## Contents

1. Introduction ................................................................................................................................................. 3
2. Licensing ......................................................................................................................................................... 3
3. Downloading DTN2 source code and the patch files ....................................................................................... 4
4. Patching and building the oasys and DTN2 source code ............................................................................... 7
5. oasys Modifications ....................................................................................................................................... 9
6. Storage Updates ............................................................................................................................................... 10
7. IPN Scheme .................................................................................................................................................... 11
8. LTP over UDP (LTPUDP) Convergence Layer ............................................................................................. 12
9. LTP Authentication for the LTP over UDP Convergence Layer .................................................................... 16
10. STCP Convergence Layer ............................................................................................................................ 22
11. UDP Convergence Layer ............................................................................................................................. 23
12. Enhanced HOSC System (EHS) Router ........................................................................................................ 24
13. Delay Tolerant Payload Conditioning ....................................................................................................... 31
14. Wireshark Dissectors .................................................................................................................................... 41
Appendix A. NASA Open Source Agreement version 1.3 ............................................................................. 42
1. Introduction

This set of patches provides the modifications to DTN2 that we have made at Marshall Space Flight Center (MSFC) for the ground DTN gateway hosted at the Huntsville Operations Support Center (HOSC) in support of the International Space Station (ISS). Payload Developers’ ground systems will connect to the HOSC DTN Gateway using the TCP Convergence Layer (CL) to send and retrieve bundles. The bundles destined to payloads on the ISS are sent via the LTP over UDP (LTPUDP) Convergence Layer on the wire where they are “sniffed” and CCSDS IP Encapsulated before transmission to the ISS. Onboard the ISS, the UDP packets are extracted and put back on to the wire to the Payload DTN gateway which is an ION implementation. The Payload DTN gateway forwards the bundles on to the Payloads using either the TCP CL or the ION STCP CL.

Included in these patches are several experimental features that are not yet incorporated into the HOSC production software. These are LTPUDP Authentication which implements the LTP authentication extensions, Delay Tolerant Payload Conditioning (DTPC) and an ION compatible STCP Convergence Layer.

2. Licensing

These patches are released under the NASA Open Source Agreement Version 1.3 and the full text is included in Appendix A.

THIS OPEN SOURCE AGREEMENT ("AGREEMENT") DEFINES THE RIGHTS OF USE, REPRODUCTION, DISTRIBUTION, MODIFICATION AND REDISTRIBUTION OF CERTAIN COMPUTER SOFTWARE ORIGINALLY RELEASED BY THE UNITED STATES GOVERNMENT AS REPRESENTED BY THE GOVERNMENT AGENCY LISTED BELOW ("GOVERNMENT AGENCY"). THE UNITED STATES GOVERNMENT, AS REPRESENTED BY GOVERNMENT AGENCY, IS AN INTENDED THIRD-PARTY BENEFICIARY OF ALL SUBSEQUENT DISTRIBUTIONS OR REDISTRIBUTIONS OF THE SUBJECT SOFTWARE. ANYONE WHO USES, REPRODUCES, DISTRIBUTES, MODIFIES OR REDISTRIBUTES THE SUBJECT SOFTWARE, AS DEFINED HEREIN, OR ANY PART THEREOF, IS, BY THAT ACTION, ACCEPTING IN FULL THE RESPONSIBILITIES AND OBLIGATIONS CONTAINED IN THIS AGREEMENT.

Government Agency: NASA Marshall Space Flight Center
Government Agency Original Software Designation: MFS-33289-1
Government Agency Original Software Title: DTN2 Software Updates version 1
User Registration Requested. Please email: david.a.zoller@nasa.gov
Government Agency Point of Contact for Original Software: David Zoller
david.a.zoller@nasa.gov
3. Downloading DTN2 source code and the patch files

This patch was developed to be applied to the latest “tip” version of the DTN2 Sourceforge repository which is currently changeset [326755] – Fix C++11 related issues. Download the DTN2 snapshot from http://sourceforge.net/p/dtn/DTN2/ci/326755a3af4372d7209436ef5673696ccf3200/tree/):

File downloaded was: dtn-DTN2-326755a3af4372d7209436ef5673696ccf3ce3200.zip
You will also need the oasys package which is currently at changeset [0838b1] – Fix C++11 related issues. Download the oasys snapshot from [http://sourceforge.net/p/dtn/oasys/ci/0838b1c4880bc4df29ec859ca5e0a6ac4361e1c5/tree/](http://sourceforge.net/p/dtn/oasys/ci/0838b1c4880bc4df29ec859ca5e0a6ac4361e1c5/tree/):

File downloaded was: dtn-oasys-0838b1c4880bc4df29ec859ca5e0a6ac4361e1c5.zip
And finally, you will need the patch related files. Download the DTPC patch file and sample configuration files from (https://sourceforge.net/p/dtn/patches/8/) :

Files to download are: dtn2_mfs-33289-1.patch.gz, oasys_mfs-33289-1.patch.gz and configs.tar.gz.
4. Patching and building the oasys and DTN2 source code

Move the files to your development location and unzip or untar them in the same parent directory

- unzip dtn-oasys-0838b1c44880bc4df29ec859ca5ea6ac4361e1c5.zip
- unzip dtn-DTN2-326755a3af4372d7209436ef5673696ccfce3200.zip
  - I was not able to find an option that extracted files with their original dates on my RHEL5 system
- gunzip dtn2_mfs-33289-1.patch.gz
- gunzip oasys_mfs-33289-1.patch.gz
- tar -xf configs.tar.gz

Rename directory dtn-oasys-0838b1c44880bc4df29ec859ca5ea6ac4361e1c5 to oasys for simplicity of these instructions

- mv dtn-oasys-0838b1c44880bc4df29ec859ca5ea6ac4361e1c5 oasys

Rename directory dtn-DTN2-326755a3af4372d7209436ef5673696ccfce3200 to dtn2 for simplicity of these instructions

- mv dtn-DTN2-326755a3af4372d7209436ef5673696ccfce3200 dtn2

Building patched oasys

External requirements:
- Tool Command Language (TCL - pronounced "tickle")
- TCL development environment
- C development files for the Berkeley DB

Set default into the oasys directory and configure and build:

- cd oasys
- patch -p 1 -i ../oasys_mfs-33289-1.patch
  - make sure that there are no errors listed during the patching
- If you are running on an ARMv6 or ARMv7 architecture there is a patch available for the atomic functions and now is a good time to install it:
  - http://sourceforge.net/p/dtn/patches/7/
  - > patch -p 1 -i atomic-armv7.patch
- ./build-configure.sh
- ./configure --disable-debug-locking
  - you may need to add other options to enable or disable elements on your system
- make
  - if you add the option -j <#> you may get an error linking and just need to run it again
  - You do not need to run "make install"
Building patched DTN2

Move to your dtn2 directory and patch and build the DTN2 programs. Below is a typical configure command for building the base DTN2 server. New configuration options added in these patches are listed below.

- cd ../*dn2
- patch -p 1 -i ../dtn2_mfs-33289-1.patch
  - make sure that there are no errors listed during the patching
- ./build-configure.sh ../oasys
- ./configure --disable-ecl --disable-edp --with-odbc=no --with-oasys=../oasys
- make
- sudo make install

- If building an unpatched version of DTN2
  - you will need a fairly recent kernel for dtntunnel to build
    - it uses the IP_TRANSPARENT socket option (or comment it out in apps/Makefile if not needed)
  - you may need to install openssl 1.0.1 for some of the security files to build and by default it installs to /usr/local/ssl which you may need to specify on the configure command:
    - export LIBS=--ldl
    - ./configure --disable-ecl --disable-edp --with-bsp
      - --with-openssl=/usr/local/ssl/]

New DTN2 configuration options

Use the command ./configure --h to see a full list of options that can be applied when configuring DTN2. The new options added with these patches are:

- --enable-bid64bit increases the Bundle ID size to 8 bytes
- --with-acs enable Aggregate Custody Signals support
- --with-bdstats enable additional bundle daemon statistics
- --with-dtpc enable Delay Tolerant Payload Conditioning support
- --with-ecos enable Extended Class of Service support
- --with-ehsrouterr enable building our EHS External Router
  (requires Xerces C)
- --with-ltpudp enable our LTP over UDP Convergence Layer
- --with-ltpudpauth enable LTP Authentication for the LTP over UDP CL
  (requires --with-bsp and openssl 1.0.1 or higher)
5. **oasys Modifications**

The updates to the oasys infrastructure include:

- Improved configuration support for non-standard oasys directory name/location
- Added new command to change the current default logging level on the fly
  
  `log loglevel <level>`

- Updated the Notifier so that it tracks pending notifications once the pipe is full and no longer aborts if the pipe full for 1 minute
- Added a TimerReaperThread to periodically clean up canceled timers
- Enhanced the RateLimitedSocket with a “wait until sent” option and added a Leaky Token Bucket as an option
- Fixed a few bugs and a couple of outdated unit tests that would not compile
6. Storage Updates

All database updates have been consolidated into a new BundleDaemonStorage thread. Add/Update and Delete requests are queued for processing and then serviced by the thread. The Storage thread defaults to a 10ms interval between processing the events. This allows multiple updates to a bundle or link for example to be consolidated into a single database update. For example, as a bundle moves through processing the Forwarding Log is updated several times and each update rewrites the full bundle information to the database. The 10ms delay allows several of these updates to be consolidated into a single write and if the bundle gets forwarded fast enough it is even possible for the delete request to be included allowing the database update to be bypassed. You can adjust your database delay update tolerance by configuring the interval with the DTN2 command:

```
storage set interval <milliseconds>
```

Note that the payload is always written to file as the bundle is received – that is separate from the database update. We found that as the number of files in a directory increased to around 15,000 there was a noticeable increase in the time it took to open a new file. The bundle payload files are now stored in multiple subdirectories of 10,000 each.

