**DZ Notes on the DTNME BARD Beta Release**

**DTNME 1.2.0-Beta**

**10/31/2022**

Contents

[BARD Overview 3](#_Toc118106936)

[Build Instructions 4](#_Toc118106937)

[New BARD Commands 5](#_Toc118106938)

[add\_quota subcommand 7](#_Toc118106939)

[del\_quota subcommand 8](#_Toc118106940)

[unlimited\_quota subcommand 8](#_Toc118106941)

[BARD Quota Persistence 9](#_Toc118106942)

[force\_restage subcommand 9](#_Toc118106943)

[quotas subcommand 10](#_Toc118106944)

[usage subcommand 11](#_Toc118106945)

[dump subcommand 12](#_Toc118106946)

[rescan subcommand 13](#_Toc118106947)

[del\_all\_restaged\_bundles subcommand 13](#_Toc118106948)

[del\_restaged\_bundles subcommand 13](#_Toc118106949)

[reload subcommand 13](#_Toc118106950)

[Reload\_all subcommand 13](#_Toc118106951)

[Restage Convergence Layer Configuration 14](#_Toc118106952)

[File transfer utility (deliver\_me) 16](#_Toc118106953)

[Miscellaneous updates 19](#_Toc118106954)

# BARD Overview

This version of DTNME implements a capability known as Bundle Architectural Restaging Daemon (BARD). It allows bundles to be offloaded from Internal Storage to an External Storage based on quotas of count or bytes for either source or destination Endpoint IDs (EID). As applicable bundles are transmitted from Internal Storage, the offloaded bundles can be automatically reloaded from the External Storage to keep a steady stream flowing to the destination.

For example, you can configure a quota such that only 1000 bundles or 200 MB of payloads from ipn:24[.\*] are kept in Internal Storage and the rest get shunted off to External Storage. When transmission of the 1000 bundles brings the count down to 200 (20% of the configured quota) and if configured for auto-reload then bundles will begin reloading from External Storage until hitting the quota again.

Quotas can also be applied to the External Storage. If the quota for External Storage is reached then, if possible, acceptance of additional bundles of that type will be put on hold or refused. Use of the External Storage is optional and if not specified then if an Internal Storage quota is exceeded the bundles will be rejected if possible.

Currently, only the TCP and STCP Convergence Layers support temporary stoppage of the intake of bundles from the link when the primary and secondary quotas (if any) are reached. The UDP Convergence Layer will delete bundles that are over quota. The LTPUDP Convergence Layer always accepts bundles even if over quota because it acknowledges LTP sessions prior to extracting bundles and would not have the capability to reject only some of the bundles in an LTP session. If the entire Internal Storage capacity fills up then the LTPUDP Convergence Layer will begin rejecting all incoming LTP sessions to prevent loss of bundles.

A new Restage Convergence Layer (CL) must be configured to manage each External Storage location. Multiple instances are allowed and can optionally be configured to act as part of a pool of storage locations. For instance, you could use two Restage CLs to treat two 1GB drives as one large 2GB shared External Storage. Alternatively, you could set up one Restage CL as a dedicated 500MB for a specific set of bundles and two other Restage CLs to act as a pool of 1.5GB for all other bundles.

When a BARD quota is configured, it specifies the name of the Restage CL instance to which those bundles should be sent for offloading. If the instance is configured as a pool then BARD will first check to see if its storage has room for the bundle and if not then it will try the other Restage CLs that are configured as part of the pool. Note that there is only support for a single pool of storage locations.

# Build Instructions

The latest NASA version of the DTNME 1.0.2-BETA source code can be downloaded from the GitHub site:

<https://github.com/nasa/DTNME/tree/TBD>

(As of 10/31/2022, the latest version is: **TBD**)

From the README file:

Dependencies:

libdb-dev g++ automake autotools-dev tk tk-dev tcl tcl-dev

1. Compiling and linking using the gcc compiler:

Run the script:

**./step\_1\_gcc\_dtnme.sh [<make\_executable\_name>] [<#jobs>]**

<make\_executable\_name> can be specified to build with an alternative to the default “make” executable.

