HDTN README and User Guide

November 21, 2022

Contents

1 High-rate Delay Tolerant Network Overview

Delay Tolerant Networking (DTN) has been identified as a key technology to facilitate the development and growth of future space networks. Existing DTN implementations have operated in constrained environments with limited resources resulting in low data speeds, and cannot utilize more than a fraction of available system capacity. However, as various technologies have advanced, data transfer rates and efficiency have also advanced. To date, most known implementations of DTN have been designed to operate on spacecraft. HDTN takes advantage of modern hardware platforms to substantially reduce latency and improve throughput compared to today's DTN operations. HDTN implementation will maintain compatibility with existing deployments of DTN that conform to IETF RFC 5050. At the same time, HDTN defines a new data format better suited to higher-rate operation. It defines and adopts a massively parallel pipelined and message-oriented architecture, allowing the system to scale gracefully as its resources increase. HDTN's architecture also supports hooks to replace various processing pipeline elements with specialized hardware accelerators. This offers improved Size, Weight, and Power (SWaP) characteristics while reducing development complexity and cost.

2 Architecture

Figure 1 shows the HDTN modules and their interactions:

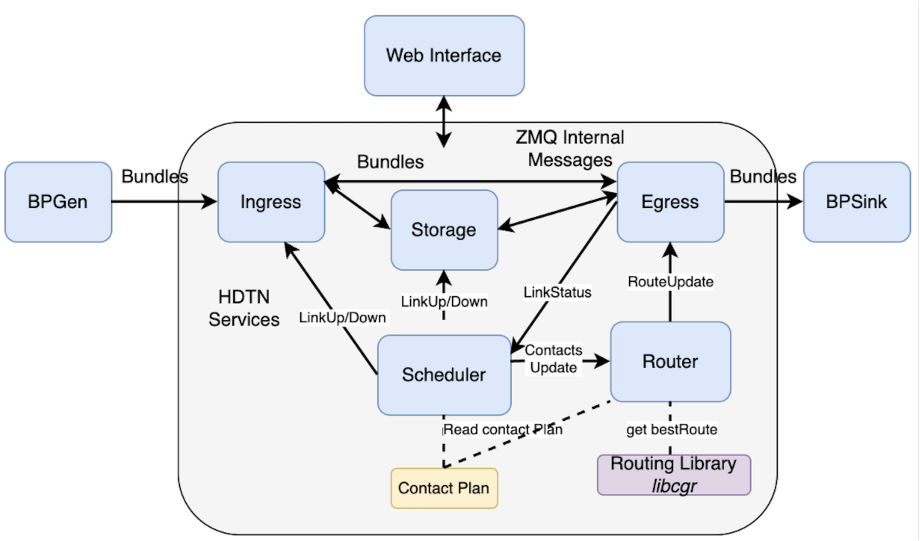

Figure 1: HDTN architecture.

2.1 BpGen

BPGen is a tool that generates bundles of any specified size and is intended to be used with its receiving tool BpSink. It is not a component of HDTN, but it shares the same HDTN codebase and libraries.

2.2 Ingress

The Ingress module intakes bundles generated by the BPGen tool and decodes the header fields to determine the source and destination of the bundles. If the link is available, Ingress will send the bundles in a cut-through mode straight to Egress, and if the link is down or custody transfer is enabled it sends the bundles to the Storage module. Even if an immediate forwarding opportunity exists, Storage is always required when custody transfer is enabled. The bundle layer must be prepared to re-transmit the bundle if it does not receive an acknowledgment within the time-to-acknowledge that the subsequent custodian has received and accepted the bundle.

2.3 Scheduler

The Scheduler sends `linkUp` or `linkDown` to Ingress and Storage to determine if a given bundle should be forwarded immediately to Egress or sent to Storage. To determine the availability of a given link, the Scheduler reads a contact plan which is a JavaScript Object Notation (JSON) file that defines all the connections between all the nodes in the network.

2.4 Storage

Storage is a multi-threaded implementation distributed across multiple disks and where custody transfer is handled. It receives messages from the Scheduler to determine when stored bundles can be released and forwarded to Egress.

2.5 Router

The Router module gets the next hop and best route leading to the final destination using one of the algorithms in the routing library. We currently support Contact Graph Routing (CGR), Dijkstra's algorithm (default algorithm used), and also Contact Multigraph routing (CMR). The Router then sends a RouteUpdate event to Egress to update its outduct to the outduct of that nextHop. If the link goes down unexpectedly or the contact plan gets updated, the Router is notified, recalculates the next hop, and send the RouteUpdate event to Egress.

2.6 Egress

The Egress module is responsible for forwarding bundles received from Storage or Ingress to the correct outduct and next hop based on the optimal route computed by CGR. HDTN uses an event-driven approach based on ZeroMQ

pub-sub sockets for sending unexpected link updates and contact plan changes from Egress to Scheduler. When the connection is lost unexpectedly, Egress will send a `LinkStatus` change message to the Scheduler, which triggers the Scheduler to send `linkUp` or `linkDown` events to Ingress and Storage. In addition, theScheduler will recompute the contact plan and send the message `ContactsUpdate` to the Router. The optimal route is then recomputed based on the new contact plan, and a `RouteUpdate` message is forwarded to Egress.

2.7 Web Interface

The Web Interface displays a data rates graph and statistics for network troubleshooting. It's also used for updating configuration, routes, and contact plans.

2.8 BPSink

BPSink is a tool that receives and validates the bundles forwarded from the HDTN Egress. It is not a component of HDTN, but it does share the same codebase and libraries as HDTN.

3 Requirements

3.1 Tested Platforms

- Linux
  - Ubuntu 20.04.2 LTS
  - Debian 10
  - RHEL (Red Hat Enterprise Linux) 8
- Windows
  - Windows 10 (64-bit)
  - Windows Server 2022 (64-bit)
  - Windows Server 2019 (64-bit)
- MacOS
- Raspbian

3.2 Dependencies

3.2.1 Linux Dependencies

The HDTN build environment requires:

- CMake version 3.16.3

- Boost library version 1.66.0 minimum, version 1.69.0 for TCPCLv4 TLS version 1.3 support
- ZeroMQ
- gcc version 9.3.0 (Debian 8.3.0-6)

These can be installed using the following command(s):
On Ubuntu
```
sudo apt-get install cmake build-essential libzmq3-dev
sudo apt-get install libboost-dev libboost-all-dev openssl
libssl-dev
```
On RHel
```
sudo yum update
sudo dnf install epel-release
sudo yum install cmake boost-devel zeromq zeromq-devel
```
On Debian
```
sudo apt-get install cmake build-essential openssl libssl-dev
sudo apt-get install libboost-dev libboost-all-dev libzmq3-dev
python3-zmq
```

3.2.2 Windows Dependencies

HDTN supports 9 permutations of the Visual Studio compilers on Windows:

- Versions: 2022, 2019, and 2017 (note: for 2017, only versions 15.7 and 15.9 have been tested)
- Editions: Enterprise, Professional, and Community

HDTN build environment on Windows requires:

- One of the supported Visual Studio compilers listed in the Overview section. Visual Studio must be installed for C++ Development during setup.
- PowerShell (recommended Visual Studio Code with the PowerShell extension installed)
- 7-Zip
- Perl (needed for building OpenSSL) with perl.exe in the Path environmental variable (Strawberry Perl for Windows has been tested)

3.2.3 Options

Optional CivetWeb Dependencies

If running in non-distributed mode (using hdtn-one-process executable only), the HDTN build environment will require the CivetWeb Embedded C/C++ Web Server Library to display only real-time HDTN data.

- The CivetWeb Embedded C/C++ Web Server Library can be obtained from https://github.com/civetweb/civetweb
- C and C++ libraries are to be built using cmake with websocket support enabled.
- Set the Cmake cache variable USE_WEB_INTERFACE to "On". (Note: The default is "Off".)
- Set the Cmake variables to point to your civetweb installation: civetweb_INCLUDE, civetweb_LIB, civetwebcpp_LIB

3.3 Known Issues

- Ubuntu distributions have been known to install older non-compatible versions of -CMake.
- Some processors do not support hardware acceleration or the RDSEED instruction. (Note: In the cmake file, both are set to "ON" by default.)

