**DZ Notes on the DTNME Bundle Multi-Destination**

**Beta Release**

**DTNME 1.3.0-Beta**

**01/26/2023**

Contents

[Bundle Multi-Destination Overview 3](#_Toc125379234)

[ION BPv7 Implementation 3](#_Toc125379235)

[DTNME BPv7 Implementation 4](#_Toc125379236)

[DTNME BPv6 Implementation 5](#_Toc125379237)

[Build Instructions 7](#_Toc125379238)

[New ROUTE Subcommands 8](#_Toc125379239)

[IMC Database Persistence 10](#_Toc125379240)

# Bundle Multi-Destination Overview

We are using the term “Bundle Multi-Destination” as opposed to “Bundle Multicast” to avoid confusion with the network capability known as UDP Multicast. Bundle Multi-Destination simply refers to the capability to originate a single bundle that eventually is delivered to multiple destinations. Bundle Multi-Destination does not require usage of network multicast. Bundle Multi-Destination may improve both storage and bandwidth resources over sending duplicate bundle payloads to each destination.

The DTNME Bundle Multi-Destination implementation was designed to be compatible with the ION implementation since there is not a specification at this time. DTNME supports both Bundle Protocol version 6 (BPv6) and version 7 (BPv7). The BPv6 implementation is compatible with ION 3.7.4 and the BPv7 implementation is compatible with ION 4.1.1.

The ION and DTNME Bundle Multi-Destination implementations require DTN nodes to use the “InterPlanetary Network” (IPN) naming scheme and utilize the experimental “InterPlanetary Multicast” (IMC) naming scheme for the bundle destination Endpoint ID. This document will use the term “IMC Bundle” to refer to a multi-destination bundle with an IMC destination Endpoint ID. The format of the IPN naming scheme is **ipn:<node#>.<service#>** and the format of the IMC naming scheme is **imc:<group#>.<service#>**.

## ION BPv7 Implementation

\*\* The ION BPv7 implementation was more advanced than the BPv6 implementation and should be the basis for future experimentation/enhancement. The general description below is for the ION BPv7 implementation.

Each DTN IPN node is configured to be in a specific “region” number or “home region” and some nodes can act as a passageway to an “outer region”. ION documentation indicates that each region is a notionally small (16-32) set of nodes that customarily use contact graph routing to compute forwarding routes among themselves. ION configures a node’s home region number while configuring the contact plan. I don’t think that the outer region will be configurable/functional until a version with Inter-Regional Forwarding (IRF) is released. I received a pre-release version with IRF late in the development process and did not have the resources to delve into it. It sounds like the IRF version will further improve the interoperability with this version of DTNME.

A DTN IPN Node notifies all of the other DTN IPN Nodes in its home region that it wants to receive IMC bundles for a specified group number by issuing an IMC petition bundle to destination **imc:0.0,** which is reserved for this purpose. At start up, nodes send an IMC Petition bundle requesting to join the special group number zero (0) which triggers a synchronization between the nodes using IMC briefing bundles described below. When an IMC endpoint is added via the bpadmin utility, an IMC petition bundle is sent to all of the home region nodes requesting to join the appropriate group number.

When a node receives an IMC petition bundle for group number zero (0) from another DTN IPN node for the first time then it sends back an administrative bundle with a briefing of all of the IMC groups that it wishes to receive. This new administrative type is a value of five and ION refers to it as type BP\_MULTICAST\_BRIEFING. The IMC petition bundles and the initial briefing messages provide the infrastructure to facilitate forwarding IMC bundles to all of the requesting destinations.

When an IMC bundle is generated, a new IMC Destinations extension block is included in the bundle. This IMC Destinations extension block uses block type code 195 and simply contains a list of IPN node numbers to which the bundle is to be sent. It is initialized with the list of nodes that the originator knows have requested to receive the IMC group number.

During the forwarding/routing of the IMC bundle, all of the destinations are evaluated to see which outduct is appropriate for each and a copy of the bundle is made and sent for each outduct with only the reachable nodes in its IMC Destinations block. When ION copies or clones a bundle, the new bundle shares the payload (memory or file) from the original so there is some overhead but not as much as sending separate bundles to each destination. As the IMC bundles are received and forwarded, the list of the IMC Destinations for each cloned bundle eventually gets whittled down to just one node for one last transmission. I believe that in the newer IRF version of ION, passageway nodes receiving bundles from an outer region can add IMC destinations for its home region before doing its forwarding process.