<#jobs> can be replaced with the number of jobs you would like to use in conjunction with the make -j option to try to speed up the build process.

You can edit this file and configuration preferences to the configure statement at or near line 95. There are two locations where a “./configure” script is executed and it is the second one that configures DTNME and is the one that you would most likely want to change.

1. Installing:

Use this command to install the DTNME executables to the default or configured location:

**sudo make install**

Or use this script to create a sample test setup for a specified [IPN] node number:

**./step\_2\_install\_dtnme.sh <installation\_directory> <node\_number>**

This uses the <installation\_directory> as the top level storage location and

* Installs the executable in a <installation\_directory>/bin subdirectory
* Creates a run\_dtnme.sh script
* Creates a sample configuration file for the specified <node\_number>:

<installation\_directory>/dtnme\_daemon.cfg

* Initializes a database storage location in <installation\_directory>/node\_<node\_number>

# New BARD Commands

A new “bard” command has been implemented to configure and manage the BARD capability.

For a full list of available commands, issue the “help” command in the DTNME console and then for details for a particular command issue “help <command>”.

Below is the help provided for the “help bard” command:

dtnme% help bard

bard add\_quota <quota type> <naming scheme> <node number/name> <internal bundles>

<internal bytes>

[<restage link name> <auto reload> <external bundles> <external bytes>]

add or update a node quota entry; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

<internal bundles> - max number of bundles allowed in internal storage (0=no max)

<internal bytes> - max number of payload bytes allowed in internal storage (0=no max)

(if no other parameters are provided then bundle refusal will be attempted)

<restage link name> - name of the restage CL to use for offloading to external storage

<auto reload> - whether to reload bundles when usage drops to 20%

('true' or 'false')

<external bundles> - max number of bundles allowed in external storage (0=no max)

<external bytes> - max number of payload bytes allowed in external storage (0=no max)

(max number values may be specified exactly or with a magnitude character of K, M, G or T;

where K = x1000, M = x1000000, etc. example: 12G = 12000000000)

bard del\_quota <quota type> <naming scheme> <node number/name>

delete a node quota entry; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

bard unlimited\_quota <quota type> <naming scheme> <node number/name>

clear limits of a node quota entry and leave it in the database; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

(forces override of a quota in a startup config file upon restart)

bard force\_restage <quota type> <naming scheme> <node number/name>

force over-quota bundles of specified type to be restaged if quota is configured for

restaging; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

bard quotas [exact]

list all quota definitions using magnitude values or exact values if specified

bard usage [exact]

list usage data for all nodes with and without quotas using magnitude values or exact values if specified

bard dump

list all usage and reserved data for all nodes with and without quota definitions

bard rescan

rescan external storage (best done while DTN daemon is mostly idle)

bard del\_all\_restaged\_bundles

delete all restaged bundles from all restage locations

bard del\_restaged\_bundles <quota type> <naming scheme> <node number/name>

delete restaged bundles for a specific quota type; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

bard reload <quota type> <naming scheme> <node number/name> [<new expiration secs> <new dest EID>]

attempt to reload restaged bundles for a specific quota type; where:

<quota type> - 'dst' = destination node or 'src' = source node

<naming scheme> - 'ipn', 'imc' or 'dtn'

<node number/name> - 'ipn' and 'imc' require a number and 'dtn' requires a name

optional:

<new expiration secs> - adjust expiration if needed to provide a minimum time to live

<new dest EID> - changes the bundle destination EID

bard reload\_all [<new expiration secs>]

attempt to reload all restaged bundles; where:

<new expiration secs> - adjust expiration if needed to provide a minimum time to live

## add\_quota subcommand

Usage of the BARD capability requires configuration of quotas using the “bard add\_quota” command:

bard add\_quota <quota type> <naming scheme> <node number/name> <internal bundles>

<internal bytes>

[<restage link name> <auto reload> <external bundles> <external bytes>]

Quotas can be applied based on either the destination Endpoint ID (EID) or the source EID and three EID naming schemes are currently supported (ipn, imc and dtn). A couple examples are:

bard add\_quota dst ipn 34 … --- applies to bundles destined to ipn:34.\*

bard add\_quota src dtn alpha … --- applies to bundles sourced from dtn://alpha/\*

Note that quotas are not granular to the service level. You cannot apply a quota to only bundles destined to ipn:34.2 for example.

