DTNproxy

DTNproxy is a DTN application developed at University of Bologna to allow file transfers from TCP/IP and DTN nodes and vice versa. It is part of the DTNsuite, which consists of several DTN applications making use of the Abstraction Layer, plus the Abstraction Layer itself.

Being built on top of the Abstraction layer, DTNproxy is compatible with all the most important bundle protocol implementations (DTN2, ION, IBR_DTN).

This document aims at providing the user with:

1) release notes
2) building instructions
3) a brief overview and a concise user guide.

Authors: Lorenzo Mustich (lorenzo.mustich@studio.unibo.it), Lorenzo Tullini (lorenzo.tullini@studio.unibo.it), Carlo Caini (Academic supervisor, carlo.caini@unibo.it)

Table of Contents

1 DTNPROXY RELEASE NOTES

1.1 DTNPROXY 1.1.0 (2 AUGUST 2019)

In the 1.1.0 version, there are two major advancements, introduced by Lorenzo Tullini.

First, TCP reception and sending threads have been redesigned to allow parallel reception and sending.

Second, in the DTN to TCP direction is now possible to insert the IP address of final destination as a metadatum, thus removing a limitation of the 1.0.0 version, where the IP address was hard coded in DTNproxy.

The closing procedure (by means of ctrl+c) has been redesigned to remove a known bug.

1.2 DTNPROXY 1.0.0 (22 NOVEMBER 2018)

First release.

1.2.1 Compilation

DTNperf can be compiled with exactly the same modality as the other application of the DTNsuite. The instructions, commons to all DTNsuite applications, are given below.

2 BUILDING INSTRUCTIONS (FOR ALL DTNSUITE APPLICATIONS)

The DTNsuite package consists of a main directory, called “dtnsuite” with one subdirectory for each application, plus one for the Abstraction Layer. For example, the DTNperf code is in “dtnsuite/dtnperf”. As all applications are based on the Abstraction Layer API, before compiling them it is necessary to compile the AL. For the user convenience it is possible to compile the AL and all applications at once (the drawback is that in this case it is more difficult to interpret possible error messages). Both ways are detailed below, starting from the latter, which is preferable for normal users.

Note that the compilation of DTNbox requires the presence of the SQLight package on the host. If not present, you must install it before compilation (“sudo apt-get update, sudo apt-get install libsqlite3-dev”, in Ubuntu). Otherwise, you can comment the DTNbox compilation line in the /dtnsuite/Makefile.

2.1 INTEGRATED COMPILATION

It is possible to compile both AL and all DTNsuite applications in the right sequence with just one simple command. To this end, the Makefile in the “dtnsuite” directory calls the AL Makefile and the DTNsuite application Makefiles in sequential order.

The commands below must be entered from the “dtnsuite” directory. It is necessary to pass absolute paths to DTN2, ION and IBRDTN. The user can compile for one or more ION implementations as shown in the help:

for DTN2 only:

```
make DTN2_DIR=<DTN2_dir_absolute_path>
```

for ION only:

```
make ION_DIR=<ION_dir_absolute_path>
```

for IBR-DTN (>=1.0.1) only:

```
$ make IBRDTN_DIR=<IBRDTN_dir_absolute_path>
```

for all:

```
make DTN2_DIR=<DTN2_dir_absolute_path> ION_DIR=<ion_dir_absolute_path> IBRDTN_DIR=<IBRDTN_dir_absolute_path>
```

The AL libraries and the DTNsuite applications will have either the extension "_vION" or "_vDTN2" or _vIBRDTN, if compiled for one specific implementation, or no extension if for all.

It is also possible to compile for just two BP implementations, by passing only two paths in the command above.

