RAPID: An External Router for DTN2

http://prisms.cs.umass.edu/rapid/

Brian Lynn
<first initial last name at cs.umass.edu>
University of Massachusetts, Amherst
November 20, 2008

Notice

The development of RAPID was supported by Defense Advanced Research Projects Agency (DARPA) grant W15P7T-05-C-P213. Distribution Statement A: Approved for public release; distribution unlimited.

Introduction

The Resource Allocation Protocol for Intentional DTN, or RAPID, is a routing algorithm for disruption-tolerant networks [1]. We have implemented RAPID for the Delay-Tolerant Networking Research Group's DTN reference implementation (DTN2) [2].

The RAPID algorithm was a result of research performed on DieselNet [3], a mobile network test bed developed by the University of Massachusetts, Amherst. A defining characteristic of a mobile network such as DieselNet is that on any given day nodes may have contact with only a subset of the other nodes, and that contact may be infrequent and for very brief periods of time. The RAPID algorithm was designed to effectively route data in a mobile network. A feature of RAPID is that it functions by optimizing for a specific routing metric. In our DTN2 implementation of RAPID, the metric is to minimize the delivery time for all DTN2 bundles.

RAPID executes as an external DTN2 router, using DTN2's XML-based External Router Interface. RAPID is implemented in Java.

Files and Directories

RAPID requires the DTN2 external router schema file. By default, DTN2 installs the file as /etc/router.xsd.

RAPID exchanges a meta data bundle when it comes into contact with an instance of the RAPID router running on another node. In order for RAPID to be able to read a received meta data bundle's payload, RAPID must have read access to the DTN2 bundle directory. The directory is often named /var/dtn/bundles.

RAPID periodically writes to a persistent file which bundles are known to have been delivered. By default, RAPID writes to a file named /var/rapid_router/acks. Therefore RAPID will need read and write access to the /var/rapid_router directory. The location of the file may be overridden in the RAPID configuration file. It is also possible to disable this functionality.

Building and Installing the RAPID Router

You should have Java 5.0 (a.k.a. 1.5.0) or later installed on your system. To build and install RAPID for DTN2:

```
$ Untar the files: tar zxf rapid_router-0.9.4.tar.gz (or whatever your tar is named)
$ cd rapid_router
$ ./install.sh
```

The install script will prompt you to build RAPID if necessary, and it will also ask you questions about what directories to use. The script should work regardless of whether you have RAPID source or a pre-built distribution.

You can use make to build RAPID by simply typing:

```
$ make
```

The syntax for installing RAPID is:

```
Make install [ INSTALLDIR=<directory> ] [ACKSDIR=<directory> ]
```

To install RAPID in its default location, /usr/local/rapid_router, using /var/rapid_router for check pointing delivered bundles:

```
$ make install
```

To install RAPID in a different directory

```
$ make install INSTALLDIR=<directory>
```

Make will create the ACKSDIR, though there are no files to copy to the directory. You will need to define the file's path in the RAPID configuration file if using a checkpoint file other than /var/rapid_router/acks.

Running RAPID

You should have DTN2 installed on your system. RAPID was developed and tested with DTN 2.5, but appears to work fine with DTN 2.6. Due to limitations in the DTN2 external router interface, RAPID should be started before starting the DTN2 daemon, dtnd. Once RAPID is running, it interacts solely with the DTN2 daemon.

When RAPID is installed, a script that runs Java with the RAPID JAR is also installed. The syntax of the RAPID command is:

```
RAPID [-c|--config file] [-x|--xml file] [-l|--log file] [-L|--level N]
RAPID -v|--version
RAPID -?|-h|--help
```

where

-c, --config Specifies a file containing RAPID configuration values. The default is to

	have no configuration file and to use all default values.
-x, --xml	Specifies the XML schema definition file that defines the XML messages exchanged between RAPID and the DTN2 daemon. The default is /etc/router.xsd. The schema file is provided by DTN2.
-l, --log	Defines a logging configuration file. This configuration file is specific to the logging class being used. By default, RAPID does not require a logging configuration file. If you are using the Apache log4j logger then its configuration file can be specified here.
-L, --level	Specifies logging verbosity, where 0 is all messages are to be logged and 6 is no messages are to be logged.
-v, --version	Tells RAPID to print its version number and then exit.
-?,-h,--help	Instructs RAPID to output some brief help text and then exit.