4 Build HDTN

4.1 Notes on HDTN CMake

All of HDTN's directories of modules/libraries contain their own CMakeLists.txt file. The root CMakeLists.txt adds all those modules/libraries to the HDTN project using the CMake add_subdirectory command. It should be noted that the HDTN CMake files are written using modern CMake paradigms, such as "dependencies as targets" which makes it much easier and cleaner to manage a multi-platform library like HDTN. In addition, package config information gets exported to the installation (install_root/lib/cmake) whenever a user wants to do a "make install". There is an example in tests/unit_tests_import_installation/CMakeLists.txt which is a copy of the regular tests/unit_tests/CMakeLists.txt except that the former uses the find_package package config information from the install_root/lib/cmake directory. The package config information is great for users that may want to write custom software projects that only use portions of the HDTN codebase such as a library of a particular convergence layer. The HDTN CMake tries to optimize the build as much as possible; it will test the compiler for more recent C++ standards, and it will test the compiler and the CPU for specific x86 hardware instructions and utilize those if available. Finally, the HDTN CMake supports building its libraries as either static or shared using CMake's GENERATE_EXPORT_HEADER which is required for building or using .dll files on Windows and for using GCC's new C++ visibility support.

4.2 Build HDTN on Linux

To build HDTN in "Release mode", perform the following steps. (Note: If the -DCMAKE_BUILD_TYPE is not specified, HDTN is built in "Release mode" by default).

- export HDTN_SOURCE_ROOT=/home/username/hdtn
- cd $HDTN_SOURCE_ROOT
- mkdir build
- cd build
- cmake ..
- make
  - Adding -j8 (i.e. make -j8) to the make will speed up the processing time but requires a system with at least 8 cores.

Note: Use **make install** to...

Note: By Default, BUILD_SHARED_LIBS is OFF and hdtn is built as static. To use shared libs, edit CMakeCache.txt, set $BUILD\_SHARED\_LIBS : BOOL = ON$ and add fPIC to the Cmakecache variable:

$CMAKE\_CXX\_FLAGS\_RELEASE : STRING = -03 -DNDEBUG -fPIC$

4.3 Optional X86 Hardware Acceleration

HDTN build environment sets the following CMake cache variables to "On" by default:
USE_X86_HARDWARE_ACCELERATION and LTP_RNG_USE_RDSEED.

Notes:

- If building natively (i.e. not cross-compiling), the HDTN CMake build environment will check the processor's CPU instruction set and the compiler to determine which HDTN hardware accelerated functions will build and run on the native host. CMake automatically sets various compiler definitions to enable supported HDTN hardware accelerated features.
- If cross-compiling, the HDTN CMake build environment will check the compiler to determine if the HDTN hardware accelerated functions will build. It is up to the user to determine if the target processor can support-/run those instructions. CMake will automatically set various compiler definitions to enable supported HDTN hardware accelerated features only if they compile.
- Hardware accelerated functions can be turned off by setting USE_X86_HARDWARE_ACCELERATION and/or LTP_RNG_USE_RDSEED to "Off" in the CMakeCache.txt.

- If building for ARM or any non X86-64 platform, USE_X86_HARDWARE_ACCELERATION and LTP_RNG_USE_RDSEED must be set to "Off".

If USE_X86_HARDWARE_ACCELERATION is turned "On" some or all of the following features will be enabled if CMake finds support for these CPU instructions:

- Fast SDNV encode/decode (BPv6, TCPCLv3, and LTP) requires SSE, SSE2, SSE3, SSSE3, SSE4.1, POPCNT, BMI1, and BMI2.
- Fast batch 32-byte SDNV decode (not yet implemented into HDTN but available in the common/util/Sdnv library) requires AVX, AVX2, and the above "Fast SDNV" support.
- Fast CBOR encode/decode (BPv7) requires SSE and SSE2.
- Some optimized loads and stores for TCPCLv4 requires SSE and SSE2.
- Fast CRC32C (BPv7 and a storage hash function) requires SSE4.2.
- The HDTN storage controller will use BITTEST if available. If BITTEST is unavailable, it will use ANDN if BMI1 is available.

If LTP_RNG_USE_RDSEED is turned "On", this feature will be enabled if CMake finds support for this CPU instruction:

- An additional randomness source for LTP's random number generator requires RDSEED. This feature can be disabled for potentially faster LTP performance.

4.4 Storage Capacity Compilation Parameters

HDTN build environment sets two CMake cache variables by default: STORAGE_SEGMENT_ID_SIZE_BITS and STORAGE_SEGMENT_SIZE_MULTIPLE_OF_4KB.

- The 'STORAGE_SEGMENT_ID_SIZE_BITS' flag must be set to the recommended default, 32 or 64. It determines the size/type of the storage module's 'segment_id_t'. Setting the flag to 32-bit significantly decreases memory usage.
  - If this value is 32, the formula for the max segments (S) is given by $S = min(UINT32\_MAX, 64^6) =\sim$ 4.3 Billion segments since segment_id_t is a uint32_t. A segment allocator using 4.3 Billion segments uses about 533 MByte RAM), and multiplying by the minimum 4KB block size gives 17TB bundle storage capacity. Make sure to appropriately set the *totalStorageCapacityBytes* variable in the HDTN JSON config so that only the required amount of memory is used for the segment allocator.

  - If this value is 64, the formula for the max segments (S) is given by $S = min(UINT64\_MAX, 64^6) = \sim$ 68.7 Billion segments since segment_id_t is a uint64_t. Using a segment allocator with 68.7 Billion segments, when multiplying by the minimum 4KB block size gives $\sim$ 281TB bundle storage capacity.

- The flag $STORAGE\_SEGMENT\_SIZE\_MULTIPLE\_OF\_4KB$ must be set to an integer of 1 or greater. It determines the minimum increment of bundle storage based on the standard block size of 4096 bytes. (Note: One is the default and recommended.) Example:
  - If $STORAGE\_SEGMENT\_SIZE\_MULTIPLE\_OF\_4KB$‘ = 1, a $4KB * 1 = 4KB$ block size is used. A bundle size of 1KB would require 4KB of storage. A bundle size of 6KB would require 8KB of storage.
  - If $STORAGE\_SEGMENT\_SIZE\_MULTIPLE\_OF\_4KB = 2$, a $4KB * 2 = 8KB$ block size is used. A bundle size of 1KB would require 8KB of storage. A bundle size of 6KB would require 8KB of storage. A bundle size of 9KB would require 16KB of storage. If $STORAGE\_SEGMENT\_ID\_SIZE\_BITS = 32$, then bundle storage capacity could potentially double from $\sim 17TBto \sim 34TB$.

For information on how the Storage works, see ‘module/storage/doc/storage.pptx’ in the repository.

4.5 Build HDTN on Windows with its Dependencies

To build HDTN and its dependencies in Release mode and as shared libraries (shared .dll files for both HDTN and its dependencies), simply run the PowerShell script in `building_on_windows\hdtn_windows_cicd_unit_test.ps1` from any working directory. The working directory does not matter. Once finished, HDTN and its dependencies will be installed to `C:\hdtn_build_x64_release_vs2022` (suffix will be 2019 or 2017 if that's the Visual Studio compiler installed). The script will also run HDTN's unit tests after the build. Once completed, you will see the following message:
`"Remember, HDTN was built as a shared library, so you must prepend the following to your Path so that Windows can find the DLL's of HDTN and its dependencies:"`
It will print four directory locations to be added to your Path environmental variable to facilitate use of HDTN outside this PowerShell script.

- From the Windows Start Menu, type `"env"`, open `"Edit environmental variables for your account"`
- double click `Path`
- Add the four directories. (Omit the directory containing `hdtn_install\lib` if modifying HDTN source code within Visual Studio. You will later build and install your HDTN binaries within Visual Studio.)

- If you are a user of HDTN and you are NOT going to modify HDTN source code within Visual Studio, also add this directory to your Path: `C:\hdtn_build_x64_release_vs2022\hdtn_install\bin`
- Click `OK`
- Click `New`
- Add the following new variable: `HDTN_SOURCE_ROOT`
- Set the variable value to your source root (the folder that contains README.md). Example `C:\path\to\hdtn`
- Click `OK`
- Click `OK`

If you are a user of HDTN and you are NOT going to modify HDTN source code within Visual Studio, you can reference any of the .bat file example tests located in `HDTN_SOURCE_ROOT\tests\test_scripts_windows`. Note that these scripts were intended for developers, so you will have to modify the scripts, fixing any lines that reference `HDTN_BUILD_ROOT`, so, for example, if you see `"%HDTN_BUILD_ROOT%\common\bpcodec\apps\bpgen-async.exe"`, replace it with `bpgen-async.exe`. Also note that these .bat files reference config files located in `HDTN_SOURCE_ROOT\config_files`, so feel free to modify those .json configs to meet your needs.