Finally, the user needs to install the program in the system directory (/usr/local/bin) with the commands (with root permissions)

```
make install
ldconfig
```

Example:

```
<path to>/dtnsuite$ make DTN2_DIR=<path to>/sources/DTN2 ION_DIR=<path to>/sources/ion-open-source IBRDTN_DIR=<path to>/sources/ibrdtn
<path to>/dtnsuite$ sudo make install
<path to>/dtnsuite$ sudo make ldconfig
```

For recalling the help:

```
make
```

2.2 SEQUENTIAL COMPILATION

For a better control of the compilation process (e.g. if for whatever reasons it is necessary to change the AL or the specific application Makefiles), it is possible to compile AL and one or more DTNsuite applications sequentially, with independent commands.

2.2.1 Abstraction Layer

The Abstraction Layer (AL) must be compiled first; the AL compilation can be performed by means of the "make" command entered from the AL directory.

The possibilities are the same as those shown for the integrated compilation and will not reported here for brevity. Even in this case the AL will have either the extension "_vION" or "_vDTN2" or _vIBRDTN, if compiled for one specific implementation, or no extension if for all.

By default the AL library are static. If the default is overridden to dynamic (by modifying the AL Makefile), then the user needs to install the library in the system directory with the command (with root permissions):

```
make install
ldconfig
```

Example:

```
<path to>/dtnsuite/al_bp$ make DTN2_DIR=<path to>/sources/DTN2 ION_DIR=<path to>/sources/ion-open-source IBRDTN_DIR=<path to>/sources/ibrdtn
<path to>/dtnsuite/al_bp# sudo make install
<path to>/dtnsuite/al_bp# sudo ldconfig
```

2.2.2 Applications

Once the AL has been compiled (and installed if dynamic), each specific application can be compiled in an analogous way. It is just necessary to add the path to the AL-BP (in bold in the example below, for DTN2 only):

```
make DTN2_DIR=<DTN2_dir> AL_BP_DIR=<al_bp_dir>
```

Note that it is necessary to have previously compiled AL_BP for exactly the same ION implementations, as the library with the corresponding extensions must be already present in AL_BP_DIR (vION" or "_vDTN2" or vIBRDTN, or no extension at all if the support is for all implementations).

Finally, the user needs to install the program in the system directory (/usr/local/bin) by entering the same commands used for both the integrated compilation and the AL.

```
make install
make ldconfig
```

Example (for DTNperf)e:

```
<path to>/dtnsuite/dtnperf$ make DTN2_DIR=<path to>/sources/DTN2 ION_DIR=<path
to>/sources/ion-open-source AL_BP_DIR=<path to>/UniboDTN/al_bp
<path to>/dtnsuite/dtnperf$ sudo make install
<path to>/dtnsuite/dtnperf$ sudo ldconfig
```

The "dtnperf" executable will have either the extension "_vION" or "_vDTN2" or _vIBRDTN, if compiled for one specific implementation; no extension if compiled for all, as always.

3 DTNPROXY OVERVIEW AND GUIDE OF USE.

DTNproxy is a simple file transfer proxy designed to connect TCP/IP to DTN nodes and vice versa. A peculiar characteristic, with respect to basic file transfer programs provided with BP implementations, is that it preserves the original name of the file. DTNproxy communicate with DTNperf client and server on the DTN side, and with two simple ancillary TCP applications ("tcpclient" and "tcpserver").

Although DTNproxy primary use is to connect a TCP/IP node with a DTN one, and vice versa, a couple of DTNproxies could also be used at the borders of a nested DTN network when both the file source and destination are TCP nodes. This possibility is however left to future versions, as it requires the sending of the final TCP address, which has not been implemented yet.

3.1 OVERVIEW

DTNproxy is a sort of "translator", consisting of four main threads, working as two independent couples: from TCP/IP to DTN, we have "tcpReceiver" followed by "dtnSender and from DTN to TCP/IP, "dtnReceiver followed by tcpSender. From version 1.1, TCP reception and sending threads have been redesigned so that multiple files can be now received or sent in parallel. To this end, any sending and receiving request is processed by a parallel subthread of tcpSender or tcpReceiver. The receiving and the sending thread of each couple follow the classic "producer-consumer" model. The receiving processes, the "producer", saves the incoming file in a circular buffer; the sending process, "the consumer" takes it from the circular file, when ready. Push primitive is blocking if circular buffer is full, and pop primitive is blocking if circular buffer is empty, thus when the buffer is full the receiver must stop waiting for the sender and vice versa, when the buffer is empty the sender must stop, waiting for the receiver. The size of circular buffer is hard coded in the file definition.h.