The RAPID configuration file is discussed in more detail later in this document.

RAPID Logging

RAPID comes with two implementations of its logging class. By default, it uses the Console_Logging class which simply logs messages to stdout. RAPID also supports the Apache log4j logger, which provides greater flexibility over where output is logged. To use the Apache log4j logger:

- Download the log4j JAR file from http://logging.apache.org/log4j/1.2/download.html. Install it on your system, if it is not already installed.
- Edit src/Makefile to build Log4j_Logging.
- Edit the RAPID script so the Java -classpath references the JAR file.
- Rebuild and re-install RAPID.
- Add a line to the RAPID configuration file: loggingClass=Log4j_Logging
- Specify the RAPID configuration file when running RAPID.
- Optionally, use the --log parameter when running RAPID to reference your log4j configuration file.

The logging level can be defined by the --level parameter of the RAPID command. Specifying a level enables output of logging messages at or above that level. RAPID uses the logging levels defined by Apache log4j.

0 - Tracing messages.
1 - Debugging messages.
2 - Informational messages.
3 - Warnings.
4 - Errors.
5 - Fatal error messages.
6 - This level results in no messages being logged.

If using the Console_Logging class the default level is 4, warnings.

DTN2 Configuration

Important: RAPID requires that the DTN2 configuration file -- normally /etc/dtn.conf -- include the following line:

```
param set early_deletion false
```

If this option is not set then DTN2 may delete bundles after they have been initially sent, preventing RAPID from replicating bundles.

You also need to make certain that DTN2 is configured to use an external router:

```
route set type external
```

RAPID also assumes that DTN2 is configured to open and close links as nodes come in and out of contact. This may include using a discovery protocol. The RAPID_Policy class supports the ability to add a simple link management class.

RAPID Routing and Implementation Overview

This section provides a very brief overview of RAPID routing and its implementation for DTN2. You should refer to [1] for more details on RAPID. The RAPID_Policy class implements RAPID and makes all of the routing decisions.

Remember that RAPID was created for mobile, disruptive networks. This is reflected in the characteristics of RAPID noted below.

- RAPID assumes that links between nodes are dynamic and that routes are transient. No effort is made to learn the network topology because it is fluid. Instead, RAPID attempts to calculate the likelihood some node will be in contact with another node.
- Since links are typically short-lived, RAPID assumes that bandwidth is in shorter supply than storage. The goal of RAPID is to prioritize bundles in order to effectively make use of bandwidth. When two nodes establish a connection, the assumption is that the link may not persist long enough to exchange all bundles. (Though RAPID makes use of on demand links, the assumption is that a DTN2 discovery service is employed.)
- RAPID does not generate explicit, per-bundle delivery acknowledgments to be propagated throughout the network. Instead, a list of bundles known to have been delivered is exchanged when a link is first established.

The DTN2 / RAPID architecture is very simple at the highest level. The DTN2 daemon, dtnd, sends multicast packets to be received by a local router process. These packets contain XML data and provide notification of events, such as the creation of a link or the receipt of a bundle. The router process, in this case RAPID, sends multicast XML messages to dtnd requesting that some action be taken, such as the transmission of a bundle. Note that the exchanged messages are not to be viewed as being conversational. For example, RAPID may request that dtnd transmit a bundle but it does not expect a response. If dtnd does transmit the bundle, it will send a multicast message about the transmission, but that message is a separate event and not a reply to the transmit request. The implication of this is that requests to dtnd do not have negative acknowledgments.

When DTN2 establishes a connection between two nodes, RAPID is notified and the RAPID peers exchange meta data. This information is used to prioritize the transmission of bundles in subsequent meetings between nodes. In the broadest sense, for each known node RAPID sorts bundles into two queues:

- Bundles that are to be delivered to that node.
- Bundles destined for other nodes but are to be replicated on the node for routing.

Bundles on the delivery queue are sorted by creation time. Bundles on the replica queue are sorted by a set of criteria that includes the creation time, the probability that the routing node will meet with the destination node, and the number of copies of the bundle that are known to exist in the network. For each node that RAPID is aware of, it maintains a list of bundles the node is known to posses. This is done to minimize bundles being redundantly sent to a node.

