DTNperf_3

DTNperf_3 is the third major version of the DTNperf tool, developed at University of Bologna.
As suggested by its name, it is a tool for performance evaluation in Bundle Protocol DTN environments.
This document aims at providing the user with:

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

Authors (v3.5): Carlo Caini (Academic supervisor, carlo.caini@unibo.it), Michele Rodolfi, Anna D'Amico, Davide Pallotti.

Table of Contents

RELEASE NOTES

DTNperf 3.5.1 (2 May 2017)

Update to allow the compilation of AL with ION 3.6.0

- The name of one structure in the Abstraction Layer (AL) has been updated according to the new name used in ION 3.6.0, to make it possible its compilation with ION 3.6.0 code. AL version number has been increased to 1.5.1. For this reason, this new version (DTNperf 3.5.1) cannot be compiled with previous version of ION.

DTNperf 3.5.0 (21 July 2016)

New features

- DTNperf now supports also IBR-DTN.
- The Abstraction Layer (AL) API now supports also IBR-DTN, as well as DTN2 and ION. AL version number has been increased to 1.5.0.

DTNperf 3.4.0 (23 November 2015)

New features

Monitor. New option: “--rt-print[=filename]”, to print in real time human readable status report information

- This option prints a summary of each SR received on the screen (or on a file) in a format that is immediately comprehensible to humans. It complements the .csv file(s) that is (are) vice versa devoted to “a posteriori” processing by a spreadsheet. In simple experiments, involving few bundles, this new option can provide enough information to distinguish between successes and failures, without recurring to a more time consuming a posteriori examination of the .csv logs.

Bug fixed

- Abstraction Layer API. A bug preventing a correct data exchange between nodes on 32 and 64-bit architectures has been fixed.

DTNperf 3.3.5 (14 April 2015)

New features

Alternate URI scheme registration always possible via the “--force-eid <[DTN|IPN]” option

- Server and Monitor: forced registration on the alternate URI scheme now possible also on DTN2 (as well as in ION as before), via “--force-eid DT|IPN” option.
- Client: in ION the previous automatic selection of the registration URI scheme based on the scheme of destination has been disabled. Now the client registers itself as both the server and monitor do, i.e. by default as dtn in DTN2 and as ipn in ION; in both cases the alternate scheme can be selected through the --force-eid option.

All modes. New option:“--ipn-local <num>” (DTN2 only).

- When forcing the ipn scheme on a DTN2 node (--force-eid IPN) it is always necessary to let DTNperf_3 know the ipn number of the node with this option.

Client. New option: “--no-bundle-stop”

- To prevent the client from sending the bundle stop (and force-stop) to the monitor at the end of the session.

Monitor. New option: “--oneCSVonly”

- To generate a unique CSV (Comma Separated Values) file, instead of as many files as concurrent client instances (sessions).

Server: new name for DTNperf ACKs”

- DTNperf ACKs, sent by the server to the client to acknowledge reception of a data bundle are now saved in /tmp as “dtnperfack_#”, where # denotes a progressive integer. This to make always shorter than 32B the DTNperf ACK filename, as requested in ION. These ACKs are automatically cancelled.

New name for bundle payloads

- Bundle payloads in ION are now saved in /tmp as “ion_PID_#”, where PID is the Process IDentifier and # is a progressive number DTNperf ACKs, sent by the server to the client to acknowledge reception of a data bundle are now saved in /tmp as “dtnperfack_#”, where # denotes a progressive integer. This to make always shorter than 32B the DTNperf ACK filename, as requested in ION.

Bundle payloads automatically cancelled (ION only)

- Bundles payloads are now automatically deleted (at present only in ION) to limit storage consumption.

Extended scope of the “-e” monitor option

- The monitor forces the closure of a .csv file when the time elapsed from the last reception of a Status Report associated to it becomes longer than the “session threshold”. Analogously, when the time elapsed between the reception of the Bundle Stop and the reception of all delivered status report becomes longer than the “closing threshold”. In DTN2 the default value for both is the bundle lifetime of the client session associated with the .csv file, while in ION we had two different fixed values. Now these two have been unified, and the default (120s) can be overridden by means of the –e option.