The actual data exchanged are shown below.

3.1.1 from TCP/IP to DTN

As shown in the figure below, the TCP client (e.g. tcpclient) must send to the DTNproxy in one TCP connection:

1. the DTN destination EID
2. the filename
3. the file content

4. only when a couple of DTNproxies is used between a TCP source and destination, the address of the final TCP destination (to be implemented yet).

DTNproxy in turns will send to the DTN server perf server (dtnperf - -server), in one bundle:

1. the filename (in the DTNperf client header used with option "-F" for file transfer)
2. the file content (in the DTNperf payload)
3. optional: the address of the TCP source (in the bundle metadata block, to be implemented yet; it could be useful when a response must be sent back)
4. only when a couple of DTNproxies is used between a TCP source and destination: the address of the final TCP destination (in the bundle metadata block, to be implemented yet)

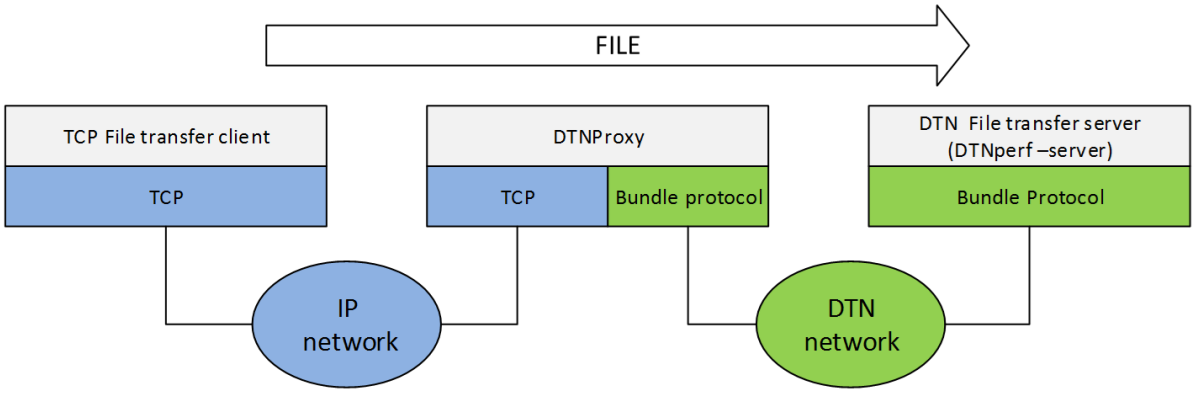

Figure 1: DTNproxy scheme: from a TCP/IP client to a DTN server.

3.1.2 from DTN to TCP/IP

In the opposite direction, see the figure below, the DTN client (e.g. dtnperf –client), used with the option "-F" (file transfer mode), must send to DTNproxy, in one bundle:

1. the address of the TCP destination in a bundle metadata block
2. the filename (in the DTNperf client header)
3. the file content (in the DTNperf payload)

DTNproxy in turns must send to the TCP server, in one TCP connection:

1. the filename
2. the file Content
3. optional: the EID of the DTN file transfer client (in the bundle metadata block; to be implemented yet; it could be useful when a response must be sent back)