The exchanged meta data provides the information for prioritizing bundles on each replica queue. The meta data contains:

- Any known contact history between nodes. This includes the average time between contacts, the average duration of contacts, and the average number of bytes transferred per connection.
- A list of unexpired bundles possessed by the node.
- A list of unexpired bundles that are known to have been successfully delivered in the network (delivery acknowledgments).

The meta data a node receives is used in the generation of future meta data bundles. For example, the list of delivered bundles is the union of bundles the node has delivered and all other acknowledgements it has learned about by receiving meta data. If RAPID receives meta data that indicates that a bundle it possesses has already been delivered, it will delete that bundle.

RAPID uses several means to determine what bundles another node possesses. These include:

- Received meta data, where a node will list the bundles in its possession.
- Receipt of a bundle. RAPID, of course, can assume that a node sending a bundle has the bundle.
- If dtnd indicates that a bundle was successfully transmitted, then RAPID assumes the peer has a copy of the bundle.
- RAPID also will infer that a bundle's originating node is in possession of the bundle.

Similarly, there are several methods for concluding that a bundle was delivered (as opposed to replicated). Note that delivery indicates that the destination node has a copy of the bundle. It does not indicate that the endpoint has been notified of the bundle.

- If a node receives a bundle and that node is the bundle's target destination, obviously it notes that the bundle has been delivered.
- If dtnd indicates that a node has successfully transmitted a bundle to the destination node, the transmitting node assumes delivery.
- Received meta data lists bundles known to have been delivered.

Below are some notes on the deletion of bundles.

- By default, when dtnd successfully transmits a bundle it then deletes the bundle. This is not the desired behavior; otherwise RAPID would not be able to replicate bundles on multiple nodes. That is why it's important to set early_deletion to false in the DTN2 configuration. The proper behavior is for dtnd to retain a copy of the bundle after transmission.
- In the case where a bundle is destined for the local node, dtnd will automatically delete the bundle after it has been received by the endpoint. Dtnd will then send RAPID a bundle deletion notification.
- Dtnd will delete bundles when they expire and send notifications to RAPID.
- If RAPID learns of a bundle delivery acknowledgement via meta data and it is in possession of that bundle, it will ask dtnd to delete the bundle.
- If RAPID forwards a bundle to the destination node, RAPID will request that the bundle be deleted.

Needless to say, RAPID does not actually possess, delete or transmit bundles. This is all performed by the DTN2 daemon, dtnd. Instead, RAPID makes decisions and informs dtnd which bundles are to be transmitted or deleted.

RAPID Classes

This section can easily be skipped. Its purpose is to provide an overview of the main classes should somebody be interested in understanding the source code.

The RAPID router acts upon several fundamental constructs:

Bundles. A bundle is DTN2 terminology for an application-defined unit of data that is to be transmitted to another application, typically on another system. The RAPID router attempts to transmit entire bundles; it does not perform explicit fragmentation of bundles. All bundles have a sender-defined expiration time. A bundle is uniquely identified across all systems by a global DTN2 identifier known as a GBOF (global bundle or fragment) identifier, or simply GBOF. Internally, DTN may also refer to a bundle present on the local system through a system-specific handle known as the local ID.

Nodes. A node is the terminology we use for systems. Nodes and applications are defined within URI-style endpoint identifiers (EIDs). For example, given URI dtn://name.dtn/app we use dtn://name.dtn to identify the node. We currently do not support wildcarded nodes, though wildcarded application endpoints are permitted.

Links. A link represents a communication channel. DTN2 defines several types of links: always on, on demand, scheduled and opportunistic. Opportunistic links are typically discovered links, such as via a DTN2 discovery protocol or as the reciprocal link to an inbound DTN2 connection. An open link provides access to a specific node. In our implementation of RAPID, we manage the association of nodes and links such that if two links are concurrently open to the same node, we make use of only a single link.

Routes. A route identifies a link that can be used for communication with a node. RAPID defines two types of routes: persistent and temporal. Persistent routes are those defined by DTN2, such as when a route is configured in the DTN2 configuration file. Persistent routes typically identify which nodes can be contacted using specific always on, on demand or scheduled links. RAPID creates a temporal route whenever a link is opened to a specific node. The temporal route is destroyed when the link is closed. If there is a bundle to be sent to a node that has no associated active link, but if a persistent route exists to that node, RAPID may request that DTN2 open the link represented by the route. Persistent and temporal routes are defined by the RAPID implementation; they are not DTN2 constructs.