## DTNME BPv7 Implementation

The current DTNME BPv7 (and BPv6) implementation is an extension to the static router and an “IMC Router” runs as a separate thread if enabled. New “route” subcommands allow configuration of region numbers and members and also IMC group numbers that a node always wants to receive whether an application is active or not. Each region must have at least one active and enabled IMC Router node for proper functionality. When configuring DNTME nodes, all ION nodes should be flagged as IMC Router nodes. Non-IMC Router nodes need to be configured to forward all IMC bundles to an active IMC Router node. Limiting the number of full blown IMC Router nodes helps reduce the coordination traffic while still providing the IMC service.

The DTNME BPv7 implementation is compatible with the ION implementation so it also utilizes the IMC Destinations extension block (type 195) exactly as described in the ION implementation above. DTNME also introduces a new IMC State extension block (type 222) which carries additional information to be used in the downstream forwarding process. ION does not recognize this block but passes it through so that other DTNME nodes downstream can use it. In the future, if elements of the new block are deemed worthy and the implementations converge on an inclusive standard then these could be combined into a single block.

The IMC State extension block actually has 3 formats and uses:

1. Included on regular IMC bundles it provide lists of region numbers and proxy/passageway nodes that have processed the bundle. The lists are retained on each copy/clone and grow as the bundle is forwarded. These lists are used to prevent redundant processing and circular forwarding.
2. Included on IMC Group petition bundles (imc:0.0), it provides the same two lists as for regular IMC bundles and also flags indicating whether or not this is a proxy/passageway petition and if an IMC sync is requested. The fact that a bundle has this extension block allows DTNME nodes to determine that it originated from a DTNME node rather than an ION node and thus whether to generate an ION compatible briefing bundle or a more detailed DTNME briefing bundle when requested.
3. Included on IMC Briefing bundles, it allows a DTNME node to determine that it originated from a DTNME node rather than an ION node which have different formats. It also has flags for DTNME nodes to indicate whether a return briefing is requested, whether this is a response to a request and whether or not the sending node is a DTNME IMC Router node.

For more detailed information on the IMC State Block see the source code file: servlib/bundling/BP7\_IMCStateBlockProcessor.cc.

DTNME nodes will always send an administrative IMC briefing bundle in response to receiving a group petition (imc:0.0) for the special group number zero rather than just the first time. It was found that if a node had to be restarted then it was not getting the briefing it needed. DTNME refers to the group zero petition as an IMC Sync request and can be configured to issue one at startup to be sent to all nodes in its home region. When a TCP Convergence Layer connection is established and the announced node is an IPN node then DTNME will issue an IMC Sync request to just that one node. Briefings between DTNME nodes are more detailed than ION supports and includes the home region number, a list of nodes in the home region and a list of IMC group numbers and the nodes requesting each.

When an application registers to receive an IMC Endpoint ID (EID), a group petition is generated to join the appropriate IMC group number and is sent to all nodes in the home region. Users can also include “manual joins” in the startup configuration file for desired IMC Group and Service numbers which will generate the group petitions at startup and keep any IMC bundles that match until an application registers to receive them (or they expire).

DTNME IMC passageway nodes support access to any number of outer regions instead of just one and must always be configured as an active and enabled IMC Router node. When they receive an IMC group petition bundle, instead of forwarding the original petition, they generate a new petition bundle indicating it is acting as a proxy and forwards it to other outer regions and/or nodes in the home region as appropriate. If the original petition bundle includes an IMC State block then its information is included in the new proxy bundle’s IMC State block. The lists of processed regions and passageway/proxy nodes in the IMC State block are used to prevent redundantly sending the proxy petition to nodes and regions.

When a “regular” IMC bundle (destination is not group zero) is received at a DTNME IMC Router node, it will add nodes within its home region to the IMC Destinations block and add or update the IMC State block to indicate that the home region and the local node number has processed it. Forwarding will then take place similar to that in the ION implementation. Instead of cloning the bundle as ION does, the IMC router includes information in the bundle object indicating which destination nodes are to be used when sending over each link/outduct. As the bundle is being serialized for transmission over a specific link, the IMC Destinations block is customized for that link.

Initial testing with a mix of DTNME and ION nodes was successful. When there are multiple interconnected regions then some downstream nodes will receive multiple copies of an IMC bundle which the application will need to be able to handle. It might be possible to include additional information in the IMC State block such as a list of known requesters to prevent some of the duplicate deliveries but I am not sure it could prevent it in all scenarios. It might also grow the IMC State block to a size too large to allow a bundle to be fragmented for a smallish MTU leg.

## BPv6 Implementations

This DTNME release is also compatible with the ION 3.7.3 BPv6 implementation. The ION BPv6 IMC implementation is not as advanced as the BPv7 implementation. It treats all nodes as being in a single region and there is not an IMC Destinations extension block. The DTNME BPv6 implementation continues to utilize the BPv7 enhancements even though ION does not recognize them. ION nodes pass through the unrecognized blocks so the DTNME nodes can still utilize them. DTNME configuration for BPv6 is the same as for BPv7 with a new route subcommand to enable sending BPv6 group petition bundles.