Sub-sub-version number added (e.g. 3.3.5)

Bug fixed

Client: -W (window-based congestion control) caused client crashes (in versions included in the ION package). Now fixed.

- In previous versions the –W (window based congestion control) option did not work correctly. Now the problem has been fixed and users can take advantage of both –W and –R congestion control modes.

Joint use of the -F and –crc client options

- A bug prevented file reconstruction on the server if the client jointly useed the –F (file Tx mode) and –crc options. Now the bug has been fixed and crc check can be enabled without interfering with the –F option. By the way, note that if the bundle that contains the whole file, or one (or more) of the bundles that contain a segment of a segmented file does not pass the crc check, the bundle is discarded and the file is not saved by the server, as it could not be correctly reconstructed. This in accordance to the fact that the client does not perform bundle retransmissions.

Monitor: In peculiar cases, the .csv file closure was anticipated. Now fixed.

AL API. One API of the AL has been changed.

- This requires the installation of the latest AL version before installing DTNperf 3.3.x

DTNperf 3.0.0 (DTNperf_3)

Authors: Michele Rodolfi, Anna d’Amico, Carlo Caini (project supervisor).

Although the aim is the same as previous versions, the project has been deeply redesigned and the code totally rewritten.

Three modes: client, server and monitor (new). They correspond to source, destination and “reply-to” dtn nodes. The monitor, in charge of collecting status reports in a .csv file, can be “internal” (on the source), or “external” (on a different node).

Server and external monitors can manage multiple instances of the client.

Three Tx modes: time, data and file. The first two are the same as before. The file mode is now much more robust as files segmented in multiple bundles can now be reassembled on the server even if received in disorder (but without losses).

Two congestion control methods: window based and rate based (new).

In the window based congestion control, bundle sent are now acknowledged by bundle ACKs sent by the server and no more by delivered status reports. In the rate based congestion control bundles are not acknowledged.

Support of both DTN2 and ION. The DTNperf application runs on top of a new “Abstraction Layer” (AL), which has the aim of decoupling the DTNperf application from the underlying BP implementation. The AL can be compiled for either DTN2, ION or both. DTNperf calls the API of the AL (or “grey” APIs), which in turns calls the API of either DTN2 or ION. If compiled for both, the choice of DTN2/ION APIs is made at run time, accordingly to the implementation that is actually running.

See the other sections (DTNperf compilation instructions, DTNperf general description and example of use) for further information.

DTNperf 2.x (DTNperf_2)

Authors: Piero Cornice, Marco Livini, Carlo Caini (project supervisor)

Window based congestion control added. With the –W parameter it is now possible to set the maximum amount of bundles in flight (sent but not acknowledged; “ACKs” are delivered status report). This is necessary to fill the pipe and to obtain accurate goodput evaluation.
Status reports of all machines are collected by the dtnperf client (the bundle source) in a .csv file, for later analysis on a spreadsheet.
Possibility to send bundles with dummy payload for either an interval time (time mode) or a specified amount of data (data mode).
Possibility to send a file, with file segmentation into multiple bundles of desired dimension (file mode).
The same version developed for GNU/Linux is ported with minimal changes on MAEMO OS for Nokia N900 smartphones (by Francesco Baldassarri). It is fully interoperable with the 2.x version for Linux machines.
Bugs removed.

DTNperf 1.x

Author: Piero Cornice
First version of DTNperf, intended to be a DTN equivalent of the Iperf tool for DTN network (DTN2 implementation).
DTNperf client sends bundles of wanted dimension and dummy payload to dtnperf server, for a specified time interval.
Only one bundle in flight allowed (“in flight” means sent but not acknowledged yet; “ACKs”are delivered status report).

Rationale of enhancements and amendments from v3.0

DTNperf_3 aims, design and use have been all described in details in the following paper, to which the user is referred as DTNperf_3 documentation. Some example of use are also reported at the end of this document.