Policy Manager. This refers to the class that makes routing decisions. It is an implementation of the RAPID Policy interface. Calls are made to the policy manager when bundles, links and nodes are created or removed. The policy manager is responsible for deciding which bundles are to be sent to which nodes, which bundles are to be transmitted on which links, and when bundles are to be deleted from a system. It is also consulted before requesting that on demand links be opened.

Below is an overview of the primary Java classes used by the RAPID router.

RAPID

The RAPID class contains the main body of the router. This class reads the command line arguments, loads the configuration file (if defined), starts logging, and sets up the SAX (Simple API for XML) handler. Once initialized, it joins the DTN2 multicast group and continuously loops receiving locally broadcast messages from the DTN2 daemon, dtnd. When XML messages are received from DTN2, the SAX handler is responsible for parsing the message and dispatching the appropriate method.

RAPID_SAX

This class is invoked when RAPID receives an XML message from the DTN2 daemon. It extends the Java SAX DefaultHandler class. RAPID_SAX parses the XML message and calls the corresponding method in the Handlers class.

Handlers

This is an abstract class that defines a method for each XML event message that may be received by RAPID_SAX. The RAPID_Routing class is the real implementation of the Handlers class. We use an abstract class that supplies null methods for all XML messages. If the class that extends Handlers, i.e. RAPID_Routing, does not support an event then the empty method in Handlers in invoked.

XMLTree

This is a utility class. When RAPID_SAX parses an XML message each element is placed in an XMLTree object. XMLTree objects may be linked to each other to represent the hierarchy of elements in an XML message. XMLTree objects have methods for accessing attributes and child elements.

RAPID_Routing

The RAPID_Routing class is the heart of the router, extending the methods defined by Handlers. It is here that the XML messages sent by the DTN2 daemon, as represented by XMLTree objects, are initially acted upon.

Logging

This is an interface that defines the logging class used by the RAPID router. RAPID provides two implementations of the Logging class: Log4j_Logging and Console_Logging. By default, Console_Logging is used though you can define which implementation to invoke via the RAPID configuration file. We use a configurable logging class so that others may implement their own loggers, allowing them to customize logging to meet specific needs.

Console_Logging

This is a simple implementation of the Logging interface that outputs logging messages to stdout. It is the default logging class

Log4j_Logging

This is a wrapper around the Apache log4j logger. Using this logger requires the log4j JAR. . Add the line “loggingClass=Log4j_Logging” to your RAPID configuration file to enable this logger.

ConfFile

This is a utility class that reads and parses the RAPID configuration file.

Bundle

A Bundle object represents a DTN2 bundle. RAPID creates an instance of a Bundle object for each bundle on the system.

Bundles

This class manages the set of individual Bundle objects.

Node

A Node object represents a node, e.g. dtn://node.dtn. RAPID creates a Node object whenever it learns of a node, such as when a link is established to a node, or when a received bundle references a node. The Policy Manager is also able to create Node objects as it learns about other nodes. Node objects contain three transmission queues for sending bundles: a meta data queue, a destination queue and a replica queue. The meta data queue is intended for bundles destined for that node's router. The destination queue is for bundles destined for that node; and the replica queue is for bundles to be replicated on the node for further routing. The implementation of the queues is maintained by the Policy Manager.

Nodes

This is the class that manages the set of individual Node objects.

Link

A Link object represents a DTN2 link and an instance is created whenever DTN2 notifies RAPID that it has created a link. A Link object may become associated with a Node object when the link is opened; a DTN2 link that is not open is not associated with a Node. When a link is open it represents communication with another node. RAPID will associate the Link with the corresponding Node object, unless the Node object is already associated with another Link object. A node will never be associated with more than one link, even if there are multiple links open to the same node. Link objects use a separate Link Thread, in addition to executing on the main RAPID thread. The Link Thread is generally dormant unless the link is open and the Link object is associated with a Node object. The Link Thread is responsible for removing a bundle from one of the Node's three transmission queues and initiating a request to DTN2 for the bundle be transmitted.

Links

The Link class manages the set of individual Link objects.