For the Berkely Database, there is a new option to enable removing the old transaction logs which can quickly fill a partition with the amount of database updates performed. Use this command to enable this feature which is disabled by default:

```
storage set db_log_auto_removal true
```

If you are testing and want to stress the processing portions of the code, you can disable database storage with the command:

```
storage set db_storage_enabled false
```
7. IPN Scheme

You can now set an IPN Scheme alias for your node which allows bundle transmission to take advantage of the Compressed Bundle Header Extension (CBHE). You specify the IPN Scheme administrative EID for the node similar to the way you specify the DTN Scheme EID. You must specify the zero service number which is reserved for administration and here is an example for node number 57223:

```
route local_eid_ipn ipn:57223.0
```

You must always specify the DTN Scheme EID as before but the IPN Scheme EID is optional. The two names do not have to be related in any way – the DTN Scheme does not have to incorporate the node number. The IPN administrative registration is built into the DTN2 server just as the DTN Scheme version is and processes the relevant bundles such as the processing custody signal bundles. You can also direct dtnping (and ION bping) requests to the administrative EID and receive a response to determine if the node is online.

**Compressed Bundle Header Encoding (CBHE)**

Bundles that have IPN Scheme compatible EIDs for all of the source, destination, custody and reply-to EIDs can take compress the header as specified in RFC 6260 to eliminate the dictionary from the Bundle’s Primary Block. To facilitate this, when a bundle is received requesting custody that is destined to an IPN Scheme EID will be accepted into custody using the IPN Scheme administrative EID rather than the DTN Scheme EID.

**TCP Convergence Layer identifier**

There is an option to specify the IPN Scheme EID on TCP Convergence Layer connections which you enable with the DTN2 command:

```
param set announce_ipn true
```

**Echo service**

If a service number is assigned to the echo service and you wish to use that instead then there are two options. You can make use of a built in echo registration or the new dtnecho application. For the built-in version you need to specify the service number and the max return length which defaults to 1024 bytes, for example:

```
param set ipn_echo_service_number 3
param set ipn_echo_max_return_length 64
```

Combined with the above example, this starts an echo registration at EID: ipn:57223.3 and will reply to any received bundle by sending back a copy of the received payload up to 64 bytes.

The dtnecho application specifies the relevant details as command line parameters. You could start a second echo application on service number 1027 as follows:

```
> dtnecho -s ipn:57233.1027
```
8. LTP over UDP (LTPUDP) Convergence Layer

The LTP over UDP Convergence Layer is our initial implementation of the Licklider Transmission Protocol (LTP) for CCSDS as defined in CCSDS 734.1-R-3 to meet our specific use case of communicating from a DTN2 ground node to an ION LTP CL implementation onboard the ISS. This implementation has a limitation of only communicating with a single remote node. We are redesigning the implementation this year to support multiple nodes.

Building DTN2 with the LTPUDP CL

Add the `--with-ltpudp` option to the configure command:

```
    ./configure --disable-ecl --disable-edp --with-odbc=no --with-oasys=../oasys \
    --with-ltpudp
```

```
    make
    sudo make install
```

LTP Green and Extended Class of Service

If you are interested in exercising LTP Green then you will also need to enable Extended Class of Service (ECOS) support by adding the `--with-ecos` option. The utilities dtnping and dtntunnel have been updated to allow you to specify the ECOS values for bundle transmission. With these utilities, you can use the `-S` option to specify ECOS streaming which will be transmitted as LTP Green.

Configuring the LTPUDP CL

The current implementation utilizes a single “link add” command to instantiate both the receiver (which is usually started with an “interface add”) and the sender. The format of the command is:

```
link add <linkname> <remoteIP>:<remotePort> ALWAYSON ltpudp \ 
    engine_id=<ID> local_addr=<IP> local_port=<port> \ 
    remote_addr=<IP> remote_port=<port> \ 
    max_sessions=<count> seg_size=160000 \ 
    agg_size=<bytes> agg_time=<milliseconds> \ 
    inact_intvl=<seconds> retran_intvl=<seconds> retran_retries=<count> \ 
    bucket_type=0 rate=<bits/sec> \ 
    qlimit_bundles_high=400 qlimit_bundles_low=200 \ 
    qlimit_bytes_high=4000000 qlimit_bytes_low=2000000
```

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>linkname</td>
<td>&lt;no default&gt;</td>
<td>User defined name for the link</td>
</tr>
<tr>
<td>remoteIP</td>
<td>&lt;no default&gt;</td>
<td>IP address of remote LTP node (required)</td>
</tr>
<tr>
<td>remotePort</td>
<td>&lt;no default&gt;</td>
<td>UDP port remote LTP node is listening on (required)</td>
</tr>
<tr>
<td>engine_id</td>
<td>&lt;no default&gt;</td>
<td>LTP Engine ID of the local node</td>
</tr>
<tr>
<td>local_addr</td>
<td>All interfaces</td>
<td>Interface to listen on for incoming UDP packets</td>
</tr>
<tr>
<td>local_port</td>
<td>1113</td>
<td>UDP port to listen on</td>
</tr>
<tr>
<td>remote_addr</td>
<td>&lt;no default&gt;</td>
<td>IP address of remote LTP node (required –will default to <code>remoteIP</code> next release :)</td>
</tr>
<tr>
<td>remote_port</td>
<td>1113</td>
<td>UDP port remote LTP node is listening on</td>
</tr>
<tr>
<td>max_sessions</td>
<td>100</td>
<td>Max number of active local LTP (sending) sessions to allow</td>
</tr>
<tr>
<td>seg_size</td>
<td>1400</td>
<td>Max LTP Segment size – recommend 16000 if network supports fragmentation</td>
</tr>
<tr>
<td>agg_size</td>
<td>100,000</td>
<td>Accumulate bundles until at least this number of bytes are reached before generating and sending an LTP session</td>
</tr>
</tbody>
</table>
Sample configuration files

Sample configuration files are included in the configs.tar.gz file that you downloaded and extracted. In the subdirectory configs/ltpudp there are the following files:

- `node1_ltpudp.cfg` - sample config file for node1
- `run_node1_ltpudp.sh` - script to run DTN2 using node1_ltpudp.cfg
- `node2_ltpudp.cfg` - sample config file for node2
- `run_node2_ltpudp.sh` - script to run DTN2 using node2_ltpudp.cfg

The configuration files are for running the two nodes on different systems. You will need to edit both the script files and the configuration files to tailor them to your specific setup.

Edit the `run_node1_ltpudp.sh` file and change the database directory location to fit your needs:

```bash
#!/bin/bash
dbdir=/tmp/node1
mkdir "${dbdir}"
...
dtnd -t -l error -c node1_ltpudp.cfg -o "${dbdir}/dtn.log"
```

And then edit the `node1_ltpudp.cfg` file to make a similar update in there and to also set the appropriate IP addresses for your layout. In the top portion of the file are parameters defining the “local” node (node #1 in this case) and the “remote” node (node2 in this case). At a minimum, you need to update the local_ip_address and the remote_ip_address values:

```bash
...
set local_ip_address 192.168.1.1
...
set local_ip_address 192.168.1.2

Also verify that the local_log_dir and local_db_dir match the dbdir that you set in the run_node1_ltpudp.sh:

... 
set local_log_dir /tmp/${local_node_name}
set local_db_dir /tmp/${local_node_name}
...

${local_node_name} is replaced with the value of local_node_name which defaults to node1 in this case so the result would be /tmp/node1 which matches the initial value in the run_node1_ltpudp.sh script.

Further down in the file, you will see the “link add” command for the ltpudp convergence layer and the “route add” commands to send bundles destined to both the dtn scheme and the ipn scheme variants of node2 through the link.

Of course you can edit any of the other elements to suit your needs.

Make similar changes to the run_node2_ltpudp.sh script and the node2_ltpudp.cfg files and then run the script files on their respective systems.

**Running applications**

Once the DTN2 servers are running on both nodes, there are several applications that can be used to send bundles between them. Below are some examples:

1) **dtnping**

Send “ping” bundles from node1 to the built-in responders on node2:

```bash
> dtnping -s dtn://node1.dtn/sender dtn://node2.dtn/ping
```

Or using the IPN scheme (ipn:2.0 is the IPN administrative endpoint for node 2 and will reply to a payload that begins with text containing “ping”:

```bash
> dtnping -s ipn:1.5 ipn:2.0
```

Or mix and match schemes:

```bash
> dtnping -s dtn://node1.dtn/sender ipn:2.0
```

Note that dtnping has been updated to allow specification of the ECOS parameters and the interval options is now milliseconds rather than seconds to provide it with some additional quick test possibilities.
2) `dtnsource` and `dtnrecv`

To change things up let’s send bundles from node2 to node1:

On node1, run the receiving application (note that “-s” in this case is for statistics and not source node):

```shell
> dtnrecv -s -n 10000 dtn://node1.dtn/rcvr
```

Then on node2, run the transmitter:

```shell
> dtnsource -s dtn:node2.dtn/sndr -d dtn://node1.dtn/rcvr -b 1024 -n 10000
```

3) `dtnperf-client` and `dtnperf-server`

Send bundles from node1 to node2:

On node2, run the server:

```shell
> dtnperf-server -m
```

Then on node1, run the client:

```shell
> dtnperf-client -d dtn://node2.dtn -t 30 -m -p 1024B
```

* There is a newer version of DTNPerf_3 available at [http://cnrl.deis.unibo.it/new/software.php](http://cnrl.deis.unibo.it/new/software.php)
9. LTP Authentication for the LTP over UDP Convergence Layer

The LTP over UDP Convergence Layer can optionally be built with support for the LTP Authentication mechanisms defined in section 2.1 of RFC 5327. We make use of some of the infrastructure in the Bundle Security Protocol (BSP) implementation which so they must be enabled when DTN2 is built.

Building DTN2 with the LTPUDP CL and LTP Authentication

Add the --with-ltpudp, --with-ltpudpauth and --with-bsp options to the configure command:

```bash
  export LIBS=-ldl
  ./configure --disable-ecl --disable-edp --with-odbc=no --with-oasys=../oasys \   --with-ltpudp --with-ltpudpauth --with-bsp \   [--with-openssl=/usr/local/ssl/]
  make
  sudo make install
```

Note that this requires openssl 1.01 or above so you may need to install it and the development packages. By default, they are installed on the /usr/local/ssl directory which you might need to point the configuration to with the option: --with-openssl=/usr/local/ssl

Configuring the LTPUDP CL

The parameters for LTP Authentication are specified on the “link add” statement and are highlighted in bold below. The Ciphersuite(s) and key(s) to be used for communicating with the remote LTP node must be specified. See the previous section for a discussion of the other parameters. The expanded format of the command is:

```bash
  link add <linkname> <remoteIP>:<remotePort> ALWAYSON ltpudp \   engine_id=<ID> local_addr=<IP> local_port=<port> \   remote_addr=<IP> remote_port=<port> \   max_sessions=<count> seg_size=160000 \   agg_size=<bytes> agg_time=<milliseconds> \   inact_intvl=<seconds> retran_intvl=<seconds> retran_retries=<count> \   bucket_type=0 rate=<bits/sec> \   qlimit_bundles_high=400 qlimit_bundles_low=200 \   qlimit_bytes_high=4000000 qlimit_bytes_low=2000000 \   icipher_suite=<suite#> icipher_engine=<ID> icipher_keyid=<key> \   ocipher_suite=<suite#> ocipher_engine=<ID> ocipher_keyid=<key>
```

The additional parameters for LTP Authentication are:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>icipher_suite</td>
<td>-1</td>
<td>The inbound cipher suite setting for authentication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 = Do not check authentication on inbound segments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Require HMAC-SHA1-80 authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Require RSA-SHA256 authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>255 = Require NULL ciphersuite authentication</td>
</tr>
<tr>
<td>icipher_engine</td>
<td>&lt;no default&gt;</td>
<td>LTP Engine ID of the local node</td>
</tr>
<tr>
<td>icipher_keyid</td>
<td>&lt;no default&gt;</td>
<td>The inbound key required for authentication</td>
</tr>
<tr>
<td>ocipher_suite</td>
<td>-1</td>
<td>The outbound cipher suite setting for authentication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>-1 = Do not apply authentication on outbound segments</td>
</tr>
<tr>
<td></td>
<td></td>
<td>0 = Apply HMAC-SHA1-80 authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1 = Apply RSA-SHA256 authentication</td>
</tr>
<tr>
<td></td>
<td></td>
<td>255 = Apply NULL ciphersuite authentication</td>
</tr>
<tr>
<td>ocipher_engine</td>
<td>&lt;no default&gt;</td>
<td>The engine ID of the remote LTP node</td>
</tr>
<tr>
<td>ocipher_keyid</td>
<td>&lt;no default&gt;</td>
<td>The outbound key to use for authentication</td>
</tr>
</tbody>
</table>
**NULL Ciphersuite - HMAC-SHA1-80 with fixed key**

The NULL Ciphersuite is selected by specifying the value 255 for the `icipher_suite` and/or `ocipher_suite`. It utilizes the hardcoded key specified in RFC 5327 with HMAC-SHA1-80 (Ciphersuite 0) and therefore is not truly secure but it does provide a quick means to verify basic inter-operability between nodes using Ciphersuite 0. You do not need to specify the key value in the DTN2 configuration for the NULL Ciphersuite since it is predefined and hardcoded. Likewise a key ID is not required either but if specified then it must be specified the same on both the transmitting node (`ocipher_keyid`) and the receiving node (`icipher_keyid`).

Sample configuration files are included in the `configs/ltpudpauth_cs255` directory:

```
node1_ltpudpauth_cs255.cfg       - sample config file for node1
run_node1_ltpudpauth_cs255.sh    - script to run DTN2 using node1_ltpudpauth_cs255.cfg

node2_ltpudpauth_cs255.cfg      - sample config file for node2
run_node2_ltpudpauth_cs255.sh    - script to run DTN2 using node2_ltpudpauth_cs255.cfg
```

Edit the run script and configuration files as detailed in the previous section to specify the correct IP addresses and desired database location. Notice the additions to the “link add” line further down in the configuration file:

```
link add ${remote_link_name} ${remote_ip_address}:${remote_ltp_port} ALWAYSON ltpudp
...
   icipher_suite=255 ocipher_suite=255
```

Run the scripts to start the two DTN2 servers and then run the desired application(s) to verify that bundles pass through properly. If you would like to verify that LTP segments which fail authentication are rejected then you can run the `node1` script from the prior section which does not specify the cipher suites and attempt to send bundles from node1 again. You should see in the log file on node2 that there are messages indicating that no Cipher Trailer was found.

**Ciphersuite 0 – HMAC-SHA1-80**

Ciphersuite 0 uses the HMAC-SHA1-80 algorithms to generate a hash of the LTP segments. The symmetric key(s) for the hash are specified using the “security setkeyltp” command:

```
security setkeyltp <engine> <cs_num> <key> <key_id>
```

where:

- `<engine>` = the engine ID the key is associated with
- `<cs_num>` = the Ciphersuite Number for the key (must be zero)
- `<key>` = the 20 byte key value in hexadecimal format (40 hex chars)
- `<key_id>` = the ID of this key value

And then you must specify the corresponding values in the “link add” command for the LTPUDP convergence layer.
Sample configuration files are included in the `configs/ltpudpauth_cs0` directory:

- **node1_ltpudpauth_cs0.cfg** - sample config file for node1
- **run_node1_ltpudpauth_cs0.sh** - script to run DTN2 using node1_ltpudpauth_cs0.cfg
- **node2_ltpudp auth_cs0.cfg** - sample config file for node2
- **run_node2_ltpudp auth_cs0.sh** - script to run DTN2 using node2_ltpudp auth_cs0.cfg

Edit the run script and configuration files as detailed in the previous section to specify the correct IP addresses and desired database location. In the configuration files you will find an LTP Security Configuration section where the keys are defined:

```plaintext
########################################
# LTP Security configuration
# ####################################
# Specify keys for LTP Ciphersuite 0:
# security setkeyltp <engine> <cs_num> <key> <key_id>
# (note that the key must be 20 bytes and thus 40 hex characters)
# Define a key for Node1 Ciphersuite0 (Key ID #11):
security setkeyltp 1 0 1234567890123456789012345678901234567890 11
# Define a key for Node2 Ciphersuite0 (Key ID #22):
security setkeyltp 2 0 1234567890abcdef1234567890abcdef12345678 22
```

Notice the additions to the “link add” line further down in the configuration file for node1:

```plaintext
link add ${remote_link_name} ${remote_ip_address}:${remote_ltp_port} ALWAYSON ltpudp\`
...  
icipher_suite=0 icipher_engine=1 icipher_keyid=11 \ 
ocipher_suite=0 ocipher_engine=2 ocipher_keyid=22
```

In the node2 configuration file, you will see that the LTP Security Configuration section is the same but in the convergence layer configuration the inbound and outbound cipher definitions are reversed.

```plaintext
link add ${remote_link_name} ${remote_ip_address}:${remote_ltp_port} ALWAYSON ltpudp\`
...  
icipher_suite=0 icipher_engine=2 icipher_keyid=22 \ 
ocipher_suite=0 ocipher_engine=1 ocipher_keyid=11
```
Run the scripts to start the two DTN2 servers and then run the desired application(s) to verify that bundles pass through properly. If there is an error in authentication then no bundles will pass through to the application and the transmit node will continually keep trying to send its bundles.

Ciphersuite 1 – RSA-SHA256
Ciphersuite 1 uses the RSA-SHA256 algorithms to generate a “signature” for each LTP segment. The asymmetric keys for the signature can be generated using openssl and are stored as files named with a specific format.

First create directories to hold the private and public key files and then specify their locations in the DTN2 configuration file. In the example excerpt from a configuration file below, the assumption is that the DTN2 server will be run such that the keys are stored in subdirectories below the current working directory:

```
# LTP Security Configuration
# Specify location for keys for LTP Ciphersuite 1:
# Local node private keys:
security privdir ./node1_privkeys
# Public keys/certificates from remote nodes:
security certdir ./pubkeys
```

With these directories in mind, first generate an RSA public key for node #1. The filename for the private key must be formatted as “ltp_<cipher_suite>_<engine>_<keyid>_privkey.pem”. Currently the only defined Ciphersuite that uses this type of key is “1” but there could more defined in the future. Now let’s generate the private key for node/engine #1 and Key ID #33 with a key that is 1024 bytes long:

```
> openssl genrsa -out ./node1_privkeys/ltp_1_1_33_privkey.pem
```

Next the public key formatted as an x509 certificate must be generated from the private key. Once again the filename must be formatted appropriately: “ltp_<cipher_suite>_<engine>_<keyid>_cert.pem”. openssl is used to generate the public key/certificate file:

```
> openssl req -new -x509 -key ./node1_privkeys/ltp_1_1_33_privkey.pem \\
-out ./pubkeys/ltp_1_1_33_cert.pem
```

(Note that there will be prompts for information for inclusion in the certificate)

This public key or certificate file is needed by the remote node that will be receiving LTP segments signed with the private key to verify that they have not been altered. In this example, node #1 uses its private key to “sign” an LTP
Segment and then node #2 uses the public key/certificate to verify the signature is valid which indicates that the segment has not been altered in transit.

If LTP authentication will also be used when sending segments from node #2 back to node #1 then a private key file will need to be generated for node #2 and its corresponding public key/certificate file installed on node #1. In the provided sample configuration files, we used Key ID #44 for the node #2 key.

Sample configuration files are included in the configs/ltpudpauth_cs1 directory:

```
node1_ltpudpauth_cs1.cfg       - sample config file for node1
run_node1_ltpudpauth_cs1.sh    - script to run DTN2 using node1_ltpudpauth_cs1.cfg

node2_ltpudp auth_cs1.cfg      - sample config file for node2
run_node2_ltpudp auth_cs1.sh    - script to run DTN2 using node2_ltpudp auth_cs1.cfg

node1_privkeys/ltp_1_1_33_privkey.pem - private key for node #1 (Key ID 33)
node2_privkeys/ltp_1_2_44_privkey.pem - private key for node #2 (Key ID 44)

pubkeys/ltp_1_1_33_cert.pem     - public key for node #1 (Key ID 33)
pubkeys/ltp_1_2_44_cert.pem     - public key for node #2 (Key ID 44)
```

Edit the run script and configuration files as detailed in the previous section to specify the correct IP addresses and desired database location. In the configuration files you will find an LTP Security Configuration section similar to the excerpt above which you can change if desired to store keys in a different location.

Notice the additions to the “link add” line further down in the configuration file for node1:

```
link add ${remote_link_name} ${remote_ip_address}:${remote_ltp_port} ALWAYSON ltpudp\ ...
  icipher_suite=1 icipher_engine=2 icipher_keyid=44 \
  ocipher_suite=1 ocipher_engine=1 ocipher_keyid=33
```

The configuration this time is a bit counterintuitive after working with Ciphersuite 0 because the inbound and outbound key info is reversed. The remote node #2 is signing its segments with its private key and node #1 is verifying it using node #2’s public key so the inbound cipher is actually node #2’s engine and Key ID. The outbound cipher in this case is node #1’s engine and Key ID.

In the node #2 configuration file, you will see that one again in the convergence layer configuration the inbound and outbound cipher definitions are reversed compared to that of node #1.

```
link add ${remote_link_name} ${remote_ip_address}:${remote_ltp_port} ALWAYSON ltpudp\ ...
  icipher_suite=1 icipher_engine=1 icipher_keyid=33 \
  ocipher_suite=1 ocipher_engine=2 ocipher_keyid=44
```
Run the scripts to start the two DTN2 servers and then run the desired application(s) to verify that bundles pass through properly. If there is an error in authentication then no bundles will pass through to the application and the transmit node will continually keep trying to send its bundles.
10. **STCP Convergence Layer**

The STPC Convergence Layer is an ION innovation that is a (S)imple version of the TCP Convergence Layer. The data format is a simple 4 byte length followed by the bytes of the bundle. The bundle transmission is strictly one way unlike the TCP CL which is two-way and there are no initial handshakes and no ACKs above the TCP layer.

**Building DTN2 with the STCP CL**

There are no special parameters required as it is included in the base configuration.

**Configuring the STCP CL**

The STCP CL configuration is exactly like the TCP CL except you specify the “stcp” convergence layer type instead of “tcp”. We default the port to 4557 so that it can coexist with the TCP CL running on its IANA designated port 4556.

Add an interface for incoming connections:

```
interface add stcp0 stcp
```

Add link(s) for outgoing connections:

```
link add remote_link 10.0.0.2:4557 ALWAYSON stcp
```

And then add routes to specify which bundles should be sent on the link:

```
route add dtn://node2.dtn/* remote_link
route add ipn:2.* remote_link
```

And finally, use your favorite applications to send and receive bundles.
11. UDP Convergence Layer

The UDP Convergence Layer implementation has been enhanced a bit to include an option to let the rate controller to wait until the bundle can be transmitted rather than returning immediately with an error. The new “wait_and_send” Boolean parameter defaults to true so this will be slightly different behavior than the base DTN2.9 implementation.

You can also specify a Leaky Token Bucket if desired by using the new “bucket_type” integer parameter which defaults to a Standard Token Bucket (bucket_type=0) or you can choose the Leaky approach with the value 1.
12. **Enhanced HOSC System (EHS) Router**

The EHS Router is an external router which is solely IPN Scheme based that we use in the Huntsville Operations Support Center (HOSC) to control the HOSC DTN Gateway and its LTPUDP Convergence Layer’s state and the rate of the transmission for communicating with the DTN gateway onboard the International Space Station. It is interfaced to the EHS systems to allow control of when payload developers access to uplink bundles to their payload. Control of the uplink is essential as it is nominally an 8 Mbps pipe shared by all payload developers for DTN as well as other IP based access.

The ExternalRouter and router XML definitions have been updated to utilize the Bundle ID as the main means of identifying specific bundles when issuing requests to the DTN2 server. The EHS Router uses this interface to manage all bundle transmissions and to reconfigure the LTPUDP CL. It also routes bundles sent from the ISS down to the Payload Developer’s ground node(s) to the appropriate incoming TCP CL when they connect to retrieve their bundles. The EHS Router is basically a standalone bundle router with an API to (re)configure it on an ongoing basis. It manages a single “Forward Link” or “uplink” convergence layer (not limited to the LTPUDP CL we use) to a remote gateway node and allows retrieval of downlinked bundles via the TCP CL connections. The EHS router does support the Extended Class of Service (ECOS) bundle prioritizing.

**Building oasys and DTN2 with the EHS Router**

Xerces-C++ is a prerequisite for utilizing the EHS Router or any External Router and both oasys and DTN2 must be configured with its location. First install and build Xerces-C 2.8.0 from the source code (xerces-c-src_2_8_0.tar.gz) which can be downloaded from this site:

http://archive.apache.org/dist/xerces/c/2/sources/

or use the direct link:

http://archive.apache.org/dist/xerces/c/2/sources/xerces-c-src_2_8_0.tar.gz

Extract the source code to a desired location and build it:

- tar -xf xerces-c-src_2_8_0.tar.gz
- cd xerces-c-src_2_8_0
- export XERCESCROOT="pwd"
- cd /src/xercesc
- ./runConfigure -plinux -cgcc -xg++ -rnone -s -z-fPIC
- make

Switch to the oasys directory and reconfigure it (assuming that you have already gone through the patch and build steps above) and rebuild:

- cd <path to oasys>
- (In RHEL7.2, had to add: export LIBS=-lpthread )
- ./configure --disable-debug-locking \  
  --with-xerces-c=<full path to xerces-c-src_2_8_0>
- grep XERCES oasys-config.h
  o verify: #define XERCES_C_ENABLED 1
- make

Next switch to the dtn2 directory, reconfigure and rebuild:

- ./configure --with-ecos --with-odbc=no --with-oasys=../oasys \  
  --with-ltpudp --with-ehsrouter

- make
Sample configuration files

Sample configuration files are included in the configs.tar.gz file that you downloaded and extracted. In the subdirectory configs/ehsrouter there are the following files:

- **node1_ltpudp.cfg** - sample DTN2 config file for node1
- **run_node1_ltpudp.sh** - script to run DTN2 using node1_ltpudp.cfg
- **node1_ehsrouter.cfg** - sample EHS Router config file for node1
- **run_node1_ehsrouter.sh** - script to run the EHS Router for node1

- **node2_ltpudp.cfg** - sample config file for node2
- **run_node2_ltpudp.sh** - script to run DTN2 using node2_ltpudp.cfg
- **node2_ehsrouter.cfg** - sample EHS Router config file for node2
- **run_node2_ehsrouter.sh** - script to run the EHS Router for node2

- **update.cfg** - sample config file used to issue on the fly commands to the EHS Router

The configuration files are for running the two nodes on different systems. You will need to edit the script files and the configuration files to tailor them to your specific setup.

Edit the **run_node1_ltpudp.sh** file and change the database directory location to fit your needs:

```bash
#!/bin/bash
dbdir=/tmp/node1
mkdir "${dbdir}"

...
dtnd -t -l error -c node1_ltpudp.cfg -o "${dbdir}/dtn.log"
```

And then edit the **node1_ltpudp.cfg** file to make a similar update in there and to also set the appropriate IP addresses for your layout. In the top portion of the file are parameters defining the “local” node (node #1 in this case) and the “remote” node (node2 in this case). At a minimum, you need to update the local_ip_address and the remote_ip_address values:

```bash
...
set local_ip_address 192.168.1.1
...
set local_ip_address 192.168.1.2
```
Also verify that the `local_log_dir` and `local_db_dir` match the `dbdir` that you set in the `run_node1_ltpudp.sh`:

```
...  
set local_log_dir /tmp/${local_node_name}
set local_db_dir /tmp/${local_node_name}
...  
```

`${local_node_name}` is replaced with the value of `local_node_name` which defaults to `node1` in this case so the result would be `/tmp/node1` which matches the initial value in the `run_node1_ltpudp.sh` script.

Further down in the file, you will see that the router type is configured as `external` instead of `static` and the associated parameters to use the loopback for the communication between the DTN2 server and the EHS Router which runs as a separate task:

```
####################################
####
# Routing configuration
#
########################################
#
# Set the algorithm used for dtn routing.
#
# route set type [static | flood | neighborhood | linkstate | external]
#
#route set type static
route set type external
route set server_port 8001
route set hello_interval 10
route set schema /etc/router.xsd
route extrtr_multicast 232.44.44.44
route extrtr_interface 127.0.0.1
```

Even further down, you will find the TCP CL interface loaded and the LTPUDP CL for communicating with the remote DTN gateway. Also note that there are no routes specified in this configuration file as those are handled by the EHS Router.

Edit the `run_node1_ehsrouter.sh` file and change the database directory location to fit your needs:

```
#!/bin/bash
dbdir=/tmp/node1
mkdir "${dbdir}"
...