HDTN Developers:

If you are a developer and you are going to modify HDTN source code within Visual Studio, you may delete the directory `C:\hdtn_build_x64_release_vs2022\hdtn_install` and continue on with the next set of instructions.

Launch Visual Studio 2022 and open HDTN as a project with these steps:

- File >> open >> cmake
- Open HDTN root CMakeLists.txt
- Make sure drop down configuration at the top is set to x64-Release. You may need to go to `Manage Configurations` if not.

Then click `Project >>` view `CMakeCache.txt` Add these lines (change _vs2022 directory suffix if different):

- BOOST_INCLUDEDIR:PATH=C:\hdtn_build_x64_release_vs2022\boost_1_78_0_install
- BOOST_LIBRARYDIR:PATH=C:\hdtn_build_x64_release_vs2022\boost_1_78_0_install\lib64
- BOOST_ROOT:PATH=C:\hdtn_build_x64_release_vs2022\boost_1_78_0_install
- OPENSSL_INCLUDE_DIR:PATH=C:\hdtn_build_x64_release_vs2022\openssl-1.1.1s_install\include

- OPENSSL_ROOT_DIR:PATH=C:\hdtn_build_x64_release_vs2022\openssl-1.1.1s_install
- libzmq_INCLUDE:PATH=C:\hdtn_build_x64_release_vs2022\libzmq_v4.3.4_install\include
- libzmq_LIB:FILEPATH=C:\hdtn_build_x64_release_vs2022\libzmq_v4.3.4_install\lib\libzmq-v143-mt-4_3_4.lib (note: may be v141 or v142)
- civetweb_INCLUDE:PATH=C:\hdtn_build_x64_release_vs2022\civetweb_v1.15_install\include
- civetweb_LIB:FILEPATH=C:\hdtn_build_x64_release_vs2022\civetweb_v1.15_install\lib\civetweb.lib
- civetwebcpp_LIB:FILEPATH=C:\hdtn_build_x64_release_vs2022\civetweb_v1.15_install\lib\civetweb-cpp.lib
- BUILD_SHARED_LIBS:BOOL=ON
- USE_WEB_INTERFACE:BOOL=ON

Then click `Project >> configure cache`

It's now time to set up additional environmental variables in order to be able to run the .bat file tests located in `HDTN_SOURCE_ROOT\tests\test_scripts_windows`:

- Right click on the open tab within Visual Studio titled `CMakeCache.txt` and then click `"Open Containing Folder"`
- Copy the path at the top of the Windows Explorer window
- From the Windows Start Menu, type `"env''`, open `"Edit environmental variables for your account''`
- Click `New`
- Add the following new variable: `HDTN_BUILD_ROOT`. The variable value will look something like `C:\Users\username\CMakeBuilds\17e7ec0d-5e2f-4956-8a91-1b324`
- Click `OK`
- Click `New`
- Add the following new variable: `HDTN_INSTALL_ROOT`. The variable value will look similar to `HDTN_BUILD_ROOT` except change "build" to "install".. something like `C:\Users\username\CMakeBuilds\17e7ec0d-5e2f-4956-8a91-1b32467252b`
- Click `OK`
- Double click Path variable, add the `HDTN_INSTALL_ROOT\lib` folder to your Path.. something like `C:\Users\username\CMakeBuilds\17e7ec0d-5e2f-4956-8a91-1b` This step is needed because HDTN is built as a shared library with multiple .dll files, so this step allows Windows to find those .dll files when running any HDTN binaries.

Relaunch Visual studio so that it get's loaded with the updated environmental variables. Now build HDTN:

- Build >> Build All
- Build >> Install HDTN
- Run `unit_tests.bat` located in `HDTN_SOURCE_ROOT\tests\test_scripts_windows`
- For a Web GUI example, run `test_tcpcl_fast_cutthrough_oneprocess.bat` and then navigate to `http://localhost:8086/web_gui.html` (note: to exit cleanly, do a ctrl-c in each cmd window before closing)

NOTE: Since CMake is currently configured to build HDTN as a shared library (because the CMake cache variable `BUILD_SHARED_LIBS` is set to `ON`), any time you make a source code change to HDTN, for it to be reflected in the binaries, don't forget to `Build >> Install HDTN` after the `Build >> Build All` step.

Setup Instructions for Developers Using Installed HDTN Libraries within their own Projects

HDTN utilizes modern CMake. When HDTN is installed, it installs the appropriate CMake Packages that can be imported. For an example of this use case, see `HDTN_SOURCE_ROOT\tests\unit_tests_import_installation\CMakeLists.txt` for a project that imports the libraries and headers from an HDTN installation and builds HDTN's unit tests from that installation.

4.6 Build HDTN on Raspberry Pi

To build HDTN on Raspberry Pi running Ubuntu (follow Ubuntu dependencies in Section 3.2 and also install `python3-zmq`):

- export HDTN_SOURCE_ROOT=/home/username/HDTN
  - Replace username with your username, (right of equals sign should be path to HDTN directory)
- cd $HDTN_SOURCE_ROOT
- mkdir build
- cd build
- cmake -DCMAKE_BUILD_TYPE=Release ..
- Open CMakeCache.txt in Editor (edit means edit variable already in file and add means add)
  - (add) Boost_USE_STATIC_LIBS:UNINITIALIZED=OFF
  - (edit) CMAKE_INSTALL_PREFIX:PATH=/user/local (User = your username)

    - (edit) USE_X86_HARDWARE_ACCELERATION:BOOL=OFF
    - (edit) LTP_RNG_USE_RDSEED:BOOL=OFF
    - (edit) CMAKE_CXX_FLAGS_RELEASE:STRING=-03 -DNDEBUG -fPIC
- Save and Leave Editor
- cd $HDTN_SOURCE_ROOT/common/util
- Open CMakeLists.txt in Editor
    - (Comment out(#)) src/CpuFlagDetection.cpp
- Save and Leave Editor
- cd $HDTN_SOURCE_ROOT/tests/unit_tests
- Open CMakeLists.txt in Editor
    - (Comment out(#)) ../../common/util/test/TestCpuFlagDetection.cpp
- Save and Leave Editor
- cd $HDTN_SOURCE_ROOT/build
- cmake -DCMAKE_BUILD_TYPE=Release ..
- make -j4
    - if using Pi without 4 core just use: [make]instead
- sudo make install

4.6.1 Debugging Errors/Problems

- For errors reporting something like: ¡cpuid.h¿ is not found for file HDTN/common/util/src/CpuFlagDetection.cpp
    - Double check CMakeList.txt edits
- For errors reporting something like: recompile with -fPIC
    - Double Check CMakeCache.txt edits
- For errors reporting ./runscript.sh not found
    - Run: export HDTN_SOURCE_ROOT=/home/user/HDTN
- For errors reporting similar to: no tcpdump
    - Run: sudo apt install tcpdump

5 Running HDTN

Note: Ensure your config files are correct, e.g., The outduct remotePort is the same as the induct boundPort, a consistant convergenceLayer, and the outducts remoteHostname is pointed to the correct IP adress. tcpdump can be used to test the HDTN ingress storage and egress. The generated pcap file can be read using wireshark: `sudo tcpdump -i lo -vv -s0 port 4558 -w hdtn-traffic.pcap`
In another terminal, run: `./runscript.sh`

Note: The Contact Plan, which lists future contacts for each node, is located under module/scheduler/src/contactPlan.json and includes source/destination nodes, start/end time, and the data rate. Based on the schedule in the contactPlan the scheduler sends events on link availability to Ingress and Storage. When the Ingress receives Link Available event for a given destination, it sends the bundles directly to egress. When the Link is Unavailable it sends the bundles to storage. Upon receiving Link Available event, Storage releases the bundle(s) for the corresponding destination. When a Link Available event is received, Storage stops releasing the bundles.

There are additional test scripts located under the directories test_scripts_linux and test_scripts_windows that can be used to test different scenarios for all convergence layers.