Routes

The Routes object maintains the persistent and temporal routes. A route is simply a mapping of a Node (as identified by the node portion of the EID) to a Link. Persistent routes identify links that are either open or closed. A temporal route maps a Node to an open link. Temporal links are created when a link is opened, even if there exists a persistent route. A node can be mapped to more that one link via separate routes. Note that a temporal route is separate from the association between a Link object and a Node object described above. A temporal link is a prerequisite for an association, but the association defines the Node object's active link. As previously noted, a Node object will never have more than one active link.

Policy

The Policy class defines the interface to be implemented by a Policy Manager. The interface source code describes the individual methods. By default, RAPID_Policy implements this class, but other implementations can be defined via the RAPID configuration file.

RAPID_Policy

This class is an implementation of the Policy class. It provides the RAPID routing algorithm, but it is also generically referred to as the Policy Manager. There are calls into the Policy class sprinkled throughout the router, often mirroring the XML events defined by the /etc/router.xsd schema file. RAPID_Policy largely consists of manipulating shadow data structures dealing with bundles and nodes. The primary function of RAPID_Policy is to prioritize the delivery and replication of bundles in anticipation of the local node coming into contact with another node. The assumption is that RAPID will be able to replicate only a subset of its bundles on each node that it meets, and that some of the bundles will expire before RAPID comes in contact with the actual destination node. RAPID has no *a priori* knowledge of the network state, but as it comes into contact with other nodes it exchanges meta data with its RAPID peers to help it make better decisions.

LinkPolicyManager

This is another interface class, and by default there is no implementation of the class. This interface is specific to the RAPID_Policy class rather than defined by the RAPID router. If an implementation of this class exists, RAPID_Policy will call into the LinkPolicyManager each time a link is created, deleted, opened or closed, or if there is a request by the router to open a link. This class can also take advantage of RAPID's XML interface into DTN2 for creating, opening and closing links. The intent of this interface is to allow for system- or network-specific link management routines to be utilized, possibly including discovery of opportunistic links. For example, an implementation of this interface exists for DieselNet, where the interface is aware of how DieselNet manages its network connections. The DieselNet implementation creates, opens and closes DTN2 links in conjunction with DieselNet's bus-discovery mechanisms.

PeerListener

The PeerListener object exists as a thread dedicated to processing router-specific bundles received by peer routers. DTN2 specifies that any EID with an endpoint that begins with ext.rtr (e.g., dtn://node.dtn/ext.rtr/RAPID) is to be destined for an external router, and it generates a specific XML event when it receives a bundle containing the ext.rtr endpoint. When the RAPID_Routing class receives the event, it dispatches the PeerListener object's thread to process the bundle. Specifically, the PeerListener thread extracts the data from the bundle, deletes the bundle, and then makes a call into the Policy Manager passing the bundle's data. This is the recipient half of the mechanism for exchanging meta data between routers. To send meta data, the policy manager must inject a bundle into DTN2. This is done by the Policy Manager via a method in the Requester class.

Requester

This is a utility class responsible for composing and sending all XML messages to the DTN2 daemon. These classes are used throughout RAPID. Most XML messages sent to DTN2 are stateless: an event may cause DTN2 to do something that results in an XML message being received by the router, but it is generally up to the router to correlate events. Also, events sent to DTN2 typically do not have negative acknowledgments. For example, the router may send a request to DTN2 to inject a bundle and later the router may receive an XML message indicating that a bundle was injected. But DTN2 will not send an XML message if it fails to inject the bundle. The same is true for transmitting bundles on a link.

To expand on injecting a bundle, since it is fundamental to RAPID exchanging meta data with a peer node: When the Requester injects a bundle for the router it assigns a unique request ID to the bundle. It is up to the policy manager to later associate the injected bundle with the request using the id. The Requester only plays a minor role in injecting bundles: it sends the request to DTN2 after the policy manager creates the data. But it should be noted that injected bundles are handled differently from other bundles by RAPID. RAPID creates a Bundle object for an injected bundle, but it does not retain knowledge of the bundle in the Bundles class.

GBOF

This is another utility class, and it contains static methods for manipulating the GBOF. This includes formatting the GBOF as XML as required by DTN2. It also includes methods for creating a hash key from the values that make up the GBOF. This hash key is used extensively and consistently throughout RAPID and the Policy Manager for referencing a bundle.

Configuring RAPID

RAPID can be customized through a configuration file specified in the runtime command line. There must be no more that one configuration option per line in the file, and must be specified using the syntax: variable=value. The value may be double or single-quoted. Lines that begin with a hash character (#) are treated as comments and ignored. Comments should not appear on the same line as a configuration value. The following options are supported.