# Build Instructions

The latest NASA version of the DTNME 1.3.0-Beta source code can be downloaded from the GitHub site:

https://github.com/nasa/DTNME/tree/v1.3.0\_Beta

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 ROUTE Subcommands

All of the IMC related configuration is done via new subcommands of the “route” command. At the DTNME console prompt, you can issue the command “help route” to see a list of all subcommands and a description. The table below lists the new subcommands which would be invoked by entering “**route <SUBCOMMAND>**” into a startup configuration file or in the DTNME console.

|  |  |
| --- | --- |
| **SUBCOMMAND** | **DESCRIPTION** |
| imc\_router\_enabled [true|false] | Get or set whether the IMC Router is enabled   * If not enabled then a static route should be included to send all IMC bundles to an IMC Router node for processing if appropriate for the node |
| is\_imc\_router\_node [true|false] | Get or set whether this node is an active IMC Router node   * Set to true and this node is to do full up IMC routing or set to false and this node will forward IMC bundles to the first reachable active IMC Router node from its configuration or learned of via briefings |
| imc\_home\_region <region#> | Get or set the home region number for this node   * Set the local\_eid\_ipn node number prior to using this command |
| clear\_imc\_region\_db <id#> | Get the current id# or clear the IMC Region database if <id#> does not match the value stored in the database. Using an id# of zero will always clear the database.   * See IMC Database Persistence section below |
| clear\_imc\_group\_db <id#> | Get the current id# or clear the IMC Group database if <id#> does not match the value stored in the database. Using an id# of zero will always clear the database.   * See IMC Database Persistence section below |
| clear\_imc\_manual\_join\_db <id#> | Get the current id# or clear the IMC Manual Join database if <id#> does not match the value stored in the database. Using an id# of zero will always clear the database.   * See IMC Database Persistence section below |

|  |  |
| --- | --- |
| **SUBCOMMMAND** | **DESCRIPTION** |
| imc\_region\_add <region#>  <router\_nodes? true|false>  <start\_node#>  [<end\_node#>] | Add a node or inclusive range of node numbers to the region number and flag whether or not they are active IMC Router nodes   * NOTE: All ION nodes should be configured as “true” router nodes |
| imc\_region\_del <region#>  <start\_node#>  [<end\_node#>] | Delete a node or inclusive range of nodes from a region number |
| imc\_group\_add <group#>  <start\_node#>  [<end\_node#>] | Add a node or inclusive range of node numbers to the list of nodes to receive bundles to the specified IMC group number |
| imc\_group\_del <group#>  <start\_node#>  [<end\_node#>] | Delete a node or inclusive range of node numbers from the list of nodes to receive bundles to the specified IMC group number |
| imc\_manual\_join\_add <group#> <service#> | Manually join/register to receive bundles to EID imc:<group#>.<service#> and hold them until delivered to application   * At startup, the node will issue a group petition bundle for the group number even though an application has not started to receive it. |
| imc\_manual\_join\_add <group#> <service#> | Delete manual join/registration to receive bundles to EID imc:<group#>.<service#>   * Bundles to this group and service number will be deleted if there is not an application actively receiving them |
| imc\_region\_dump [<region#>] | List all of the configured IMC Regions and their member nodes or those of a specified region# |
| imc\_group\_dump | List all of the configured IMC Groups and their member nodes or those of a specified group# |
| imc\_manual\_join\_dump | List all of the manually joined/registered IMC destination EIDs |
| imc\_issue\_bp6\_joins [true|false] | Get or set whether this node initiates BPv6 join requests (default: false) |
| imc\_issue\_bp7\_joins [true|false] | Get or set whether this node initiates BPv7 join requests (default: true) |
| imc\_sync\_on\_startup [true|false] | Get or set whether this node initiates BPv7 synchronization between all IMC nodes in the region at startup (default: true) |
| imc\_sync [<node\_number>] | manually initiate an IMC Sync with all nodes or a single node\_number |
| imc\_delete\_unrouteable [true|false] | Get or set whether to delete bundles with all dest nodes unrouteable (default: true) |
| imc\_unreachable\_is\_unrouteable [true|false] | Get or set whether to treat known but currently unreachable nodes as unrouteable (default: true) |

# IMC Database Persistence

This section describes how the ID# for the subcommands clear\_imc\_region\_db, clear\_imc\_group\_db and clear\_imc\_manual\_join\_db are utilized. The potential issue being addressed is if you use a startup configuration file to initialize, for example, the nodes in an IMC Region and then make changes manually or via AMP what happens if you recycle the DTNME server? Will the startup configuration file which was not updated wipe out the changes made manually? The description below is for the regions but the same applies to the groups and manual joins.