5.1 Directory Structure

common/	Common Libraries and utils
module/	HDTN modules
|– egress	CL adapter(s) that forwards bundle traffic
|– ingress	CL adapter(s) that accepts traffic in bundle format
|– storage	Stores bundles
|– scheduler	Sends events on link availability
|– router	Sends the optimal route and next hop to egress
|– hdtn_one_process	Combines the main processes into one HDTN process
|– udp_delay_sim	Proxy that simulates long delays
|– gui	Web interface for stats display and configuration
config_files/	HDTN config files
tests/	Example Test cases and experiments

5.2 Unit Tests

After building HDTN (see Section 4), unit tests can be run using the following command within the build directory:

```
./tests/unit_tests/unit-tests
```

5.3 Integrated Tests

After building HDTN (see Section 4), integrated tests can be run using the following command within the build directory:

```
./tests/integrated_tests/integrated-tests
```

6 Web User Interface

6.1 Running the Web User Interface

This repository comes equipped with code to launch a web-based user interface to display HDTN engine statistics. However, it relies on the dependency CivetWeb, which must be installed. CivetWeb can be found here:

$$https://sourceforge.net/projects/civetweb/$$

Download and extract the CivetWeb repository. Then follow these instructions to create a moveable library on Linux:

- cd civetweb
- make build
- make clean lib WITH_WEBSOCKET=1 WITH_CPP=1
- Move the CivetServer.h and civetweb.h (located at civetweb/include/) files from the Civetweb directory to the civetweb_INCLUDE location for Linux (hdtn/external/include/).
- Move the libcivetweb.a file to the corresponding civetweb_LIB location (hdtn/external/lib).

After compiling HDTN open the CMakeCache file in the build directory and make the following edits under the "USE_WEB_INTERFACE" section:

- Set "USE_WEB_INTERFACE:BOOL" to ON

Now anytime that HDTNOneProcess is ran, the web page will be accessible at http://localhost:8086

6.2 Statistics Page

This page displays real-time telemetry of HDTN. The top three boxes display the current Data Rate in Mega-Bits Per Second, the Average Data Rate, and the Maximum Data Rate reached. These are measured in the Ingress module.

Beneath are three graphs. The first two display the data rate of the Ingress and Egress Modules in Mega-Bits Per Second. The third graph is a pie chart displaying the location of data bundles received by Ingress - either sent to the Storage module or directly to Egress. If a bundle is sent from Storage to Egress, it will be measured on the pie chart as having gone to Egress.

Beneath the graphs, cards display statistics for different parts of HDTN. At this time only Ingress and Egress are displayed.

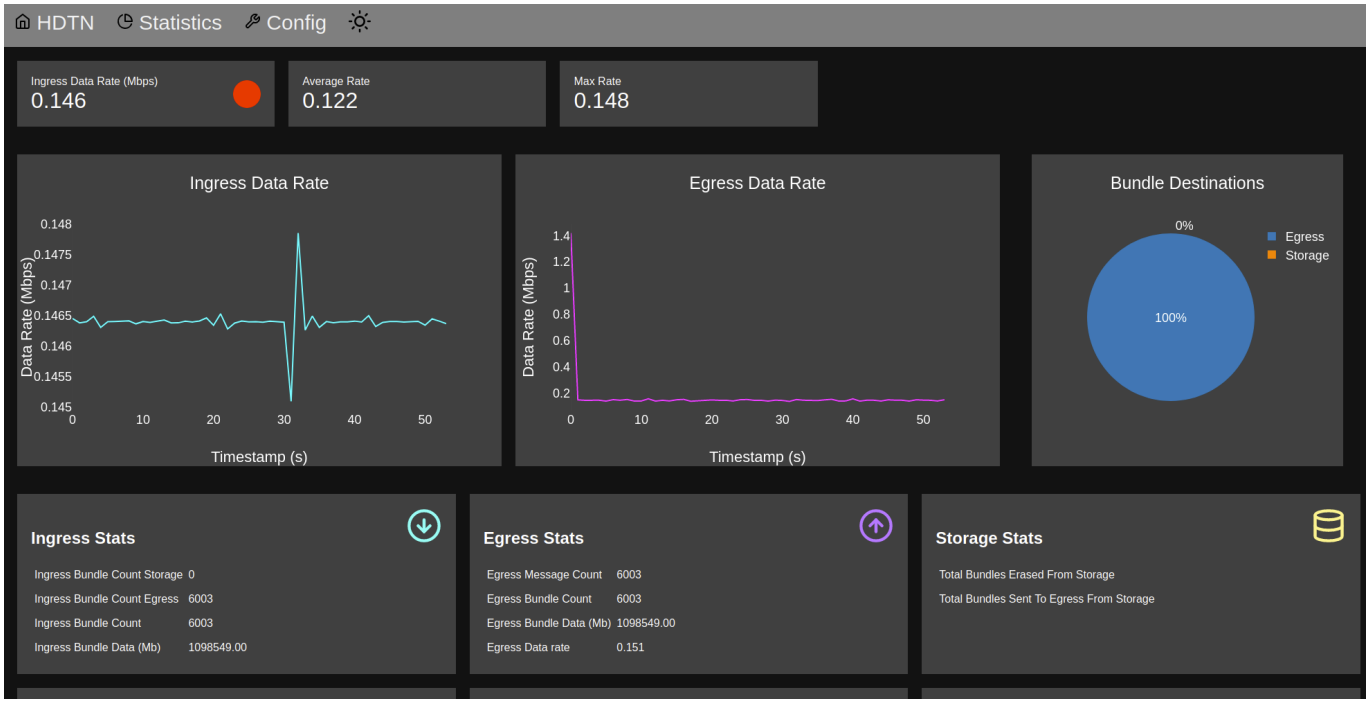

Figure 2: Statistics Page of the Web Interface

6.3 Config Page

This page is not built yet.

7 Simulations

HDTN can be simulated using DtnSim, a simulator for delay tolerant networks built on the OMNeT++ simulation framework. Use the "support-hdtn" branch of DtnSim which can be found in the official DtnSim repository. HDTN simulation with DtnSim has only been tested on Linux (Debian and Ubuntu). Follow the readme instructions for HDTN and DtnSim to install the software. Alternatively, a pre-configured Ubuntu VM is available for download here (the username is hdtnsim-user, and the password is grc). More Details about the installation steps can be found here

8 HDTN Applications

8.1 BpGen

BPGen is a tool that generates bundles of any specified size, and it is intended to be used with its receiving tool BpSink. It is not a component of HDTN, but it does share the same codebase and libraries as HDTN; hence it can use any of the convergence layers available to HDTN. Additionally, the config file that BpGen uses shares the outductsConfig part of the HDTN config file. If a user desires BPv6 custody transfer support, BpGen supports passing in an additional separate config file containing the inductsConfig part of an HDTN config file; this additional config file is necessary when using unidirectional (non-bidirectional) convergence layers. The BpGen source code is very small because it derives from a helper C++ class called BpSourcePattern and overrides two virtual methods. Users who wish to write a utility that generates bundles as fast as possible and then terminates can model after the BpGen source code.

8.2 BPSink

BPSink tool receives and validates the bundles sent from BPGen (and possibly bundles that were forwarded from the HDTN Egress). It is not a component of HDTN, but it does share the same codebase and libraries as HDTN; hence it can use the convergence layers available to HDTN. Additionally, the config files that BpSink uses shares the inductsConfig part of the HDTN config file. If a user desires BPv6 custody transfer support, BpSink supports passing in an additional separate config file containing the outductsConfig part of an HDTN config file; this additional config file is necessary when using unidirectional (non-bidirectional) convergence layers. The BpSink source code is very small because it derives from a helper C++ class called BpSinkPattern and overrides one virtual method; any users that want to write their own utility that receives

bundles indefinitely and terminates cleanly with a SIGINT signal can simply model after the BpSink source code.

8.3 BpSendFile

BpSendFile is a tool that sends either a single file or a directory of files (with recursion), and it takes those file(s) and breaks them into max specified size bundles, and it is intended to be used with its receiving tool BpReceiveFile. It can remain open after all files have been sent to monitor those directories (up to a user-specified max recursion depth) for newly added files. It (like BpGen) is not a component of HDTN, but it does share the same codebase and libraries as HDTN; hence it can use any of the convergence layers available to HDTN. The config files used are the same as BpGen. The BpSendFile source code is very small because it derives from a helper C++ class called BpSourcePattern and overrides three virtual methods; any users that want to write their utility that generates bundles as fast as possible and then remains open and episodically sends new bundles can model after the BpSendFile source code.

8.4 BpReceiveFile

BpReceiveFile tool receives the bundles sent from BpSendFile in any order and reassembles the file fragments, closes the file when all file fragments have been received and writes them to a user-specified directory. It is not a component of HDTN, but it does share the same codebase and libraries as HDTN; hence it can use any of the convergence layers available to HDTN. The config files used and the derived C++ classes used are the same as BpSink.