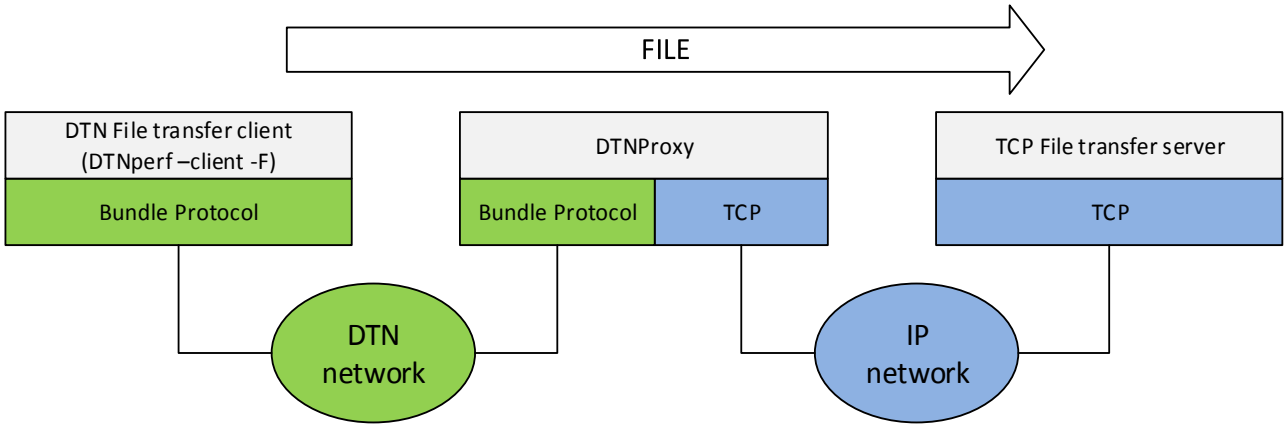

Figure 2: DTNproxy scheme: from a DTN client to a TCP/IP server.

3.2 SYNTAX

```
SYNTAX: dtnproxy [options]
```

The syntax of DTNproxy is very simple, with no compulsory parameters and only few options, all without arguments:

- "- - help"
- "- - no-header" (to omit the insertion of the DTNperf header)
- "- - debug" (to have log messages on screen)

Only the second deserves an explanation. When used, the"dtnSender" thread inserts the file content in the bundle payload but omits the DTNperf header. This allows the use of a DTN file server different from DTNperf (e.g. bpreceivfile in ION). This way, however, the name of the file is not transmitted and thus cannot be preserved at destination. In future versions the filename could be sent as a metadatum.

3.3 USE OF DTNPROXY AND ANCILLARY PROGRAMS

3.3.1 TCPclient and TCPsender compilation

The ancillary programs TCPclient and TCPsender, in the "auxiliary_TCPprograms" directory must be compiled a part. To compile TCPclient (and analogously for TCPserver) enter from the "tcpclient" directrory

```
gcc main-c -o tcpclient
```

Then to install

```
sudo cp tcpclient /usr/local/bin/
```

You can also use compile.sh, also in the "tcpclient" directory.

3.3.2 Ports and DTN demux tokens

The TCP port of TCPserver is 3000.

The TCP port and DTN demux tokens of DTNproxy are:

- Listening port (used by tcpReceive): 2500
- Rx bundle demux token: either 5000 (ipn scheme) or "proxy_server" (dtn scheme)
- Tx bundle demux token: either 5001 (ipn scheme) or "proxy_client" (dtn scheme)

3.3.3 To transfer a file from TCPclient to DTNperf server

Let us assume the TCP client on vm1, the DTNperf server on vm4, (EID=ipn:4.2000) and DTNproxy on vm2 (listening on IP 10.0.1.2:2500).

on vm1:

```
tcpclient namefile ipn:4.2000 10.0.1.2 2500
```

on vm2:

```
dtnproxy --debug
```

on vm4:

```
dtnperf --server
```

3.3.4 To transfer a file form DTNperf to TCPserver

Let us assume the DTNperf client on vm1, TCPserver on vm4 (10.0.1.4: 2500), and DTNproxy on vm2 (EID=ipn:2.5000):

```
dtnperf_vION --client –d ipn:2.5000 –F nomefile –R1b -P 100k --mb-type222 --
                          mb-string 10.0.1.4:2500
```

on vm2:

```
dtnproxy --debug
```

on vm4:

```
tcpserver
```

Warning: the bundle payload (-P option) must be larger than the file dimension plus the DTNperf header, not to split the file in multiple bundles.