test_ehsrouter -c node1_ehsrouter.cfg -l info -o ${dbdir}/ehsrouter.log
```
Finally, edit the `node1_ehsrouter.cfg` file. The parameters that specify the communication with the DTN2 server must match those in the `node1_ltpudp.cfg` file:

```
# Multicast Address in use by the DTN2 External Router
EXTERNAL_ROUTER_MC_ADDRESS = 232.44.44.44

# Port in use by the DTN2 External Router
EXTERNAL_ROUTER_PORT = 8001

# Network Interface on which to listen
# - "localhost" or "lo" or "any" or "eth0", "eth1", etc. or IP Address of a local interface
NETWORK_INTERFACE = lo

# XML Schema file defining message formats
SCHEMA_FILE = /etc/router.xsd
```

The rest of the file assumes that node #1 is the local DTN Gateway that allows local nodes #1000 through #1003 to connect in to transmit bundles through it to the remote DTN Gateway which is node #2 or to any of three nodes (#2001 through #2003) reachable from node #2. This is similar to our usage with the ISS except that the onboard gateway (#2) is an ION implementation rather than DTN2 and obviously it would not use the EHS Router.

First the FORWARD_LINK convergence layer is defined by name, rate and the nodes that are reachable through it:

```
### Configure iss_link_ltp at 20 Mbps and provide the list of reachable nodes:
#     2 is the node number of the remote gateway and
#     nodes 2000 through 2099 are reachable through node 2
FORWARD_LINK = remote_link`20000000`2,2001-2003
```

First the FORWARD_LINK convergence layer is defined by name, rate and the nodes that are reachable through it:

```
### Configure iss_link_ltp at 20 Mbps and provide the list of reachable nodes:
#     2 is the node number of the remote gateway and
#     nodes 2000 through 2099 are reachable through node 2
```
FORWARD_LINK = remote_link`20000000`2,2001-2003

Next the nodes that are currently allowed to transmit through the FORWARD_LINK are specified. These are enabled on the fly in the HOSC but for demo purposes we will enable them all by default:

```markdown
### Enable transmit from the HOSC DTN GW itself
# (actually does not need to be specified as the gateway can always transmit)
FWDLINK_TRANSMIT_ENABLE = 1`2,2001-2003
### Allow transmission from any local node (1001, 1002 or 1003) to any remote node (2, 2001, 2002 or 2003)
FWDLINK_TRANSMIT_ENABLE = 1001-1003`2,2001-2003
```
For demo purposes, we will just transmit bundles directly from the DTN Gateway node #1 but if you have the resources you could create the other local nodes (#1001 through #1003) and send from them via the TCP CL. To allow them to connect in to the node #1 Gateway you would specify their source IP address that they will connect from:

```bash
# Examples:
# "192.168.1.74`false`3774,3776;3969,33050-33053" - Allow an incoming TCP connection from
# 192.168.1.74 and then accept bundles with
# source node of 3774 or 3776 and destinations
# of 3969, 33050, 33051, 33052 or 33053
# >> do not establish an outgoing connection
LINK_ENABLE = 192.168.1.101`false`1001`2,2001-2003
LINK_ENABLE = 192.168.1.102`false`1002`2,2001-2003
LINK_ENABLE = 192.168.1.103`false`1003`2,2001-2003
```

Make similar changes to the node2 scripts and configuration files allowing communication in the reverse direction.

**Running the EHS Router**

You will need at least 3 terminal windows on each system: one to run the DTN2 server (dtnd), one to run the EHS Router (test_ehsrouter) and one to run an application.

First start node1. On node1 in window 1, run the DTN2 server:

```
`/run_node1_ltpudp.sh`
```

You should see the prompt “node1 dtn%” if successful otherwise check the log file to see determine the error. The log file is /tmp/node1/dtn.log unless you changed the default directory in your editing.

On node1 in window2, run the EHS Router:

```
`/run_node1_ehsrouter.sh`
```

When the EHS Router detects the DTN2 server the status line at the bottom will indicate that it has detected 1 node and show the current bundle statistics for # received, # transmitted, # of transmit failures, # delivered, # rejected, # pending and # currently in custody:

<table>
<thead>
<tr>
<th>EHS External Router Test Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid keyboard inputs:</td>
</tr>
<tr>
<td>b = bundle stats</td>
</tr>
<tr>
<td>B = bundle list</td>
</tr>
<tr>
<td>r = bundle stats: src-dst</td>
</tr>
<tr>
<td>l = link dump</td>
</tr>
<tr>
<td>f = fwdlink transit dump</td>
</tr>
<tr>
<td>e = bundle delete 32774-&gt;32969</td>
</tr>
<tr>
<td>w = enable forward link</td>
</tr>
<tr>
<td>A = Simulate AOS</td>
</tr>
<tr>
<td>t = toggle statistics update</td>
</tr>
<tr>
<td>d = display this header</td>
</tr>
<tr>
<td>y = toggle show interval stats</td>
</tr>
<tr>
<td>x = exit</td>
</tr>
<tr>
<td>z = send/recv stats</td>
</tr>
<tr>
<td>c = process config; update.cfg</td>
</tr>
</tbody>
</table>

States and Statistics:
Nodes: 1 (enabled/AOS) Bundles Rcv: 0 Xmt: 0 XFail: 0 Dlv: 0 Rjct: 0 Pend: 0 Cust: 0

29
The keyboard inputs list the commands that you can issue to the EHS Router and notice that they are case sensitive. Typical usage would be to simulate AOS/LOS with the <A> and <L> keys which for us stands for “Acquisition of Signal” and “Loss of Signal”. While AOS, we are in communication with the ISS and the convergence layer transmission is enabled – be careful to press shift key to enter the upper case <A> key rather than lower case <a> which deletes all bundles! You can also enter commands such as changing the transmission rate into the update.cfg file and then invoke processing of the file by pressing the <c> key.

Next start node2. On node2 in window 1, run the DTN2 server:

```
./run_node2_ehsrouter.sh
```

You should see the prompt “node2 dtn%” if successful otherwise check the log file to see determine the error. The log file is /tmp/node2/dtn.log unless you changed the default directory in your editing.

On node2 in window2, run the EHS Router:

```
./run_node2_ltpudp.sh
```

You should see the EHS Router display similar to above.

Now, let’s transmit some bundles. Remember that the EHS Router is designed for use with the IPN Scheme so all of the EIDs must be of the IPN scheme format.

On node1, run the receiving application (note that “-s” in this case is for statistics and not source node):

```
> dtnrecv -s -n 10000 ipn:1.28
```

Then on node2, run the transmitter:

```
> dtnsource -s ipn:2.7 -d ipn:1.28 -b 1024 -n 10000
```

You should see the bundle statistics updating in the EHS Router windows as well as in the dtnrecv application.
13. Delay Tolerant Payload Conditioning

Delay-Tolerant Payload Conditioning (DTPC) is an application service that utilizes the Delay-Tolerant Networking (DTN) Bundle Protocol (BP) in order to provide DTN end-to-end services that are similar to those provided by “transport” protocols such as the Transmission Control Protocol (TCP). A copy of the specification (CCSDS 734.2-R-3 Appendix E) is currently available for download at:

DTPC provides two services which can be performed individually or in combination. The first service is the Transport Service which provides delivery in transmission order without duplication or omissions except under extreme conditions. The second service is the Optimization Service which provides aggregation of Application Data Items (ADI) into larger payloads to reduce BP overhead. The Optimization Service also has provisions that allow an application to review the ADIs prior to transmission with the opportunity to remove redundant ADIs to further reduce waste of bandwidth.

These DTPC notes are adapted from the preview version that was released for the TopCoder competition to develop test procedures to validate the DTPC implementations for CCSDS.

Building DTN2 with DTPC

Add the --with-dtpc option to the configure command:

```
> ./configure --disable-ecl --disable-edp --with-odbc=no --with-oasys=../oasys
--with-dtpc
> make
> sudo make install
```

Sample DTPC configuration files

The sample configuration files are for running the DTN2 and ION nodes all on the same system

- You can adjust the IP addresses and ports as desired if running on different systems

Node 1 - DTN2

The scripts and configuration files for running a DTN2 node as a DTPC Endpoint ID ipn:1.129 are in the <path>/configs/dtpc/node1_dtn2 directory.

The run script, run_dtn2.sh, assumes you are running it locally so you will have to set directory there before running it.

- Usage: run_dtn2.sh <node_number>
  - Note that the run script includes the "-t" option which clears the database on each run
  - Enter the "exit" command in the DTN2 console prompt when ready to shut it down
- Using the default Berkely DB for persistent storage results in a lot of transaction logs which can quickly fill up a partition if you are transmitting a lot of bundles. I included a script you can run to keep the log files under control – usage: clean_logs.sh <node_number>

The configuration file, dtpc_node1.cfg, is used to initialize node 1. Most of the pertinent configuration elements are listed here:

- TCP and UDP Convergence Layer Port = 4551
- Bundle Protocol Endpoint ID: dtn://node1.dtn
- DTPC Endpoint ID: ipn:1.129
Applications use API Port: **5011**
DTN2 Console Port: **5051**
Database location: `/tmp/node1_dtn2`
Log file: `/tmp/node1_dtn2/dtn.log`
Search for **ion_node3** in the config file to update the connection info to the ION node (Port **3113**) if not running on the same system
DTPC configuration is at the end of the file
param set early_deletion is set to false to provide some time to list and view the bundles
- Bundles will not be deleted until their lifetime expires allowing use of some BP commands:
  - bundle list command shows list of bundles including id, source and destination EIDs and payload size
    - This is handy for seeing if retransmits are properly sent or received
  - bundle info <bundle_id> will show all of the metadata for a bundle
  - bundle dump <bundle_id> will show the payload for a bundle

**Node 3 - ION**
The scripts and configuration files for running an ION node as a DTPC Endpoint ID ipn:3.129 are in the `<path>/configs/dtpc/node3_ion` directory.

The run script, **run_ion.sh**, assumes you are running it locally so you will have to set directory there before running it also.
- Usage: `source ./run_ion.sh`
  - You need to "source" the script so that the variable **ION_NODE_LIST_DIR** gets exported out
  - If you run apps in another window you will need to set default to this directory and also set **ION_NODE_LIST_DIR** to point here
  - Log files: `<path>/configs/dtpc/node3_ion/ion.log` & `node3.stdout`
  - Run the killm command when ready to shut it down and clean up shared memory

The run_ion.sh script runs the **ionstart** script which starts each admin program with its config file. Most of the pertinent configuration elements are listed here:
- File **node.bprc** file defines
  - TCP Convergence Layer induct Port = **3113**
  - TCP Convergence Layer outduct to Port = **4551** (node1)
  - Bundle Protocol Endpoint ID: **ipn:3.0**
  - DTPC Endpoint ID: **ipn:3.129** (receive service) **ipn:3.128** (send service)
- File **node.ipnrc** defines
  - ipn:1.* routes to tcp/localhost:4551 (node1)
- File **node.dtn2rc** defines
  - dtn://node1.dtn/* routes to tcp/localhost:4551 (node1)
    - ION DTPC does not support dtn:// scheme EIDs for DTPC but this is needed for BP custody signals
- File **node.dtpcrc** defines
  - Transmission profiles which must match those defined in the DTN2 configuration files
DTN2 DTPC Configuration and Applications

The DTPC Endpoint EID must be set in the DTN2 configuration file so that it is known at startup. The Transmission Profiles and Topic IDs should be set up in the DTN2 configuration file but can be added manually from the DTN2 console after it is running.

Here is the list of commands that can be used to configure and status the DTPC component of the DTN2 BP node:

- help dtpc
  - list all DTPC commands

- dtpc local_eid <eid>
  - view or set the EID for the DTPC application

- dtpc add_profile <profile id> [arg1=val1 arg2=val2 argN=valN...]
  - add a transmission profile
  - valid options:
    - The Profile ID number
    - <args>:
      - retrans=<number> - retransmission limit (default=0)
      - agg_size=<number> - aggregation size limit in bytes (default=1000)
      - agg_time=<number> - aggregation time limit in seconds (default=60)
      - custody=[true|false] - request custody transfer (default=false)
      - lifetime=<number> - lifetime seconds for bundles (default=864000)
      - replyto=<EndpointID> - ReplyTo EID for bundles (default=dtn:none)
      - priority=[b|n|e|r] - priority for bundles [bulk,normal,expedited or reserved] (default=bulk)
      - ecos=<number> - extended class of service value (default=0)
      - rpt_rcpt=[true|false] - request bundle reception report (default=false)
      - rpt_acpt=[true|false] - request custody acceptance report (default=false)
      - rpt_fwrd=[true|false] - request bundle forward report (default=false)
      - rpt_dlvr=[true|false] - request bundle delivery report (default=false)
      - rpt_dele=[true|false] - request bundle deletion report (default=false)

- dtpc del_profile <profile id>
  - delete a transmission profile
  - valid options:
    - <profile id> - The Profile ID number

- dtpc list_profiles
  - list all of the transmission profiles

- dtpc add_topic <topic id> ["quoted description"]
  - add a DTPC topic
  - valid options:
    - <topic id> - The Topic ID number
    - "quoted description" - optional description – enclose in quotes if it has embedded spaces

- dtpc del_topic <topic id>
  - delete a DTPC topic
  - valid options:
    - <topic id> - The Topic ID number
- dtpc list_topics
  - list all of the DTPC Topics

- dtpc set require_predefined_topics
  - Require predefined topics or allow ad hoc topics - (default is false)

- dtpc set restrict_send
  - Restrict send of a topic to the registered client - (default is false)

- dtpc set ipn_receive_service_number <svc num>
  - IPN DTPC receive service number - (default is 129)

- dtpc set ipn_transmit_service_number <svc num>
  - IPN DTPC transmit service number - (default is 128)

### DTN2 DTPC Applications

There are two sample utility applications provided with the DTN2 source to exercise the DTPC functionality. They are dtpc_send and dtpc_recv and the source code for both are located in `<path_dtn2_src>/apps/dtpc_apps/`.

- dtpc_send [-A api_IP][-B api_port][-c count][-i interval (ms)] -p <profile> -t <topic> eid
  - Where:
    - A api_IP = IP address of the BP node hosting the DTPC application
    - B api_port = API Port number of the BP node hosting the DTPC application
    - c count = Number of Application Data Items to send
    - i interval = Millisecond interval between sends
    - p profile = Transmission Profile ID (required)
    - t topic = Topic ID (required)
    - eid = Destination EID

  - Transmits <count> ADIs for <topic> to destination <eid> using <profile> with an <interval> sleep between each

  - Application Data Items are currently fixed 100 byte null terminated C strings filled with an ASCII character
    - Feel free to change as needed: `<path_to_source>/apps/dtpc_apps/dtpc_send.c`

  - dtpc_send does not register the DTPC Topic ID (see dtpc_recv)
```bash
```

- **Where:**
  - `-A api_IP` = IP address of the BP node hosting the DTPC application
  - `-B api_Port` = API Port number of the BP node hosting the DTPC application
  - `-c count` = Number of Application Data Items to receive before quitting
  - `-q` = Quiet mode - suppress output of data bytes (recommended if ION sending 64K ADIs)
  - `-e` = use the elision function (see below)
  - `-t topic` = Topic ID (required)

- dtpc_recv prints out a message each time an ADI is received and if quiet mode is not in effect then it also prints out all of its bytes

- If the elision function is enabled then whenever a dtpc_send app sends an ADI for this topic, the elision function is invoked. See the source file, dtpc_recv.c, for the example implementation. This elision function adds a fifth ADI when it is invoked with a set of 4 aggregated ADIs and when it is invoked with 7 aggregated ADIs it will modify the 6th ADI and remove the 7th.

**ION DTPC Applications**

There are also two sample utility applications provided with the ION source to exercise the DTPC functionality. They are `dtpcsend` and `dtpcrecv` which have slightly different names so they can coexist if they are installed on the same system.

```bash
  dtpcsend
  - Use command "man dtpcsend" for a description
  - The elision function is built into this application and it deletes any duplicate ADIs that it detects
  - You can use the random payload size option to aggregate multiple ADI into a PDU
  - If sending multiple ADIs of a fixed size it will always pare it down to a single ADI
    - During my testing, I found it handy to add an option to disable the elision function
```