8.5 BPing

BPing is a tool that generates ping bundles intended to be used with any bundling agent that supports an echo service. HDTN has an echo service with a service number specified in its config as myBpEchoServiceId. All HDTN apps (BpSink and BpReceiveFile) that inherit from BpSinkPattern have an echo service number of 2047. BPing is not a component of HDTN, but it does share the same codebase and libraries as HDTN; hence it can use any of the convergence layers available to HDTN. The config files used are the same as BpGen. BPing does not support custody transfer, but it must also use the same config file BpGen would use for BPv6 custody transfer support in order for BPing to receive an echo bundle back; this additional config file is necessary when using unidirectional (non-bidirectional) convergence layers. The BPing source code is very small because it derives from a helper C++ class called BpSourcePattern and overrides two virtual methods; any user that wants to write their own utility that generates a single bundle, waits for a response, and then either terminates or continues can simply model after the BPing source code.

9 Run Script

One run script file (in JSON format) is required per each instance of HDTN. Each file starts with assigning path variables, starting up Egress (the outduct), and the bpsink (the method of where bundles will be received). Each file ends with starting up Ingress, assigning bpgen (the process by which bundles can be generated), and a cleanup procedure (optional). Within each file, two different setups can be found depending if the setup is cut-through, i.e. by-passes Storage or utilizes Storage and, in turn, the Scheduler. The cut-through option requires the link to be up and no BPv6 custody. Note: To avoid starting up Egress and Ingress separately, a method called hdtn-one-process combines the two with one command.

9.1 Path Variables

The Path Variables section of each run script file points to the locations of all required configuration files. This way, one needs only to edit this section instead of each section if only changing a configuration file. The path variables contain locations of:

- config_files
- hdtn_config
- sink_config
- gen_config

The section ends with a *cd* $*HDTN_SOURCE_ROOT* an already declared variable in your linux path. This command is here if the run script is not in the HDTN source root directory (where HDTN runs).

9.2 bpsink

A typical instantiation of bpsink is:

```
./build/common/bpcodec/apps/bpsink-async --my-uri-eid=ipn:2.1
--inducts-config-file=$sink_config &
```

The command has three main parts: (1) the location of the bpsink code within HDTN, (2) the endpoint ID number, and (3) the inducts configuration file location. The ipn:2.1 shows, in this particular case, the receiving node is the second endpoint. The $sink_config response ties to the respectable Path Variable. For more information about the sink configuration file see section 10.2. To end bpsink, it is customary to wait 3 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 3

9.3 Egress

A typical instantiation of Egress is:
`./build/module/egress/hdtn-egress-async --hdtn-config-file=$hdtn_config &`

The command has two main parts: (1) the location of the egress code within HDTN and (2) the HDTN configuration file that the egress code requires. For more information about the HDTN configuration file see section 10.1. To end Egress, it is customary to wait 3 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 3

9.4 Scheduler

A typical instantiation of the scheduler is: `./build/module/scheduler/hdtn-scheduler --contact-plan-file=contactPlan.json --hdtn-config-file=$hdtn_config &`

The command has three main parts: (1) the location of the Scheduler code within HDTN, (2) the contact plan and (3) the same HDTN configuration file that the Egress required. For more information about the HDTN configuration file see section 10.1. To end the Scheduler, it is customary to wait 1 seconds to initialize before the configuration file starts up the next section, i.e. *sleep* 1

9.5 Router

A typical instantiation of the Router is: `./build/module/router/hdtn-router --contact-plan-file=contactPlan.json --dest-uri-eid=ipn:2.1 --hdtn-config-file=$hdtn_config &` The command has four main parts: (1) the location of the router code within HDTN, (2) the contact plan (3) the destination endpoint ID and (4) the same HDTN configuration file that the Egress required. For more information about the HDTN configuration file see section 10.1. To end the Scheduler, it is customary to wait 1 seconds to initialize before the configuration file starts up the next section, i.e. *sleep* 1
An example of a Routing scenario test with 4 HDTN nodes was added under $HDTN_SOURCE_ROOT/test/test_scripts_linux/Routing_Test

9.6 Ingress

A typical instantiation of Ingress is:
`./build/module/ingress/hdtn-ingress --hdtn-config-file=$hdtn_config &`

The command has two main parts: (1) the location of the Ingress code within HDTN and (2) the HDTN configuration file that the ingress code requires. For more information about the HDTN configuration file see section 10.1. To end Ingress, it is customary to wait 3 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 3

9.7 Storage

A typical instantiation of Storage is:
```
./build/module/storage/hdtn-storage --hdtn-config-file=$hdtn_config
&
```
The command has two main parts: (1) the location of the storage code within HDTN and (2) the HDTN configuration file that the storage code requires. For more information about the HDTN configuration file see section 10.1. To end Storage, it is customary to wait 3 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 3

9.8 bpgen

A typical instantiation of bpgen is:
```
./build/common/bpcodec/apps/bpgen-async --bundle-rate=100 --my-uri-eid=ipn:1.1
--dest-uri-eid=ipn:2.1 --duration=40 --outducts-config-file=$gen_config
&
```
The command has six main parts: (1) the location of the bpgen application code within HDTN, (2) the bundle rate, (3) the endpoint ID, (4) the destination endpoint ID, (5) the duration, and (6) the gen configuration file that the bpgen code requires. For more information about the gen configuration file see section 10.3. In the example instantiation, the bundle rate was designated for 100 bps with the bpgen's endpoint ID being the first node (ipn:1.1) and is sending to the second node (ipn:2.1), which matches the bpsink's endpoint ID. The duration value is in seconds, and in this case, 40 seconds. To end bpgen, it is customary to wait 8 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 8

9.9 bping

A typical instantiation of bping is:
```
./build/common/bpcodec/apps/bping --my-uri-eid=ipn:1.1 --dest-uri-eid=ipn:2.2047
--outducts-config-file=$ping_config & &
```
The command has six main parts: (1) the location of the bping application code within HDTN, (2) the endpoint ID, (4) the destination endpoint ID, and (4) the configuration file that the bping code requires is the same as BpGen config file. For more information about the gen configuration file see section 10.3. In the example instantiation, node is sending ping bundles to the node 2 (using the echo service number 2047).

9.10 CleanUp

If HDTN only needs to run for a certain amount of time and then end, add a line under all other sections (minus the path variables) after the instantiation command in the format of $< SectionName > \_PID =$ \$! For example, after bpgen's instantiation, the cleanup command will be: bpgen_PID=$!

Within the clean-up section, wait for HDTN to run. Then, from the bottom to the top of the configuration file sections, end them via format of *kill* $-2\ \$ < PID\ name >$. A wait statement for at least 2 seconds between each kill command is included. Clean-up script example:

```
sleep 30
echo "\nkilling bpgen..." && kill -2 $bpgen_PID
sleep 2
echo "\nkilling HDTN storage..." && kill -2 $storage_PID
sleep 2
echo "\nkilling HDTN ingress..." && kill -2 $ingress_PID
sleep 2
echo "\nkilling scheduler..." && kill -9 $scheduler_PID
sleep 2
echo "\nkilling egress..." && kill -2 $egress_PID
sleep 2
echo "\nkilling bpsink..." && kill -2 $bpsink_PID
```

This example will run HDTN for 30 seconds before closing HDTN.

9.11 HDTN One Process

A typical instantiation of hdtn-one-process is:

```
./build/module/hdtn_one_process/hdtn-one-process --hdtn-config-file=$hdtn_config
&
```

The command has two main parts: (1) the location of the hdtn-one-process code within HDTN and (2) the HDTN configuration file that the hdtn-one-process code requires. For more information about the HDTN configuration file see section 10.1. To end hdtn-one-process, it is customary to wait 10 seconds to initialize before the configuration file starts the next section, i.e. *sleep* 10

Note: When using the hdtn-one-process, the runscript does not need to instantiate the Egress, Storage, and Ingress separately.

10 Config Files

10.1 hdtn_config

The typical HDTN configuration file instantiation can be seen below. All values are default and changeable. More information on each line can be seen bulleted.

```
"hdtnConfigName":  my hdtn config,
```

- User description of config file

```
"userInterfaceOn":  true,
```

- When compiled determines if the interface is displayed

```
"mySchemeName":  "unused_scheme_name",
```

- DTN scheme name
- Deprecated, still needs to be defined

```
"myNodeId":  10,
```

- Node running ID
- Must be an integer

"myBpEchoServiceId": 2047,
- Service number to ping if user wants to ping HDTN
- Must be an integer

"myCustodialSsp": "unused_custodial_ssp",
- Custodial scheme specific part
- Deprecated, still needs to be defined

"myCustodialServiceId": 0,
- Service ID where custody reports are sent to HDTN
- Must be an integer

"isAcsAware": true,
- Aggregate Custody Signals (ACS)
- Specifies if HDTN is to use ACS
- Must be a Boolean

"acsMaxFillsPerAcsPacket": 100,
- How many custody signals to be packed into one ACS packet
- Must be an integer

"acsSendPeriodMilliseconds": 1000,
- Aggregation time in ms
- Must be an integer

"retransmitBundleAfterNoCustodySignalMilliseconds": 10000,

"maxBundleSizeBytes": 10000000,
- The maximum size of the bundle HDTN can receive or send in Bytes
- NOTE: if the bundle is larger than this, the bundle will be dropped.