You then specify the quotas for the Internal Storage based on number of bundles allowed and/or the max number of payload bytes allowed. Zeroes for either of these values indicate there is no limit.

bard add\_quota dst ipn 34 1000 200M

--- allows up to 1000 bundles or 200 MB of payloads for bundles destined to ipn:34.\* whichever

limit is hit first.

* If bundles have 50 MB payloads then only four would be allowed
* If a bundle of 201 MB is received as the first bundle then it would be rejected if possible
* If a bundle with a 1 byte payload is received while there are 1000 bundles then it would be rejected if possible
* If it is not possible to reject a bundle (see the BARD Overview) then the bundle will be loaded into primary memory even though it exceeds the quota

bard add\_quota dst ipn 34 1000 0

--- allows up to 1000 bundles regardless of how large the payloads

A quota can optionally be configured to offload bundles that exceed the Internal Storage quota to External Storage instead of trying to reject them:

bard add\_quota dst ipn 34 1000 200M restage1 true 0 0

--- all bundles received destined to ipn:34.\* that exceed the 1000 count or 200 MB quota will be

routed to a Restage Convergence Layer named “restage1” to be written to External Storage

* If External Storage fills up then bundles will be rejected if possible
* The “true” parameter indicates that as the Internal Storage bundles are transmitted and the Internal Storage usage drops to 20% of the quota then bundles will be automatically reloaded from External Storage
* The first “0” parameter indicates that there is no limit on the number of bundles that can be written to External Storage
* The second “0” parameter indicates that there is no limit to the total bundle payload size that can be written to External Storage

Quota values for count and payload size can be specified for the External Storage if desired. If a bundle that would exceed both Internal Storage quota and External Storage quota is received then, if possible, it will be rejected; otherwise, it will be accepted into Internal Storage even though it exceeds quota. This is to prevent loss of bundles after a reliable link has indicated to the other end that a bundle has been accepted.

If you issue an add\_quota subcommand for a quota that already exists then it just replaces it so it is really an “add or modify” command.

See the following BARD Quota Persistence section for more info.

## del\_quota subcommand

Quotas can be removed using “bard del\_quota” command:

bard del\_quota <quota type> <naming scheme> <node number/name>

Given the example quota from the add\_quota example above:

bard add\_quota dst ipn 34 1000 200M restage1 true 0 0

The quota can be removed with the command:

bard del\_quota dst ipn 34

TBD: If a quota is deleted while bundles are offloaded then are they automatically reloaded into Internal Storage?

See the following BARD Quota Persistence section for more info.

## unlimited\_quota subcommand

Instead of deleting a quota, the “bard unlimited\_quota” command explicitly specifies that there is no limit on a particular set of bundles.

bard unlimited\_quota <quota type> <naming scheme> <node number/name>

Note that this is just eye candy for the equivalent command:

bard add\_quota <quota type> <naming scheme> <node number/name> 0 0

See the following BARD Quota Persistence section for more info.

## BARD Quota Persistence

BARD quota commands contained in the DTNME startup configuration file are not stored in the database but commands entered on the console or via AMP after the BARD is up and running are stored in the database. This is by design so that manual changes applied to the BARD system will persist across recycles and override those in the startup configuration file.

This feature applies to the add\_quota, del\_quota and unlimited\_quota commands.

The add\_quota command is the simplest use case. If the startup configuration contains a quota definition and then it is modified with a manual add\_quota command, the modified settings are saved in the database will override the configuration file if the node is recycled.

The del\_quota command has little effect if used in the startup configuration file, as it will not delete an entry from the database. It would delete the quota from an add\_quota earlier in the configuration file but in that case, you could just delete the earlier line.