- C. Caini, A. d’Amico and M. Rodolfi, “DTNperf_3: a Further Enhanced Tool for Delay-/Disruption- Tolerant Networking Performance Evaluation”, in Proc. of IEEE Globecom 2013, Atlanta, USA, December 2013, pp. 3009 - 3015. Digital Object Identifier: 10.1109/GLOCOM.2013.6831533

The notes below are intended to complement this document, as they describe the rationale of the most important enhancements introduced after its publication.

IBR-DTN support
From DTNperf v3.5 and AL v1.5 the support of BP implementation has been extended to IBR-DTN.

Alternate URI scheme registration always possible via the “--force-eid <[DTN|IPN] option
If DTNperf_3 is compiled for multiple BP implementation (at least two among DTN2, ION and IBR-DTN), the choice of DTN2/ION/IBR-DTN APIs is made at run time, accordingly to the implementation that is actually running, i.e. “dtn” in DTN2 and IBR-DTN, or “ipn” in ION. It is always possible to force the alternate scheme registration with the –force-eid [DTN|IPN] option.

New option:“--ipn-local <num>” (DTN2 only).
When forcing an ipn registration on DTN2 it is compulsory to tell dtnperf the ipn node number of the registration, as in DTN2 configuration file, e.g. /etc/dtn.conf, there is not any notions of this ipn alias. This must be done with the new option “–ipn-local #” associated to “–force-eid IPN”. Vice versa, when forcing a dtn registration in ION (--force-eid DTN) there is no need of indicating the dtn name as the host name of the machine is automatically used to derive the dtn name (e.g host “vm1” becomes “dtn://vm1.dtn”.

Client. New option: “--no-bundle-stop”.
By default, the client sends a bundle STOP to the “reply to:” address, to inform the monitor that the bundle transmission has ended and that n bundles were sent. The monitor uses this information to wait for n delivered status reports before closing the .csv file where all status reports concerning the same client instance are collected (if a bundle or a delivered SR is lost, a timeout force the closure of the .csv file). In some circumstances (e.g. routing studies) it may be preferable not to have the bundle stop sent. The new client option “–no-bundle-stop” has this aim. The user can either force the closure of the .csv file by pressing ctrl+c on the monitor or wait for the timeout expiration.

Monitor. New option: “--oneCSVonly”
As it is perfectly possible to have a monitor always on, working for multiple instances of the client running either on the same or in other nodes, the monitor by default collects status reports referring to bundle generated by different client instances on different files, as said above. In some circumstances, however, it may be preferable to have all status

reports collected in the same file. To this end the monitor option “--oneCSVonly” has been introduced. The .csv file is saved only when either the user press ctrl+c or a timeout expires (in case no new status reports are received form a while). This feature may be useful when dealing with multiple priorities (e.g. in routing studies), as one different instance of the client must be started for each priority; in this case, all bundles logically belong to the same experiment, although generated by concurrent instances, thus the corresponding status reports should be preferably collected in just one file.

Monitor. New option: “--rt-print[=filename]”,
This option prints a summary of each SR received on the screen (or on a file) in a format that is immediately comprehensible to humans. It complements the .csv file(s) that is (are) vice versa devoted to “a posteriori” processing by a spreadsheet. In simple experiments, involving few bundles, this new option can provide enough information to distinguish between successes and failures, without recurring to a more time consuming a posteriori examination of the .csv logs.

BUILDING INSTRUCTIONS

The DTNperf package consists of a main directory, called either “UniboDTN” or “dtnperf” (the latter in ION), with two subdirectories, “dtnperf” and “al_bp”. As DTNperf is based on the Abstraction Layer API, before compiling DTNperf is necessary to compile the AL. For the user convenience it is also possible to perform both tasks 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, as it is easier and adequate in most cases. Instructions below refer to DTNperf v3.5.0 and AL v1.5. ION users should replace UniboDTN with dtnperf in the instructions below.

Integrated compilation

It is possible to compile both AL and DTNperf in the right sequence with just one simple command. To this end, in the UniboDTN directory there is a Makefile that calls both the AL Makefile and the DTNperf application Makefile in sequential order.
The commands below must be entered from the UniboDTN directory containing the “father” Makefile, which will call, in a transparent way, the Makefiles contained in al_bp and dtnperf subdirectories. It is necessary to pass absolute paths to DTN2, ION and IBRDTN.