"maxIngressBundleWaitOnEgressMilliseconds": 2000,
- During Cut-through, if egress cannot finish a bundle within this time it will give up cut-through and send to storage; in ms

"maxLtpReceiveUdpPacketSizeBytes": 65536,
- Maximum packet size in bytes that can be received by HDTN
- Set to the largest datagram the protocol will see on the network.
- 65536 is the typical max size of a local UDP packet will support
- This is a Don't Care if not using LTP

"zmqIngressAddress": "localhost",
- IP or hostname of the machine running the Ingress module of HDTN

"zmqEgressAddress": "localhost",
- IP or hostname of the machine running the Egress module of HDTN

"zmqStorageAddress": "localhost",
- IP or hostname of the machine running the storage module of HDTN

"zmqRegistrationServerAddress": "localhost",
- Deprecated, still needs to be defined

"zmqSchedulerAddress": "localhost",
- IP or hostname of the machine running the scheduler module of HDTN

"zmqRouterAddress": "localhost",
- IP or hostname of the machine running the router module of HDTN

"zmqBoundIngressToConnectingEgressPortPath": 10100,
- ZMQ bound TCP port of the Ingress module for internal messages sent from Ingress to Egress
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqConnectingEgressToBoundIngressPortPath": 10160,
- ZMQ bound TCP port of the Ingress module for internal messages sent from Egress to Ingress
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqConnectingEgressBundlesOnlyToBoundIngressPortPath": 10161,
- ZMQ bound TCP port of the Ingress module for internal TCPCL opportunistic bundles sent from Egress to Ingress
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqConnectingEgressToBoundSchedulerPortPath": 10162,
- ZMQ bound TCP port of the scheduler module for internal link down messages sent from Egress to Scheduler
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqBoundIngressToConnectingStoragePortPath": 10110,
- ZMQ bound TCP port of the Ingress module for internal messages sent from Ingress to Storage
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqConnectingStorageToBoundIngressPortPath": 10150,
- ZMQ bound TCP port of the Ingress module for internal messages sent from Storage to Ingress
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqConnectingStorageToBoundEgressPortPath": 10120,
- ZMQ bound TCP port of the Egress module for internal messages sent from Storage to Egress

- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqBoundEgressToConnectingStoragePortPath": 10130,
- ZMQ bound TCP port of the Egress module for internal messages sent from Egress to Storage
- NOTE: This value is unused when using hdtn-one-process; still needs to be defined.
- NOTE: TCP is unidirectional in ZMQ
- Must be an integer

"zmqRegistrationServerPortPath": 10140,
- Deprecated, still needs to be defined
- Must be an integer

"zmqBoundSchedulerPubSubPortPath": 10200,
- ZMQ bound port of the scheduler ZMQ pub-sub socket
- Must be an integer

"zmqBoundRouterPubSubPortPath": 10210,
- ZMQ bound port of the router ZMQ pub-sub socket
- Must be an integer

"zmqMaxMessagesPerPath": 5,
- Between HDTN modules, this is the maximum ZMQ queue size (number of bundles) on each final destination node ID(s) concurrently; i.e. if the final destination nodes are 2 and 3, it will send bundles up to this number at once to each of the nodes.
- NOTE: this relates to the pipelining of bundles. Critical for when the receive rate is larger than the transmit rate.
- Must be an integer

"zmqMaxMessageSizeBytes": 100000000,
- Maximum bundle size that can be sent within the ZMQ messaging in Bytes.
- Must be an integer

```
"inductsConfig": {
  "inductConfigName": "myconfig",
  "inductVector": [
      {
          "name": "stcp_ingress",
          "convergenceLayer": "stcp",
          "boundPort": 4556,
          "numRxCircularBufferElements": 200,
      }
   ]
},
```

- inductConfigName and name are for user comment

- Can choose within the convergence layer: stcp, tcpcl_v3, tcpcl_v4, udp, ltp_over_udp
- boundPort and numRxCircularBufferElements must be integers
- NOTE: numRxCircularBufferElements differs for each convergence layer. STCP this represents the number of bundles to buffer up; TCPCL this is the number of bundles or data fragments; LTP this is the number of UDP packets; UDP this is the number of packets/bundles.

```
"outductsConfig":  {
   "outductConfigName":  "myconfig",
   "outductVector":  [
       {
           "name":  "stcp_egress",
           "convergenceLayer":  "stcp",
           "nextHopNodeId":  2,
           "remoteHostname":  "localhost",
           "remotePort":  4558,
           "bundlePipelineLimit":  50,
           "finalDestinationEidUris":  [
              "ipn:2.1"
            ],
       }
    ]
 },
```

- outductConfigName and name are for user comment
- Can choose within the convergence layer: stcp, tcpcl_v3, tcpcl_v4, udp, ltp_over_udp
- remoteHostname is the IP or hostname that HDTN is sending bundles to. remotePort is the port HDTN is sending bundles to.
- Final destination IDs can be multiple or one IPN URIs. IPN service number can be an * for a service wildcard.
- nextHopNodeID, remotePort, and bundlePipelineLimit must be integers.

```
"storageConfig":  {
   "storageImplementation":  "asio_single_threaded",
   "tryToRestoreFromDisk":  false,
   "autoDeleteFilesOnExit":  true,
   "totalStorageCapacityBytes":  8192000000,
   "storageDiskConfigVector":  [
       {
           "name":  "d1",
           "storeFilePath":  ".\/store1.bin"
       },
       {
           "name":  "d2",
           "storeFilePath":  ".\/store2.bin"
```

```
        },
    ]
},
```

- storageImplementation has 2 options: asio_single_threaded and stdio_multi_threaded. Default to asio_single_threaded.
- tryToRestoreFromDisk: if the bundle storage was left used, when HDTN reloads it can restore the state
- autoDeleteFilesOnExit: tells HDTN, when cleanly closed, to delete or save all storage bundles
- totalStorageCapacityBytes: storage module capacity or quota for bundles, in Bytes. NOTE: each storage item in the storageDiskConfigVector must be able to hold totalStorageCapacityBytes divided by the number items in the total storageDiskConfigVector
- storageDiskConfigVector is a striping scheme similar to RAID 0 but not using RAID itself. NOTE: can have unlimited storage vector siz i.e. number of storeFilePath(s).

Miscellaneous notes for the HDTN configuration file:

- Depending on the convergence layer there may be additions to the "inductVector", "outductVector". See section 12 for details.
- Ingress, Egress, and/or storage are optional additions to the HDTN configuration file depending on the HDTN node need.

10.2 sink_config

A typical sink configuration file includes an "inductConfigName" and an "inductVector". The "inductVector" requires the name, convergence layer, the bound port number, and the number of received circular buffer elements. For details about this section, see 10.1 since the bp_sink config file is a copy of the inductConfigName/inductVector of the hdtn_config file. This is because the bp_sink config file goes to the bp sink application detailed in section **??**. Note: Depending on the convergence layer there may be additions to the "inductVector". See section 12 for details. Structure example:

```
"inductsConfig": {
  "inductConfigName": ''myconfig",
  "inductVector": [
      {
          "name": "stcp_ingress",
          "convergenceLayer": "stcp",
          "boundPort": 4556,
          "numRxCircularBufferElements": 200,
      }
  ]
},
```

10.3 gen_config

A typical gen configuration file includes an "outductConfigName" and an "outductVector". The "outductVector" requires: the name, convergence layer, next hop, remote hostname, remote port, bundle pipline limit, and the final destination endpoint ID. For details about this section see 10.1 since the gen_config file is a copy of the outductConfigName/outductVector of the hdtn_config file. This is due to the gen_config file going to the bp source application detailed in section **??** Note: depending on the convergence layer there may be additions to the "outductVector". See section 12 for details. Structure example:

```
{
  "outductConfigName":  "myconfig",
  "outductVector":  [
     {
        "name":  "bpgen",
        "convergenceLayer":"tcpcl_v3",
        "nextHopNodeId":  10,
        "remoteHostname":  "localhost",
        "remotePort":  4556,
        "bundlePipelineLimit":  5,
        "finalDestinationEidUris":  [
          "ipn:1.1",
          "ipn:2.1",
          "ipn:3.1"
         ],
      }
   ]
},
```