```bash
  dtpcrecv
  - Use command "man dtpcrecv" for a description
```

Note that only one instance of dtpcsend or dtpcrecv can be running for a single topic as they both try to register the topic.

**DTN2 and ION DTPC Interoperability Testing Notes**

- If a bundle (transmitted Data PDU) is in local custody when it expires then a deletion report is sent to the DTPC application. It eventually fails a validity check and gets dropped but need to see if there is a way to detect a report message or other non-DTPC message.

- The run_dtn2.sh script clears the database so if you restart the DTN2 node you will need to also restart the ION node (using the killm utility to destroy the persistent memory) to keep the sequence counters in sync.

- The ION DTPC implementation tested only supports DTPC Endpoint IDs of the IPN scheme
  - Incoming PDUs can only be received on service number 129 (ipn:3.129 in this example configuration)
The source Endpoint ID service number does not matter for incoming ACKs - effectively, only its node number is used

Outgoing PDUs are sent to the destination_endpoint specified on the dtpcsend command line regardless of the service number

Outgoing PDUs (Data and ACKs) have a source EID with a service number of 128 (ipn:3.128 in this example configuration)

This DTN2 DTPC implementation supports Endpoint IDs of both the IPN scheme and the DTN scheme

- Incoming PDUs can only be received on the configured DTPC EID
  - for complete compatibility with the ION implementation, it must be an IPN scheme with service number 129 (ipn:1.129 in this example configuration)
- When ACKs are received, it attempts to find an unacked PDU sent to the source EID of the ACK. If no unacked PDU is found and the source EID is of the IPN scheme then it replaces the service number with 129 and looks for an unacked PDU again.
- Outgoing Data PDUs are sent to the <eid> specified on the dtpc_send command line with a source EID of the configured DTPC Endpoint EID (ipn:1.129 in this configuration)
- Outgoing ACKs to IPN scheme destination EIDs, override the service number to 129 if the original Data PDU was received from a 128 service number

The specification does not directly address the disposition of ADIs received for a topic when there is not a registration

- This DTN2 DTPC implementation will retain the ADIs in queue for delivery until they expire
- I’m not sure what the ION implementation does in that case

The specification does not address coordinated resetting of the sequence counter if a node has a catastrophic error and cannot recover its last used counter

- I believe the ION DTPC implementation supports some sort of reset detection on the receiving side if the transmitter encounters said catastrophic error

**Sending from DTN2 to DTN2 example**

You will need several windows to run these sample sessions.

- Start the DTN2 DTPC node1 in one window
  - cd <path>/configs/dtpc/node1_dtn2
  - ./run_dtn2.sh 1
    - You should see this output and DTN2 command prompt:
      - configuring for node1
      - node1 dtn%

- Start the DTN2 DTPC receiver app in a second window
  - cd <path>/configs/dtpc/node1_dtn2
    - Starting directory is not really important
  - dtpc_recv -A 127.0.0.1 -B 5011 -t 1
    - You should see this output:
      - dtpc_register succeeded - wait for application data items...
      - looping forever to receive Application Data Items
MSFC DTN2 Patches 2016-02-29

○ Start the DTN2 DTPC node2 in a third window
  ➢ cd <path>/configs/dtpc/node1_dtn2
  ➢ ./run_dtn2.sh 2
    ○ You should see this output and DTN2 command prompt:
      configuring for node2
      node2 dtn%

○ Start the node2 DTN2 DTPC transmit app running in a fourth window
  ➢ dtpc_send -B 5012 -c 15 -p 1 -t 1 ipn:1.129
    • After completion you will see output:
      dtpc_send (pid 22417) exiting: 15 ADIs transmitted, 1500 total bytes

In the second window running the node1 dtpc_recv application, you should see the print out of the 15 received Application Data Items. Remember there is aggregation taking places so they will not start printing out immediately.

Sending from ION to DTN2 example
You will need several windows to run these sample sessions.

○ Start the DTN2 DTPC node1 in one window
  ➢ cd <path>/configs/dtpc/node1_dtn2
  ➢ ./run_dtn2.sh 1
    ○ You should see this output and DTN2 command prompt:
      configuring for node1
      node1 dtn%

○ Start the DTN2 DTPC receiver app in a different window
  ➢ cd <path>/configs/dtpc/node1_dtn2
    ○ Starting directory is not really important
  ➢ dtpc_recv -A 127.0.0.1 -B 5011 -t 1 -e
    ○ You should see this output:
      dtpc_register succeeded - wait for application data items...
      looping forever to receive Application Data Items

○ Start the ION DTPC node3 in a different window
  ➢ cd <path>/configs/dtpc/node3_ion
  ➢ source run_ion.sh
    ○ You should see output similar to this:
      [2013/09/13-20:50:06] [i] rfxclock is running.
      [2013/09/13-20:50:06] [i] No congestion forecast; horizon reached.
      [2013/09/13-20:50:06] [i] ionwarn finished.
      [2013/09/13-20:50:07] [i] No congestion forecast; horizon reached.
      [2013/09/13-20:50:07] [i] ionwarn finished.
      [2013/09/13-20:50:08] [i] No congestion forecast; horizon reached.
      [2013/09/13-20:50:08] [i] ionwarn finished.
      [2013/09/13-20:50:09] [i] Bundle security is enabled.
      [2013/09/13-20:50:10] [i] ipnadminep is running.
      [2013/09/13-20:50:10] [i] tcpclio is running, spec=[127.0.0.1:4551].
      [2013/09/13-20:50:10] [i] ipnfw is running.
Start the ION DTPC transmit app in the same window already in the node3_ion directory

- dtpcsend 1 100000 30 1 1 ipn:1.129
  - sends 1 ADI of 30 bytes when the aggregation timer expires and you will see output:
    - Stopping dtpcsend.
    - Time (seconds): 0
    - Total payloads: 1
    - Total bytes: 30
    - Interval is too short to measure rate.

- In the DTN2 DTPC receiver app you should see output:
  - ADI #1 - Topic 1 from ipn:3.128 - 30 bytes:
    - Payloads received: 0
    - Payloads received: 0

- In the DTN2 DTPC node1 window, enter the bundle list command before the lifetime and you will see this output:
  - node1 dtn% bundle list
    - All Bundles (2):
      - 0 : ipn:3.128 -> ipn:1.129 length 36
      - 1 : ipn:1.129 -> ipn:3.129 length 3
  - ION node sent the Data PDU in bundle 0
  - DTN2 node replied with an ACK in bundle 1

Sending from DTN2 to ION example

This section is a follow on to the previous one so keep all of the applications in the above windows running and continue with these steps:

- Start the ION DTPC receiver app in same window already in the node3_ion directory
  - dtpcreceive 1
    - You will start to see outputs detailing how many payloads have been received:
      - Payloads received: 0
      - Payloads received: 0

- Start the DTN2 DTPC transmit app running in a new window
  - dtpc_send -A 127.0.0.1 -B 5011 -c 15 -p 1 -t 1 ipn:3.129
    - After completion you will see output:
      - dtpc_send (pid 22417) exiting: 15 ADIs transmitted, 1500 total bytes

- In the DTN2 DTPC receiver app window you should see the invocations of the elision function similar to:
  - dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 1
    - ADI[1] len = 100 data = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
    - ADI[2] len = 100 data = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 3
ADI[1] len = 100 data = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ADI[2] len = 100 data = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
ADI[3] len = 100 data = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 5
ADI[1] len = 100 data = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ADI[2] len = 100 data = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
ADI[3] len = 100 data = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
ADI[4] len = 100 data = DDDDDDDDDDDDDDDDDDDDDDDDDDDDD
ADI[5] len = 32 data = Elision Insertion: 1

- Notice above that 4 ADIs were aggregated and the elision function added a fifth

dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 6
ADI[1] len = 100 data = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ADI[2] len = 100 data = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
ADI[3] len = 100 data = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
ADI[4] len = 100 data = DDDDDDDDDDDDDDDDDDDDDDDDDDDDD
ADI[5] len = 32 data = Elision Insertion: 1
ADI[6] len = 110 data = EEEEEEEE

- Notice above that the elision function received 7 aggregated ADIs and modified the 6th while dropping the 7th which continues below until the aggregation timer expires and sends the payload

dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 6
ADI[1] len = 100 data = AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
ADI[2] len = 100 data = BBBBBBBBBBBBBBBBBBBBBBBBBBBBBB
ADI[3] len = 100 data = CCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
ADI[4] len = 100 data = DDDDDDDDDDDDDDDDDDDDDDDDDDDDD
ADI[5] len = 32 data = Elision Insertion: 1
ADI[6] len = 111 data = EEEEEEEE

- Notice that the first payload was sent and new aggregation begins again with a single ADI:

dtpc_recv - elision function - topic: 1, dest: ipn:3.129, profile: 1, num data items: 1
ADI[1] len = 100 data = KKKKKKKKKKKKKKKKKKKKKKKKKKKKK
Enter <Ctrl-C> to terminate and you will see a summary:

dtpc_recv (pid 23301) exiting: 1 ADIs received, 30 total bytes - Elision function calls: 15

dtpc_unregister succeeded - exiting

- In the ION DTPC receiver app window you should see that payloads have been received:
  - Payloads received: 0
  - Payloads received: 0
  - Payloads received: 6
  - Payloads received: 6
  - Payloads received: 12
  - Payloads received: 12
  - Payloads received: 12

- In the DTN2 DTPC node1 window, enter the bundle list command before the lifetime expires and you will see this output:
  - node1 dtn% bundle list
    All Bundles (4):
    2 : ipn:1.129 -> ipn:3.129 length 557
    3 : ipn:3.128 -> ipn:1.129 length 3
    4 : ipn:1.129 -> ipn:3.129 length 543
    5 : ipn:3.128 -> ipn:1.129 length 3
  - DTN2 node sent the Data PDUs in bundles 2 and 4
  - ION node replied with ACKs in bundles 3 and 5
14. **Wireshark Dissectors**

Updates to the DTN and LTP dissectors are included in this release and will also be submitted to wireshark.org. Hopefully, they will include them in a future release. In the meantime you can build your own if you would like. Extract the files from the archive that you downloaded:

```bash
    tar -xf wireshark_dissectors_mfs-33289-1.tar.gz
```

There are three updated files that you can apply to the latest 2.1 version of wireshark:

```bash
    wireshark_dissectors/packet-dtn.c
    wireshark_dissectors/packet-dtn.h
    wireshark_dissectors/packet-ltp.c
```

Follow the Wireshark Developer’s Guide located at: [https://www.wireshark.org/docs/wsdg_html_chunked/](https://www.wireshark.org/docs/wsdg_html_chunked/)

After downloading the source code, copy the three provided files to the wireshark/epan/dissectors/ directory to replace the current ones and build per the instructions for your operating system.

The LTP updates include support for the CCSDS specification including expert warnings when session or sequence numbers exceed the 32 bit restriction. All bundles in a data segment are viewable. The Cancel ACK segment no longer displays as a protocol error.

The DTN updates provide better support for the IPN Scheme and Compressed Bundle Header Encoding and display the bundle blocks in the proper order instead of putting the Primary Block at the end. An entry for the Payload Data itself was also added to highlight those bytes. The TCP Convergence Layer dissection has been improved to show multiple bundles in a segment.
Appendix A. NASA Open Source Agreement version 1.3

NASA OPEN SOURCE AGREEMENT VERSION 1.3

THIS OPEN SOURCE AGREEMENT ("AGREEMENT") DEFINES THE RIGHTS OF USE, REPRODUCTION, DISTRIBUTION, MODIFICATION AND REDISTRIBUTION OF CERTAIN COMPUTER SOFTWARE ORIGINALLY RELEASED BY THE UNITED STATES GOVERNMENT AS REPRESENTED BY THE GOVERNMENT AGENCY LISTED BELOW ("GOVERNMENT AGENCY"). THE UNITED STATES GOVERNMENT, AS REPRESENTED BY GOVERNMENT AGENCY, IS AN INTENDED THIRD-PARTY BENEFICIARY OF ALL SUBSEQUENT DISTRIBUTIONS OR REDISTRIBUTIONS OF THE SUBJECT SOFTWARE. ANYONE WHO USES, REPRODUCES, DISTRIBUTES, MODIFIES OR REDISTRIBUTES THE SUBJECT SOFTWARE, AS DEFINED HEREIN, OR ANY PART THEREOF, IS, BY THAT ACTION, ACCEPTING IN FULL THE RESPONSIBILITIES AND OBLIGATIONS CONTAINED IN THIS AGREEMENT.

Government Agency: NASA Marshall Space Flight Center
Government Agency Original Software Designation: MFS-33289-1
Government Agency Original Software Title: DTN2 Software Updates version 1
User Registration Requested. Please email: david.a.zoller@nasa.gov
Government Agency Point of Contact for Original Software: David Zoller
david.a.zoller@nasa.gov

1. DEFINITIONS

A. "Contributor" means Government Agency, as the developer of the Original Software, and any entity that makes a Modification.
B. "Covered Patents" mean patent claims licensable by a Contributor that are necessarily infringed by the use or sale of its Modification alone or when combined with the Subject Software.
C. "Display" means the showing of a copy of the Subject Software, either directly or by means of an image, or any other device.
D. "Distribution" means conveyance or transfer of the Subject Software, regardless of means, to another.
E. "Larger Work" means computer software that combines Subject Software, or portions thereof, with software separate from the Subject Software that is not governed by the terms of this Agreement.
F. "Modification" means any alteration of, including addition to or deletion from, the substance or structure of either the Original Software or Subject Software, and includes derivative works, as that term is defined in the Copyright Statute, 17 USC 101. However, the act of including Subject Software as part of a Larger Work does not in and of itself constitute a Modification.
G. "Original Software" means the computer software first released under this Agreement by Government Agency with Government Agency designation MFS-33289-1 and entitled DTN2 Software Updates, including source code, object code and accompanying documentation, if any.
H. "Recipient" means anyone who acquires the Subject Software under this Agreement, including all Contributors.
I. "Redistribution" means Distribution of the Subject Software after a Modification has been made.
J. "Reproduction" means the making of a counterpart, image or copy of the Subject Software.
K. "Sale" means the exchange of the Subject Software for money or equivalent value.
L. "Subject Software" means the Original Software, Modifications, or any respective parts thereof.

M. "Use" means the application or employment of the Subject Software for any purpose.

2. GRANT OF RIGHTS

A. Under Non-Patent Rights: Subject to the terms and conditions of this Agreement, each Contributor, with respect to its own contribution to the Subject Software, hereby grants to each Recipient a non-exclusive, world-wide, royalty-free license to engage in the following activities pertaining to the Subject Software:

1. Use
2. Distribution
3. Reproduction
4. Modification
5. Redistribution
6. Display

B. Under Patent Rights: Subject to the terms and conditions of this Agreement, each Contributor, with respect to its own contribution to the Subject Software, hereby grants to each Recipient under Covered Patents a non-exclusive, world-wide, royalty-free license to engage in the following activities pertaining to the Subject Software:

1. Use
2. Distribution
3. Reproduction
4. Sale
5. Offer for Sale

C. The rights granted under Paragraph B. also apply to the combination of a Contributor's Modification and the Subject Software if, at the time the Modification is added by the Contributor, the addition of such Modification causes the combination to be covered by the Covered Patents. It does not apply to any other combinations that include a Modification.

D. The rights granted in Paragraphs A. and B. allow the Recipient to sublicense those same rights. Such sublicense must be under the same terms and conditions of this Agreement.

3. OBLIGATIONS OF RECIPIENT

A. Distribution or Redistribution of the Subject Software must be made under this Agreement except for additions covered under paragraph 3H.

1. Whenever a Recipient distributes or redistributes the Subject Software, a copy of this Agreement must be included with each copy of the Subject Software; and
2. If Recipient distributes or redistributes the Subject Software in any form other than source code, Recipient must also make the source code freely available, and must provide with each copy of the Subject Software information on how to obtain the source code in a reasonable manner on or through a medium
customarily used for software exchange.

B. Each Recipient must ensure that the following copyright notice appears prominently in the Subject Software:


C. Each Contributor must characterize its alteration of the Subject Software as a Modification and must identify itself as the originator of its Modification in a manner that reasonably allows subsequent Recipients to identify the originator of the Modification. In fulfillment of these requirements, Contributor must include a file (e.g., a change log file) that describes the alterations made and the date of the alterations, identifies Contributor as originator of the alterations, and consents to characterization of the alterations as a Modification, for example, by including a statement that the Modification is derived, directly or indirectly, from Original Software provided by Government Agency. Once consent is granted, it may not thereafter be revoked.

D. A Contributor may add its own copyright notice to the Subject Software. Once a copyright notice has been added to the Subject Software, a Recipient may not remove it without the express permission of the Contributor who added the notice.

E. A Recipient may not make any representation in the Subject Software or in any promotional, advertising or other material that may be construed as an endorsement by Government Agency or by any prior Recipient of any product or service provided by Recipient, or that may seek to obtain commercial advantage by the fact of Government Agency's or a prior Recipient's participation in this Agreement.

F. In an effort to track usage and maintain accurate records of the Subject Software, each Recipient, upon receipt of the Subject Software, is requested to provide Government Agency, by e-mail to the Government Agency Point of Contact listed in clause 5.F., the following information: Name and Email Address. Recipient's name and personal information shall be used for statistical purposes only. Once a Recipient makes a Modification available, it is requested that the Recipient inform Government Agency, by e-mail to the Government Agency Point of Contact listed in clause 5.F., how to access the Modification.

G. Each Contributor represents that its Modification is believed to be Contributor's original creation and does not violate any existing agreements, regulations, statutes or rules, and further that Contributor has sufficient rights to grant the rights conveyed by this Agreement.

H. A Recipient may choose to offer, and to charge a fee for, warranty, support, indemnity and/or liability obligations to one or more other Recipients of the Subject Software. A Recipient may do so, however, only on its own behalf and not on behalf of Government Agency or any other Recipient. Such a Recipient must make it absolutely clear that any such warranty, support, indemnity and/or liability obligation is offered by that Recipient alone. Further, such Recipient agrees to indemnify Government Agency and every other Recipient for any liability incurred by them as a result of warranty, support, indemnity
and/or liability offered by such Recipient.

I. A Recipient may create a Larger Work by combining Subject Software with separate software not governed by the terms of this agreement and distribute the Larger Work as a single product. In such case, the Recipient must make sure Subject Software, or portions thereof, included in the Larger Work is subject to this Agreement.

J. Notwithstanding any provisions contained herein, Recipient is hereby put on notice that export of any goods or technical data from the United States may require some form of export license from the U.S. Government. Failure to obtain necessary export licenses may result in criminal liability under U.S. laws. Government Agency neither represents that a license shall not be required nor that, if required, it shall be issued. Nothing granted herein provides any such export license.

4. DISCLAIMER OF WARRANTIES AND LIABILITIES; WAIVER AND INDEMNIFICATION

A. No Warranty: THE SUBJECT SOFTWARE IS PROVIDED "AS IS" WITHOUT ANY WARRANTY OF ANY KIND, EITHER EXPRESSED, IMPLIED, OR STATUTORY, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL CONFORM TO SPECIFICATIONS, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, OR FREEDOM FROM INFRINGEMENT, ANY WARRANTY THAT THE SUBJECT SOFTWARE WILL BE ERROR FREE, OR ANY WARRANTY THAT DOCUMENTATION, IF PROVIDED, WILL CONFORM TO THE SUBJECT SOFTWARE. THIS AGREEMENT DOES NOT, IN ANY MANNER, CONSTITUTE AN ENDORSEMENT BY GOVERNMENT AGENCY OR ANY PRIOR RECIPIENT OF ANY RESULTS, RESULTING DESIGNS, HARDWARE, SOFTWARE PRODUCTS OR ANY OTHER APPLICATIONS RESULTING FROM USE OF THE SUBJECT SOFTWARE. FURTHER, GOVERNMENT AGENCY DISCLAIMS ALL WARRANTIES AND LIABILITIES REGARDING THIRD-PARTY SOFTWARE, IF PRESENT IN THE ORIGINAL SOFTWARE, AND DISTRIBUTES IT "AS IS."

B. Waiver and Indemnity: RECIPIENT AGREES TO WAIVE ANY AND ALL CLAIMS AGAINST THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT. IF RECIPIENT'S USE OF THE SUBJECT SOFTWARE RESULTS IN ANY LIABILITIES, DEMANDS, DAMAGES, EXPENSES OR LOSSES ARISING FROM SUCH USE, INCLUDING ANY DAMAGES FROM PRODUCTS BASED ON, OR RESULTING FROM, RECIPIENT'S USE OF THE SUBJECT SOFTWARE, RECIPIENT SHALL INDEMNIFY AND HOLD HARMLESS THE UNITED STATES GOVERNMENT, ITS CONTRACTORS AND SUBCONTRACTORS, AS WELL AS ANY PRIOR RECIPIENT, TO THE EXTENT PERMITTED BY LAW. RECIPIENT'S SOLE REMEDY FOR ANY SUCH MATTER SHALL BE THE IMMEDIATE, UNILATERAL TERMINATION OF THIS AGREEMENT.
5. GENERAL TERMS

A. Termination: This Agreement and the rights granted hereunder will terminate automatically if a Recipient fails to comply with these terms and conditions, and fails to cure such noncompliance within thirty (30) days of becoming aware of such noncompliance. Upon termination, a Recipient agrees to immediately cease use and distribution of the Subject Software. All sublicenses to the Subject Software properly granted by the breaching Recipient shall survive any such termination of this Agreement.

B. Severability: If any provision of this Agreement is invalid or unenforceable under applicable law, it shall not affect the validity or enforceability of the remainder of the terms of this Agreement.

C. Applicable Law: This Agreement shall be subject to United States federal law only for all purposes, including, but not limited to, determining the validity of this Agreement, the meaning of its provisions and the rights, obligations and remedies of the parties.

D. Entire Understanding: This Agreement constitutes the entire understanding and agreement of the parties relating to release of the Subject Software and may not be superseded, modified or amended except by further written agreement duly executed by the parties.

E. Binding Authority: By accepting and using the Subject Software under this Agreement, a Recipient affirms its authority to bind the Recipient to all terms and conditions of this Agreement and that that Recipient hereby agrees to all terms and conditions herein.

F. Point of Contact: Any Recipient contact with Government Agency is to be directed to the designated representative as follows: david.a.zoller@nasa.gov