It is suggested that when you create the startup configuration file, include a “route clear\_imc\_region\_db <id#>” before using the imc\_region\_add commands to define the nodes in the regions. A good candidate for the <id#> is to use the date in the format yyyymmdd as the value. That does double duty providing you a timestamp as to when the configuration was last updated.

At initial startup, when the DTNME server processes the clear\_imc\_region\_db subcommand, there will not be an ID# stored in the database so it will clear the database just in case it isn’t already and it will store the ID# in the database. It will then continue on and apply any imc\_region\_add and imc\_region\_del subcommands to the database.

Say, you manually move one of the nodes to a different region using the imc\_region\_add subcommand. That change is made to the database so it is now different than what is in the startup configuration file.

If then you re-cycle the DTNME server, when it processes the clear\_imc\_region\_db subcommand, the matching ID# is in the database so it will ignore the command and not clear the all of the region info currently in the database. As it continues on processing the imc\_region\_add and imc\_region\_del subcommands, it will find the node numbers are already in the database and will not change them. (If you add some new nodes in the configuration file then it will add those to the database).

NOTE: If you issue the imc\_region\_del subcommand then it does not delete the record for the node number(s) from the database. Instead, it sets a flag in the database indicating that the node number(s) was explicitly deleted from the region number. If you then issue an imc\_region\_add subcommand for a “deleted” node number it will update the node’s record to indicate that it is explicitly added to the specified region number and no longer indicate that it was “deleted” from the prior region.

All of this allows you to set an initial configuration and then make manual changes without having to worry about keeping the startup configuration file up to date in case of a re-cycle.

If there comes a time when you do want to start over and wipe out all database entries at startup then just change the ID# in the “route clear\_imc\_region\_db <id#>” subcommand. Once again, I would suggest using the current date when making the changes. If you need to this multiple times a day then could include the time as well as the date in the ID# value.

For the database to be actually cleared, the ID# just needs to be different than what is stored in the database. It does not have to be greater than what is currently stored in the database. Using a value of zero or leaving off the <id#> will always clear the region database records.

# Sample Configuration

A sample configuration for a DTNME node 100 that is in home region 1 and is passageway node to regions 2 and 3 might look something like this:

########################################

#

# IMC Routing Configuration

#

########################################

# clear the region database if this is the first

# startup today’s timestamp (01/25/2023)

route clear\_imc\_region\_db 20230125

# Enable the IMC Router processing

route imc\_router\_enabled true

# and designate this node as an official IMC Router Node

route is\_imc\_router\_node true

# only issue BPv7 IMC Group Petitions

route imc\_issue\_bp6\_joins false

route imc\_issue\_bp7\_joins true

# set the home region for this node

route imc\_home\_region 1

# at startup, send an IMC Sync Request bundle to

# all nodes in the home region

route imc\_sync\_on\_startup true

# configure all nodes in our home region

## nodes 101 through 105 are DTNME nodes but not IMC Router nodes

route imc\_region\_add 1 false 101 105

## ION nodes 106 through 109 must be configured as IMC router nodes

route imc\_region\_add 1 true 106 109

# configure just the gateway node for outer region 2

## since passageway node 200 is a DTNME node it will handle

## all of the DTNME and ION nodes in its region

route imc\_region\_add 2 true 20

# region 3 has an ION node 300 as its passageway node

## If there is a DTNME node 301 in region 3 that could act as an

## IMC Router node then you could configure it as the sole IMC Router

## node for the region even though you have to go through

## node 300 to get to it.

## Otherwise, configure all nodes in the outer region as

## IMC Router nodes

route imc\_region\_add 3 true 300 309

### there may be some nodes that need to receive IMC bundles

### but for some reason cannot issue IMC Group Petition bundles.

### You can configure the IMC Router node to make sure that they

### receive it

#

# clear the group database if this is the first startup with

## today’s timestamp (01/25/2023)

route clear\_imc\_group\_db 20230125

# manually add node 108 to IMC Group 42

route imc\_group\_add 42 108

### this node needs to receive bundles to imc:16.4 but the

### application will only run once a day at 2:00 am to

### collect the bundles so you can configure a “manual join”

### to force it to send the IMC Group Petition for

### IMC Group 16 and if bundles to imc:16.4 are received then

### they will be kept.

### (IMC Group 16 bundles to service numbers other than 4 will

### be deleted if an application has not registered to receive

### them.)

#

# clear the manual join database if this is the first startup with

## today’s timestamp (01/25/2023)

route clear\_imc\_manual\_join\_db 20230125

# manually join to IMC Group 16 and Service Number 4

route imc\_manual\_join\_add 16 4