The default values for most, if not all variables should be sufficient.

Variable	**Type**	**Default**	**Description**
routerEndpoint	string	ext.rtr/RAPID	Defines the external router's endpoint, which is typically used to receive control messages from peer instances of the router. An example is meta data exchanged between the routers. DTN2 special-cases endpoints that begin with "ext.rtr/". All communicating routers need to agree on the endpoint name.
multicastGroup	string	224.0.0.2	Defines the multicast group the router is to join for exchanging XML messages with the DTN2 daemon. This is defined by DTN2.
multicastPort	integer	8001	Defines the multicast port used by the DTN2 daemon. This is defined by DTN2.
multicastSends	boolean	false	Defines the type of socket used to send requests to dtnd. If true, use a multicast socket joined to the multicast group and a TTL of 0. If false, use a datagram socket bound to the loopback address. The goal is to send messages with their scope limited to the local system. On some systems the TTL is not honored. On other systems, not using a multicast socket does not work.
loopbackAddress	string	127.0.0.1	Address to bind to if multicastSends=false.

Variable	Type	Default	Description
xmlSchema	string	/etc/router.xsd	File containing the XML schema definition for the messages exchanged with dtnd, the DTN2 daemon. This file should be supplied by DTN2. The command line has precedence.
xmlValidate	boolean	true	If set to false then the XML received from the daemon is not validated against the schema.
loggingClass	string	Console_Logging	Allows for a user-specified logging class. The default is Console_Logging. A class supporting the Apache log4j logger is also provided.
logConfiguration	string		Logging configuration file. The command line has precedence. The format of the file is logging class specific. The command line has precedence.
logLevel	integer		Logging level. The command line has precedence. If not specified, the default is defined by the logger.
terminateWithDTN	boolean	true	By default RAPID terminates when dtnd indicates that it is shutting down. Set to false to override this behavior.
bundlesActiveCapacity	integer	384	Initial capacity of the Java HashMap of all bundles on the system.
linksHashCapacity	integer	16	Initial capacity of the Java HashMap of all links.
nodesHashCapacity	integer	32	Initial capacity of the Java HashMap of all nodes.
policyAckFile	string	/var/rapid_router/acks	RAPID_Policy specific: Defines the file where acknowledgments of RAPID-delivered bundles are checkpointed. This allows acks to be persistent if RAPID terminates. If set to an empty string, checkpointing will be disabled.

Variable	Type	Default	Description
policyMaxReplicationCount	integer	3	RAPID_Policy specific: RAPID gives replication priority to bundles whose known replication count doesn’t exceed this value.
policyLocalBundlesCapacity	integer	256	RAPID_Policy specific: Initial capacity of table of all present, locally generated bundles.
policyForeignBundlesCapacity	integer	256	RAPID_Policy specific: Initial capacity of table of all present, remote generated bundles.
policyAckedBundlesCapacity	integer	256	RAPID_Policy specific: Initial capacity of table of all known acknowledgments.
policyKnownBundlesCapacity	integer	512	RAPID_Policy specific: Initial capacity of table of all known bundles, present or otherwise.
policyNodeBundlesCapacity	integer	64	RAPID_Policy specific: Initial capacity of tables noting bundles possessed by a specific node.
policyNodeAcksCapacity	integer	64	RAPID_Policy specific: Initial capacity of tables noting acknowledgments possessed by a specific node.
policyNodeContactCapacity	integer	16	RAPID_Policy specific: Initial capacity of tables noting a specific node's contact with other nodes.
policyKnownNodesCapacity	integer	16	RAPID_Policy specific: Initial capacity of table of all known nodes.
policyDeliveryQueueCapacity	integer	24	RAPID_Policy specific: Initial size of the delivery queue.
linkPolicyClass	string		RAPID_Policy specific: Instructs RAPID to load the defined class implementing the LinkPolicyManager interface.

References

[1] A. Balasubramanian, B. N. Levine, and A. Venkataramani. DTN Routing as a Resource Allocation Problem. In Proceedings of ACM SiIGCOMM, August 2007.

[2] Delay Tolerant Networking Group, http://www.dtnrg.org.

[3] UMass DieselNet, http://prisms.cs.umass.edu/dome/index.php?page=umassdieselnet.