time is negligible, since block sizes are quite large.
- All storage buffers in the network are large enough to handle any valid block size when empty.

The current values of D_i and $tx_{i,i+1}$ can be estimated by node *i* and transmitted to all other nodes via LSA flooding. Furthermore, the source node can estimate P1 in the following way. $P_1 = 0$ if the current SETT path metric is normal or low relative to the LETT path metric. $P_1 = 1$ if the current SETT path metric is high relative to the LETT path metric. Furthermore, HOLD1 is the average amount of time for that path that the SETT is relatively high. This can be estimated by proactively keeping track of the following: given the SETT is high, on average how long does it take to become normal or low?

With this information, a source node has everything needed to compute the delay metric for any given path. This metric can be used to select the best path from numerous ones. This particular instantiation of the metric is therefore:

$$delay_{1,2,\ldots,k} = P_1 \cdot HOLD_1 + \sum_{i=1}^{k} \left(BACK_i + T_i \right)$$

where D_i and $tx_{i,i+1}$ are estimated from the received LSA message and $TWS_i = 0$ for all *i*.

At this point, a forwarding table can be generated for all GUIDs within the router's intra-partition graph.

3.2.4 Control Plane – DTN Path Selection

Path selection for the DTN graph is based on a simple Dijkstra computation over the received DLSA messages. Specifically, the received DLSA messages allow a graph to be formed using GUIDs are vertices and a function of the *average availability* metric as weights.

If the *average availability* between two GUIDs is *AA*, then the corresponding weight *w* on the graph feed into Dijkstra's Algorithm is:

$$w(AA) = 1 - AA + 0.01$$

Note that the small constant (0.01) is added to all weights. This allows Dijkstra's Algorithm to favor paths with a shorter number of hops if the average availabilities in question are all *1*.

3.2.5 Data Plane

PDUs have the following header format:

Source GUID	Destination GUID	PDU Size	Service ID

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:

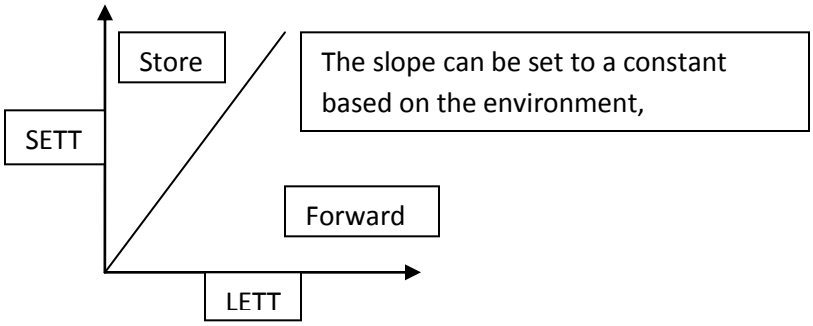
1. PDU's service ID indicates real-time data, and the PDU is small in size
2. PDUs whose destination is found in the *Intra-Partition Graph*
    a. Further ordering is done based on (short-term ETT / long-term ETT) value for the PDU
3. PDUs whose destination is found only in the *DTN Graph*

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.

The decision to store or forward can be pictorially illustrated by the following graph, where the slope is simple set to 1.1 as an example.



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. If replication is being used, then a set of disjoint paths are found via Djikstra's Algorithm.
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 the node must make a query to the GNRS to figure out which network the GUID is part of. If the result of this query is that the GUID is *not* in the same network as the router in question, then the router must send the data unit to one of the GUIDs whose corresponding F-LSA (preferably) or D-LSA "gateway flag" was set. If the result of the query is that the GUID is in the same network as the router in question, then the router must simply store the data unit and revisit it in the future.

4. Simulation Details and Results

GSTAR, as well as hop-by-hop transport, has been implemented in the *ns3* network simulator. This work is currently ongoing.

5. Implementation Details and Experimental Results

GSTAR, as well as low-level hop-by-hop transport, has been fully implemented in the CLICK software router and is currently running over the Orbit testbed. For implementation details and experimental results, please see the corresponding "MobilityFirst Router Implementation" tech report.

6. Extensions and Discussion

There are many possible extensions that we plan to explore both in simulation and experimentation. Two important ones are (1) a path storage metric for proactively controlling storage-based congestion, and (2) partial source routing in the DTN graph for loop mitigation.

6.1 Path Storage Metric

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:

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

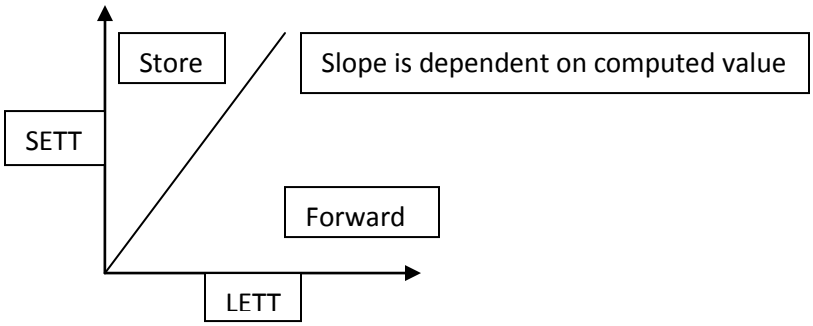
since the weights add to 1. Consider a finite length path of *h* hops. Then the PSM would be:

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

In order to use the path storage metric, it must be integrated in the decision to either store or forward. 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.



6.2 DTN Partial Source Routing

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	Destination GUID	PDU Size	Service ID
DTN Flag	Intermediate GUID	DTN Next GUID	

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 GUID* to the GUID of the appropriate marked node. Furthermore, the noted “DTN next hop” node is set in the *DTN next GUID*, 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 GUID* field, store the replica until the *DTN next GUID* 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 GUID* 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.