After the BARD is up and running, the del\_quota command will delete a quota from in memory use and from the database. Now in this case, the current run would no longer apply the deleted quota but if the node is recycled then if the deleted quota is in the startup configuration file then it will be in force again. The del\_quota command cannot persistently override a quota that is in the startup configuration file.

So, that brings us to the unlimited\_quota command which is the correct way to persistently override and delete a quota that is in the startup configuration file. This command also has limited value if it is in the startup configuration, as it will not override a quota that was previously stored in the database.

There is not currently a way for the startup configuration to clear/override quota definitions that are stored in the database. The quotas stored in the database that were active when the node terminated will be active when it is restarted.

## force\_restage subcommand

The “bard force\_restage” command can be used to offload bundles that are over quota after a quota for a set of bundles been set or lowered after the bundles were already in Internal Storage:

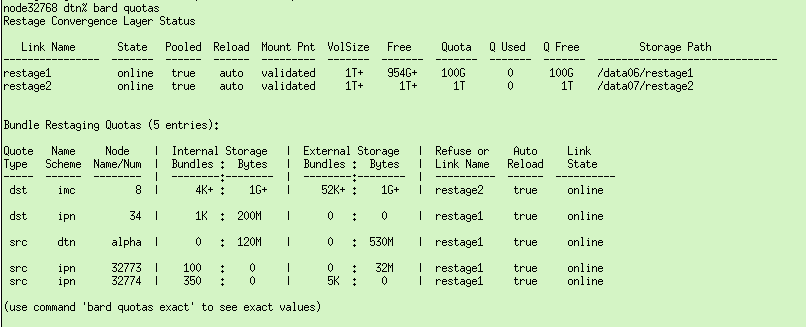
bard force\_restage <quota type> <naming scheme> <node number/name>

## quotas subcommand

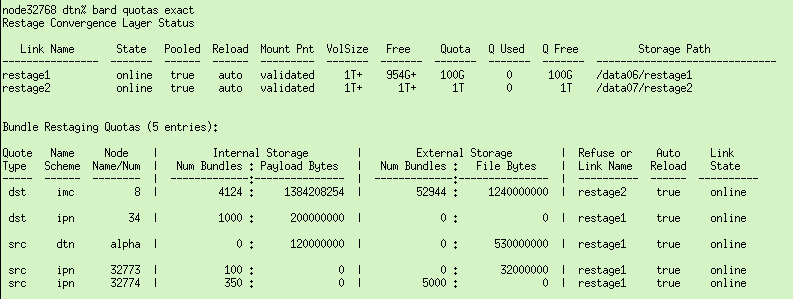
The “bard quotas” command generates a report showing a summary of the defined Restage Convergence Layers and a listing of all of the defined quotas. The base command rounds the values to the nearest magnitude value or you can add the optional “exact” parameter to see the actual numbers.

bard quotas [exact]

Here is an example report using the base command:



Here is the same definitions using the “exact” optional parameter:

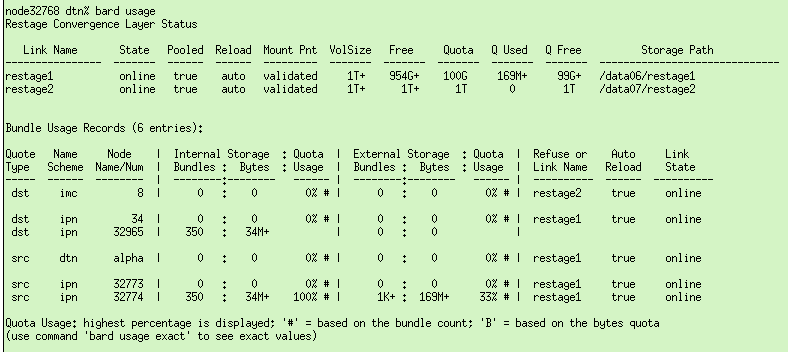


## usage subcommand

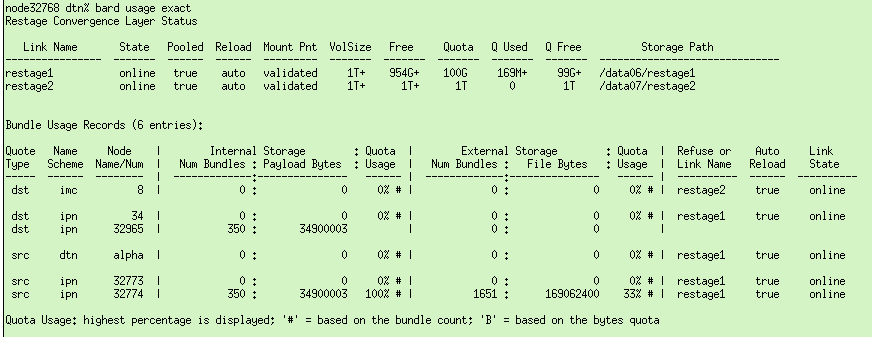
The “bard usage” command generates a report showing a summary storage usage for Restage Convergence Layer level and all of the defined quotas. A highest percentage usage of the quota[s] is shown with an indication as to whether it is the quota based on bundle count or payload bytes. The base command rounds the values to the nearest magnitude value or you can add the optional “exact” parameter to see the actual numbers.

bard usage [exact]

Here is an example report using the base command:



Here is the same definitions using the “exact” optional parameter:

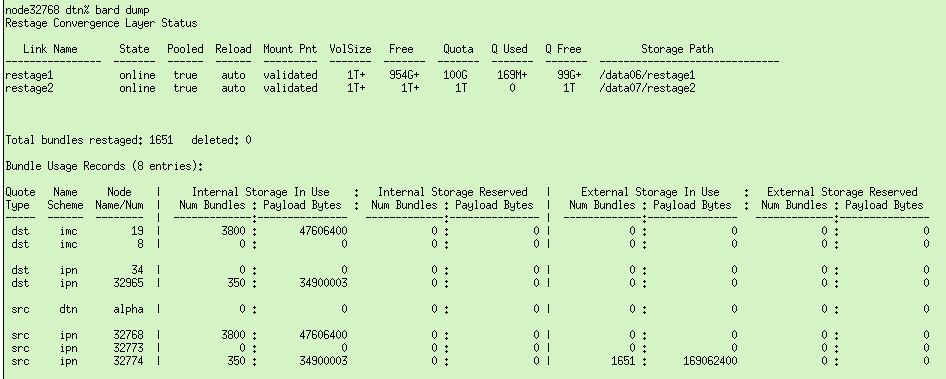


## dump subcommand

The “bard dump” command generates a report showing storage usage information for all bundles in Internal and External Storage whether or not they have a defined quota.

bard dump

Here is an example report:



Remember that the previous “bard usage” report showed that there were 350 bundles in internal storage and 1651 bundles in external storage source from IPN node 32774. Here those bundles are shown on the bottom line of the report and the same 350 bundles in Internal Storage show up under the destination, which is IPN node 32965. The 1651 bundles in External Storage were restaged based on the source node quota so they do not show up under the destination node.

Notice that there are also 3800 bundles from IPN node 12345 to IMC node/group 19 which in Internal Storage. There is no quota defined that applies to these bundles but they still show up in this report. Once again the usage numbers show up under both the source and destination nodes.

## rescan subcommand

The “bard rescan” command can be used to manually force the Restage Convergence Layers to rescan their storage location to determine if any bundle files have been added or deleted by some action outside of the DTNME server operations. Maybe a system manager restored a backup of the External Storage that you would like the running DTNME system to pull in and send. The reload capability includes the option to adjust the bundle expiration so this feature could be used to “replay” bundles that have previously expired.

bard rescan

## del\_all\_restaged\_bundles subcommand

The “bard del\_all\_restaged\_bundles” command can be used to delete all bundles in External Storage.

bard del\_all\_restaged\_bundles

## del\_restaged\_bundles subcommand

The “bard del\_restaged\_bundles” command can be used to delete a specific set of bundles from External Storage.