11 Contact Plans

The contact plan is a JSON file which has a list of all forthcoming contacts for all the nodes in the network. The contact plans are accessible under `HDTN/module/scheduler/src`

11.1 JSON Fields

`"contact":  1`
- Identification number of the contact.
- Integer

`"source":  10`
- Source node number for that contact.
- Integer

`"dest":  2`
- Destination node number for that contact or next hop.
- Integer

`"finalDestination":  2`

- The end node where bundles are destined via that contact.
- Integer

`"startTime": 25`
- The time after which the link is UP for that contact
- Integer

`"endTime": 38`
- The time after which link will be DOWN for that contact
- Integer

`"rate": 1000`
- The data rate in Mbits/s
- Integer

`"owlt": 1`
- One Way Light Time
- Integer

12 Convergence Layers and Routing Protocols

12.1 Overview of Compatible Convergence Layers

BP (Bundle Protocol) is used in space and other areas that experience intermittent connectivity and/or long latencies. The BP software included two major revisions between version 6 and version 7. The HDTN software can understand both versions depending if it is being used with current and legacy assets or future assets slated to use BPvs7.

TCPCL (Transmission Control Protocol (TCP) Convergence Layer) is utilized in space applications for users using Internet Protocol (IP). In addition, this convergence layer provides a bridge from the Bundle Protocol(BP), if hops are required, to get to the destination. TCP requires acknowledgment before the message is sent and, for space, can be inefficient. HDTN currently uses TCP version 4.

UDP (User Datagram Protocol) is like TCP, except UDP does not require acknowledgment before sending a message. This protocol is over IP but can be used with BP to enable hops, like TCP.

LTP (Licklider Transmission Protocol) is the main transport layer for BP. Therefore, no matter how many hops or delays the message goes through to arrive at its destination, the message will remain intact. The HDTN's LTP is compatible with LTP version 6 with and without custody.

STCP....

12.2 Additions to Config Files

Each convergence layer has additional "inductVector" and "outductVector" configuration fields. These additions apply to all config files that utilize "inductVector" and "outductVector", e.g. hdtn_config, sink_config, and gen_config. The additions are listed in the sections below.

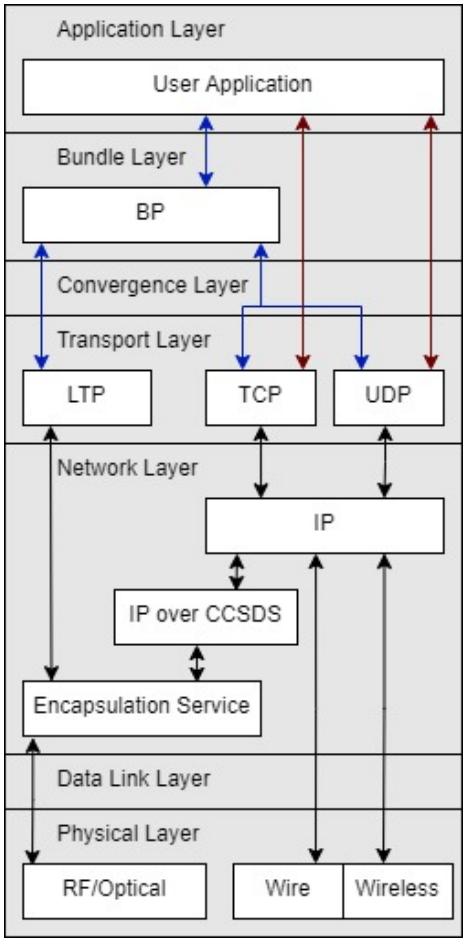

Figure 3: Simplified Protocol Stack

12.2.1 TCPCLv3

Common induct and outduct fields:
`"keepAliveIntervalSeconds": 15`
- This is the minimum interval, in seconds, to negotiate as the Session Keepalive. See RFC7242 Section 5.6.
- Integer

`"tcpclV3MyMaxTxSegmentSizeBytes": 200000`
- This is the maximum segment size, in bytes, to use for transmitting data segments.
- Integer

Induct fields:
`"numRxCircularBufferBytesPerElement": 100`
- This is the maximum size, in bytes, of each element in the circular receive buffer.
- Integer

Outduct fields:
`"tcpclAllowOpportunisticReceiveBundle": false`
- This is whether to allow receiving opportunistic bundles.
- Boolean

12.2.2 TCPCLv4

Common induct and outduct fields:
`"keepAliveIntervalSeconds": 15`
- This is the minimum interval, in seconds, to negotiate as the Session Keepalive. See RFC7242 Section 5.6.
- Integer

`"tcpcl4MyMaxTxSegmentSizeBytes": 200000`
- This is the maximum segment size, in bytes, to use for transmitting data segments.
- Integer

`"tlsIsRequired": false`
- This is whether TLS (Transport Layer Security) is required.
- Boolean

Induct fields:
`"numRxCircularBufferBytesPerElement": 100`
- This is the maximum size, in bytes, of each element in the circular receive buffer.
- Integer

`""tcpclV4MyMaxRxSegmentSizeBytes": 20000,`
- This is the maximum segment size, in bytes, to use for transmitting data segments.

- Integer

"certificatePemFile": "C:hdtn_ssl_certificatescert.pem"
- This is a path to the file containing the certificate for SSL (Secure Socket Layer).

"privateKeyPemFile": "C:hdtn_ssl_certificatesprivatekey.pem"
- This is a path to the file containing the private key for SSL.

"diffieHellmanParametersPemFile": "C:hdtn_ssl_certificatesdh4096.pem"
- This is a path to the file containing the Diffie-Hellman parameters for TLS.

Outduct fields:

"tryUseTls": false
- This is whether TLS is required.
- Boolean

"useTlsVersion1_3": false
- This is whether TLS version 1.3 is required. If not specified, version 1.2 will be used.
- Boolean

"doX509CertificateVerification": false
- This is whether to do X.509 certificate validation is required.
- Boolean

"verifySubjectAltNameInX509Certificate": false
- This is whether to verify the subject alternative name in the X.509 certificate is required.
- Boolean

"certificationAuthorityPemFileForVerification": "C:hdtn_ssl_certificatescert.pem
- This is a path to the file containing the certificate authority.

12.2.3 UDPCL

Induct fields:

"numRxCircularBufferBytesPerElement": 100
- This is the maximum size, in bytes, of each element in the circular receive buffer.
- Integer

Outduct fields:

"udpRateBps": 800000
- This is the UDP bitrate.
- Integer

12.2.4 LTPCL

Common induct and outduct fields:

"clientServiceId": 1
- This is the ID of the client service.

- Integer

"ltpDataSegmentMtu": 1360

- This is the maximum size of the data portion (excluding LTP headers and UDP headers and IP headers) of an LTP sender's Red data segment being sent. Set this low enough to avoid exceeding ethernet MTU to avoid IP fragmentation.
- Integer

"ltpMaxRetriesPerSerialNumber": 500

- This is the maximum number of retries/resends of a single LTP packet with a serial number before the session is terminated.
- Integer

"ltpMaxUdpPacketsToSendPerSystemCall": 1

- This is the maximum number of UDP packets to send per system call.
- Integer

"ltpRandomNumberSizeBits": 64

- This is whether to use a 32 or 64 bit random number is required.
- Integer (32 or 64)

"oneWayLightTimeMs": 1000

- This is the one way light time. Round trip time (retransmission time) is computed by (2 * (oneWayLightTime + oneWayMarginTime)).
- Integer

"oneWayMarginTimeMs": 200

- This is the one way margin (packet processing) time. Round trip time (retransmission time) is computed by (2 * (oneWayLightTime + oneWayMarginTime)).
- Integer

"remoteLtpEngineId": 20

- This is the ID of the remote LTP engine.
- Integer

"thisLtpEngineId": 10

- This is the ID of this LTP engine.
- Integer

Induct fields:

"ltpMaxExpectedSimultaneousSessions": 500

- This is the number of expected simultaneous LTP sessions for this engine.
- Integer

"ltpRemoteUdpHostname": "localhost"

- This is the remote IP address or hostname.

"ltpRemoteUdpPort": 4556

- The remote UDP port
- Integer

"ltpRxDataSegmentSessionNumberRecreationPreventerHistorySize": 1000

- This is the number of recent LTP receiver history of session numbers to remember. If an LTP receiver's session has been closed and it receives

a session number, within the history, the receiver will refuse the session to prevent a potentially old session from being reopened, which has been known to happen with IP fragmentation enabled.
- Integer

"preallocatedRedDataBytes": 200000
- ESTIMATED_BYTES_TO_RECEIVE_PER_SESSION
- The number of Red data contiguous bytes to initialized on a receiver. Make this large enough to accommodate the max Red data size so that the LTP receiver does not have to reallocate, copy, and/or delete data while it is receiving Red data. Make this small enough so that the system does not have to allocate too much extra memory per receiving session.
- Integer

Outduct fields:

"ltpCheckpointEveryNthDataSegment": 0
- This enables accelerated retransmission for an LTP sender by making every Nth UDP packet a checkpoint.
- Integer

"ltpMaxSendRateBitsPerSecOrZeroToDisable": 0
- This is rate limiting UDP send rate in bits per second.
- Integer

"ltpSenderBoundPort": 1113
- This is the bound port of the LTP sender.
- Integer

"ltpSenderPingSecondsOrZeroToDisable": 15
- This is the number of seconds between LTP session sender pings during times of zero data segment activity. An LTP ping is defined as a sender sending a cancel segment of a known non-existent session number to a receiver, in which the receiver shall respond with a cancel ACK in order to determine if the link is active.
- Integer

12.2.5 STCPCL

Common induct and outduct fields:

"keepAliveIntervalSeconds": 17
- This is the minimum interval, in seconds, to negotiate as the Session Keepalive.
- Integer

13 Test Configurations

13.1 TCP Loopback Test

To run this simple Loopback Test, from the HDTN source directory run the command:

```
./runscript.sh
```

This works by running four modules for about 30 seconds:

- BPGen - Generates the bundles and sends them to the Ingress module.
- Scheduler - This reads the contact plan and sends link-up events to the Ingress and Storage modules (note: storage is not used in this test).
- HDTN One Process - Launches the modules for HDTN as a single process. Since this is a cutthrough test, only Ingress and Egress are used.
- BPSink - Receives the bundle data from Egress.

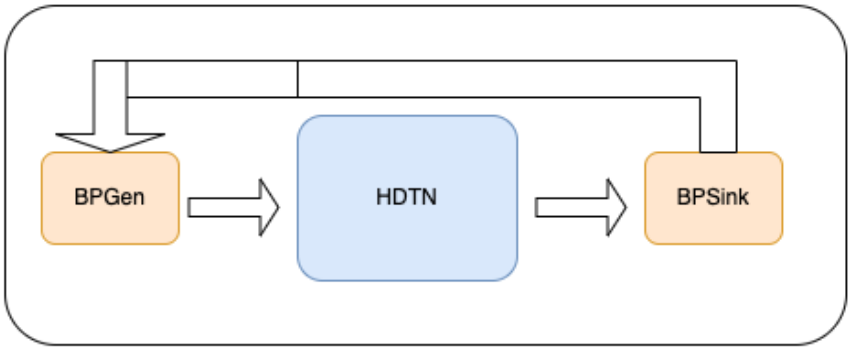

Figure 4: HDTN Loopback test.

13.2 Two Node LTP Test

This test relies on having two machines running HDTN: the sender and the receiver. The sender will run BPGen, scheduler and HDTN One Process. The receiver will run BPSink, scheduler and HDTN One Process.

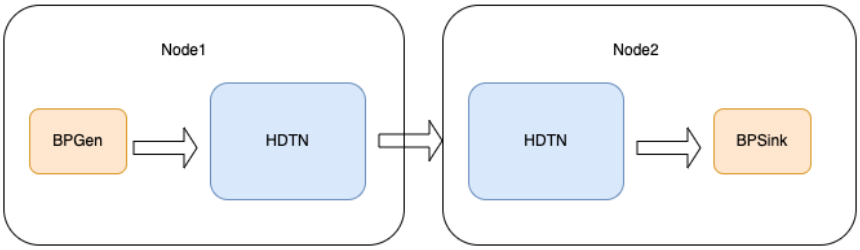

Figure 5: HDTN 2 nodes test.

13.3 Four Nodes STCP Test

This test relies on having 4 machines running HDTN and is using the router module. Node 1 will run BPGen router scheduler and HDTN One Process. Node 2 and 3 will run router scheduler and HDTN One Process. The final destination Node 4 will run router scheduler HDTN One Process and BPSink. In Node 1 HDTN config outductVector the next hop is configured to node 3 originally. After the router computes the optimal route to the final destination the outduct will select node 2 as next hop instead. At initialization, the HDTN json config file for each node has all possible next hops. If we have multiple hops leading to the same final destination, only one Outduct should be initialized with the final destinations vector, and the other next hops Outducts should be dormant, ie having no values initialized in their final destinations json fields. Router will compute the best route and send an event to Egress to update the Outduct to use the nextHop for that optimal route leading to the final destination.

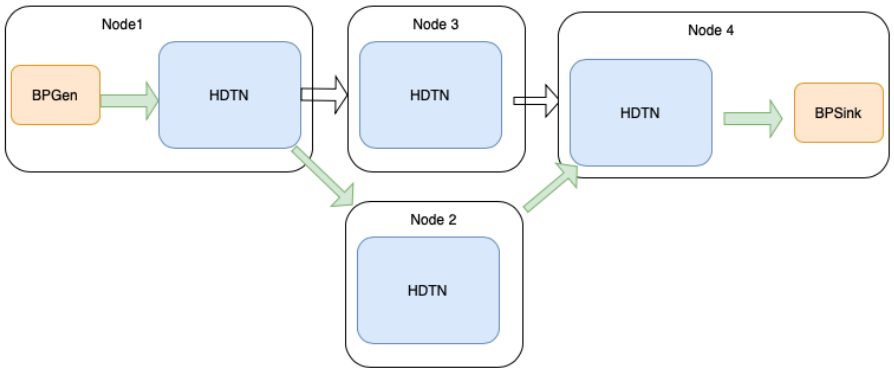

Figure 6: HDTN Routing test.

14 Troubleshooting

By default HDTN is built in Release mode. To enable DEBUG mode during build use: cmake .. -DCMAKE_BUILD_TYPE=Debug

14.1 Logging

Logging is controlled by CMake cache variables:

- LOG_LEVEL_TYPE controls which messages are logged. The options, from most verbose to least verbose, are TRACE, DEBUG, INFO, WARNING, ERROR, FATAL, and NONE. All log statements using a level more verbose than the provided level will be compiled out of the application. The default value is INFO.

- LOG_TO_CONSOLE controls whether log messages are sent to the console. The default value is ON.

- LOG_TO_ERROR_FILE controls whether all error messages are written to a single error.log file. The default value is OFF.

- LOG_TO_PROCESS_FILE controls whether each process writes to their own log file. The default value is OFF.

- LOG_TO_SUBPROCESS_FILE controls whether each subprocess writes to their own log file. The default value is OFF.

15 Notes

15.1 TLS Support for TCPCL Version 4

TLS Versions 1.2 and 1.3 are supported for the TCPCL Version 4 convergence layer. The X.509 certificates must be version 3 in order to validate IPN URIs using the X.509 "Subject Alternative Name" field. HDTN must be compiled with ENABLE_OPENSSL_SUPPORT turned on in CMake. To generate (using a single command) a certificate (which is installed on both an outduct and an induct) and a private key (which is installed on an induct only), such that the induct has a Node Id of 10, use the following command:

```
openssl req -x509 -newkey rsa:4096 -nodes -keyout privatekey.pem
-out cert.pem -sha256 -days 365 -extensions v3_req -extensions
v3_ca -subj "/C=US/ST=Ohio/
L=Cleveland/O=NASA/OU=HDTN/CN=localhost" -addext "subjectAltName
= otherName:1.3.6.1.5.5.7.8.11;IA5:ipn:10.0" -config /path/to/openssl.cnf
```

Note: RFC 9174 changed from the earlier -26 draft in that the Subject Alternative Name changed from a URI to an otherName with ID 1.3.6.1.5.5.7.8.11 (id-on-bundleEID).

- Therefore, do NOT use: -addext "subjectAltName = URI:ipn:10.0"
- Instead, use: -addext "subjectAltName = otherName:1.3.6.1.5.5.7.8.11;IA5:ipn:10.0"

To generate the Diffie-Hellman parameters PEM file (which is installed on an induct only), use the following command:

```
openssl dhparam -outform PEM -out dh4096.pem 4096
```