bard del\_restaged\_bundles <quota type> <naming scheme> <node number/name>

## reload subcommand

The “bard reload” command can be used to manually kick of an attempt to reload a specific set of bundles from External Storage with an option to change the configured expiration adjustment and/or the destination of the bundles.

bard reload <quota type> <naming scheme> <node number/name> [<new expiration secs>]

[<new dest EID>]

## Reload\_all subcommand

The “bard reload” command can be used to manually kick of an attempt to reload all bundles from External Storage with an option to override the configured expiration adjustment.

bard reload\_all [<new expiration secs>]

# Restage Convergence Layer Configuration

The Restage Convergence Layer (CL) must be configured to manage each External Storage location.

The common format for adding a link definition is:

link add <name> <nexthop> <type> <conv\_layer\_type> <opt=val> <opt2=val2>...

In this case, the <type> is “ALWAYSON”, the <conv\_layer\_type> is “restage”, and the path to the top-level directory of the External Storage location is specified in the <nexthop> position.

If the location is on a mounted disk drive then the “mount\_point=true” parameter can be set to force a check to verify that the disk is actually mounted before using the storage location path.

You can specify how long to retain bundles and whether to delete them if they expire prior to the retention time. Why would you not want to delete them when they expire? The “ttl\_override=<seconds>” parameter allows you to override the bundle expiration on retrieval to provide at least that amount of time for the destination node to get and process them. This can be beneficial to some use cases such as ours.

The “auto\_reload\_interval=<seconds>” parameter can be used to configure the Restage CL instance to periodically try to reload bundles into Internal Storage if quota is available.

The “part\_of\_pool=<true|false>” parameter informs the BARD system whether or not this Restage CL instance should be considered as part of the pool of storage locations or a standalone location.

The “disk\_quota=<size>” parameter is used to limit the amount of disk space that can be used for bundle storage. If you have a 1GB disk then you might want to reserve 100MB for other usage or overhead. The “min\_disk\_space=<size>” parameter specifies the minimum amount of disk space that must be available before declaring the storage location ONLINE vs FULL. Your other usage might actually exceed

When a BARD quota is configured, it specifies the name of the Restage CL instance to which those bundles should be sent for offloading. If the instance is configured as a pool then BARD will first check to see if its storage has room for the bundle and if not then it will try the other Restage CLs that are configured as part of the pool. Note that there is only support for a single pool of storage locations.

From the DTNME console, the “link options restage” command shows the options available for configuring an External Storage location for BARD usage:

dtnme% link options restage

Restage Convergence Layer [restage] - valid Link options:

<next hop> format for "link add" command is <path to storage location>

CLA specific options:

mount\_point <Bool> - Whether to verify <path to storage location> is on a

mounted drive (default: true)

days\_retention <U64> - Maximum number of days to store bundles for retrieval

(default: 7)

expire\_bundles <Bool> - Whether to delete bundles from storage when they expire

(default: false)

ttl\_override <U64> - Override bundle expiration if necessary to provide

specified minimum number of seconds time to live (TTL) on reload (default: 0)

auto\_reload\_interval <U64> - How often (seconds) to try to automatically reload

bundles (0=never; default: 3600)

disk\_quota <Size> - Max disk space to use for storage (default: 0 = no limit

other than disk space)

min\_disk\_space <Size> - Minimum disk space needed to declare state ONLINE vs

FULL (default = 100MB)

min\_quota\_avail <Size> - Minimum quota space needed to declare state ONLINE after

a FULL state (default = 1MB)

part\_of\_pool <Bool> - Whether this instance is part of the BARD pool of

storage locations (default: true)

email\_enabled <Bool> - Whether to send alert emails (default: true)

from\_email <String> - Email address to use as the sender of alerts

add\_email <String> - Add an email address for alerts; (may be specified

multiple times)

del\_email <String> - Remove an email address from alert notifications; (may

be specified multiple times)

(parameters of type <U64> and <Size> can include magnitude character (K, M or G): 125G = 125,000,000,000)

Options for all links:

<not applicable to the Restage Convergence Layer>

Example:

link add restage1 /data01/ext\_storage ALWAYSON restage mount\_point=true ttl\_override=86400 auto\_reload\_interval=3600

(create a link named "restage1" with top level storage location of /data01/ext\_storage;

\* verify that /data01/ext\_storage or /data01 is mounted;

if /data01 is a mount point but /data01/ext\_storage does not exist then it will be

created;

\* retain bundles for up to 7 days regardless of their expiration time;

\* try to reload bundles to internal storage every hour and override the bundle TTL if needed

to provide a minimum of 1 day;

\* it is part of the BARD pool and if it fills up, bundles configured to restage here can be

sent to other members of the pool and vice versa;

\* email alerts are enabled but no email addresses were specified so none will be sent;

# File transfer utility (deliver\_me)

The deliver\_me application is an initial implementation of the CCSDS File Delivery Protocol (CFDP) based on the CCSDS 727.0-B-5 recommended standard. This version is mostly compatible with the ION implementation without support for all of the ION options. The most notable limitation is that it does not currently support the closure requested parameter and so does not send a Finished PDU.

The CFDP receive capability is not built into the DTNME server so the app must be run to receive files as well as to send files. DTNME does retain bundles sent to the local node so file can be sent to the node and then the deliver\_me application can be run (before the bundle expire) to receive them and the file. Note that one the deliver\_me app begins receiving bundles for a file it must be left running until the entire file is received. It will not pick up where it left off after a restart.

Below is the help output for the deliver\_me application:

**> deliver\_me -h**

usage: deliver\_me [opts]

opts:

-h, --help show usage

--a <api\_address> address used to connect to the DTNME instance. Default is 127.0.0.1

--p <api\_port> port used to connect to the DTNME instance. Default is 5010.

**--e <entity\_id> id to represent the local cfdp entity**

--s <file\_segment\_size> size used to define how much of file to send per bundle.

Default is 65000.

--l <log\_path> path used to store logs. Default is local directory (./).

--t <time\_to\_live> set the time to live to be used by bundles transmitted from this

entity. Default is 86400 seconds.

--c <checksum\_type> set the checksum type to be used for calculating checksums on received

data

--v <bundle\_version> set the bundle protocol version to be used. Options are 6 or 7.

Default is 7.

--q <sequence\_num> set the starting transaction sequence number.

Most options default for general usage. The single option that must be specified is the **--e <entity id>** parameter. This value should match the IPN node number of the local DTNME node. It will receive bundles sent to the **ipn:< entity\_id>.65** Endpoint ID.

When the deliver\_me app is run, it outputs the available commands and a prompt as shown below.

> **deliver\_me --e 12345**

[1667185311.039196 /deliver\_me notice] random seed is 39195

|----Usage----|

put.request <destination\_entity\_id> <source\_filename> <destination\_filename> <checksum\_type>

transaction.list

exit

|-------------|

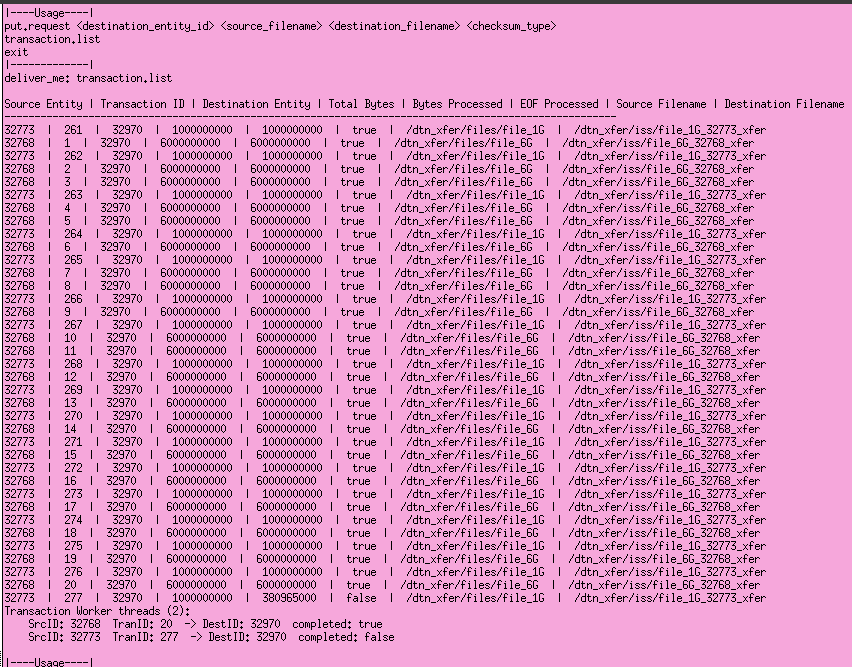
deliver\_me:

To send a file, enter a put.request command at the deliver\_me prompt:

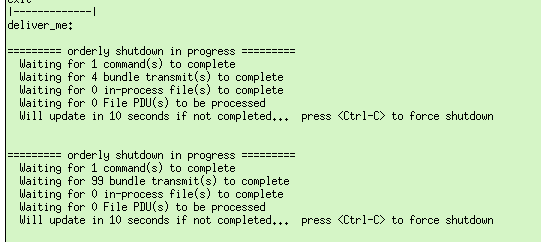
deliver\_me: **put.request 54321 /path/source\_file /remote\_path/destination\_file 0**

This command would send the /path/source\_file to node 54321 (bundle destination ipn:54321.65) where it will be written to /remote\_path/destination\_file. The only supported checksum value is zero.

Below is a sample output for the “transaction.list” command. It shows the file transfer history since the deliver\_me app has been running. This history is on node 32970, which had an ION node 32773 and a DTNME node 32768 sending files to it. The nodes are all running on the same computer so the transmit test scripts could verify the transferred file and delete it before sending it again.



When you use the exit command, the deliver\_me app will continue sending or receiving in-progress files before actually terminating. The output below is from the test script entering the exit command so you cannot actually see the “exit” next to the “deliver\_me:” prompt. Every 10 seconds, it will show what it is waiting on. If needed, you can force immediate termination using <Ctrl-C> but this will result in loss of the transfer(s) currently in progress.



**Bonus file transfer throughput hack:**

The CFDP standard uses a 16-bit field for its PDU length, which theoretically, limits PDUs to 65535 bytes. To date, both ION and DTNME use the bundle payload size to determine the length of the file segment rather than the PDU length field. The deliver\_me app allows you to specify a larger file segment size, which improves overall throughput performance. It does display a warning in case you are testing strict CFDP compliance.

The ION implementation allows you to specify a larger value as well but it truncates it to the lowest 16 bits so be aware that if you specify a segment length of 65536 then it will try to use a length of zero an not send the file. A length of 65537 will use result in using a 1-byte segment length and so on. The max you can use is 65535. This does result in exceeding the CFDP standard by a few bytes, as it does not include the length of the offset value, which is also part of the File PDU.

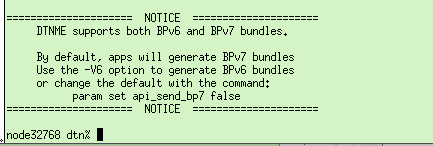
# Miscellaneous updates

There are a few minor tweaks of note in this release:

\* Whether or not to generate Status Reports has been changed to default to true. Use this command to disable Status Report generation: bp set status\_rpts\_enabled false

\* The default expiration time for ping\_me bundles has been changed from 30 seconds to 5 minutes.

\* When the DTNME server starts up, it now displays messages indicating whether applications will default to sending BPv6 or BPv7 bundles and how to change it.



\* The BPv6 Primary Block processor now supports receiving IMC Endpoint IDs in the CBHE format.

\* Updated to build on Red Hat 8 and support checking for newer versions of the gcc compiler.

\* TCP Convergence Layer was not applying all specified parameters on the “interface add” command to incoming connections. (Notably, break\_contact\_on\_keepalive\_fault could not be set to false for better compatibility with incoming ION 3.6.2 connections).