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>
```

It is also possible to compile for just 2 BP implementations, by passing only 2 of them in the command above.
Finally, the user needs to install the program in the system directory with the commands (with root permissions)

```
sudo make install
```

```
sudo ldconfig
```

Example:

```
<path to>/UniboDTN$ make DTN2_DIR=<path to>/sources/DTN2 ION_DIR=<path
to>/sources/ion-open-source IBRDTN_DIR=<path to>/sources/ibrdtn
```

```
<path to>/dtnperf# make install
```

For an exhaustive help:

```
make
```

Sequential compilation

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

Abstraction Layer

Before compiling DTNperf, it is necessary first to compile the Abstraction Layer (AL); the AL compilation can be performed by means of the the “make” command. There are many possibilities, depending on the BP implementation(s) you want to support. The commands below must be entered from the AL directory
(e.g. <path to>/UniboDTN/al_bp)

for DTN2 (>=2.9) only:

```
$ make DTN2_DIR=<DTN2_dir>
```

for ION only (>=3.3; also from 3.1.3 to 3.2.2 by flipping the “NEW_ZCO” compiler directive in the Makefile):

```
make ION_DIR=<ION_dir>
```

for IBR-DTN (>=1.0.1) only:

```
$ make IBRDTN_DIR=<IBRDTN_dir>
```

for all:

```
make DTN2_DIR=<DTN2_dir>  ION_DIR=<ION_dir>  IBRDTN_DIR=<IBRDTN_dir>
```

Note that it is also possible to compile for just 2 BP implementations, by passing only 2 parameters in the command above.
Then the user needs to install the library in the system directory with the command (with root permissions):

```
make install
```

The AL will have either the extension “_vION” or “_vDTN2” or _vIBRDTN, if compiled for one specific implementation, or no extension if for all.

Example:

```
<path to>/UniboDTN/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>/UniboDTN/al_bp# make install
```

DTNperf application

Once the AL has been compiled and installed, DTNperf can be compiled in an analogous way. The commands below must be entered from the directory containing the DTNperf application files (e.g. <path to>/dtnperf/dtnperf)

for DTN2 only:

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

for ION only:

```
make ION_DIR=<ION_dir> AL_BP_DIR=<al_bp_dir>
```

for IBR-DTN (>=1.0.1) only:

```
$ make IBRDTN_DIR=<IBRDTN_dir> AL_BP_DIR=<al_bp_dir>
```

for all:

```
make DTN2_DIR=<DTN2_dir> ION_DIR=<ion_dir> IBRDTN_DIR=<IBRDTN_dir>
AL_BP_DIR=<al_bp_dir>
```

Note that It is also possible to compile for just 2 BP implementations, by passing only 2 of them in the command above.
Finally, the user needs to install the program in the system directory with the command (with root permissions)

```
make install
```

Example:

```
<path to>/UniboDTN/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>/UniboDTN/dtnperf# make install
```

The “dtnperf” executable will have either the extension “_vION” or “_vDTN2” or vIBRDTN, if compiled for one specific implementation, or no extension if compiled for all.

DTNPERF_3 OVERVIEW AND USE.

DTNperf_3 aims, design and use have been all described in details in the following paper, to which the user is referred as DTNperf_3 primary source of documentation.

- C. Caini, A. d'Amico and M. Rodolfi, "DTNperf_3: a Further Enhanced Tool for Delay-/Disruption- Tolerant Networking Performance Evaluation", in Proc. of IEEE Globecom 2013, Atlanta, USA, December 2013, pp. 3009 - 3015. Digital Object Identifier: 10.1109/GLOCOM.2013.6831533

The following notes aims at providing the user with a brief overview about DTNperf_3 and with a concise guide of possible uses, with many syntax examples.

DTNperf_3 Overview

DTNperf_3 has three operating modes: client, server and monitor. The client generates and sends bundles, the server receives it and the monitor collects status reports in .csv files. Client, server and monitor correspond to the BP addresses "from", "to" and "reply to".

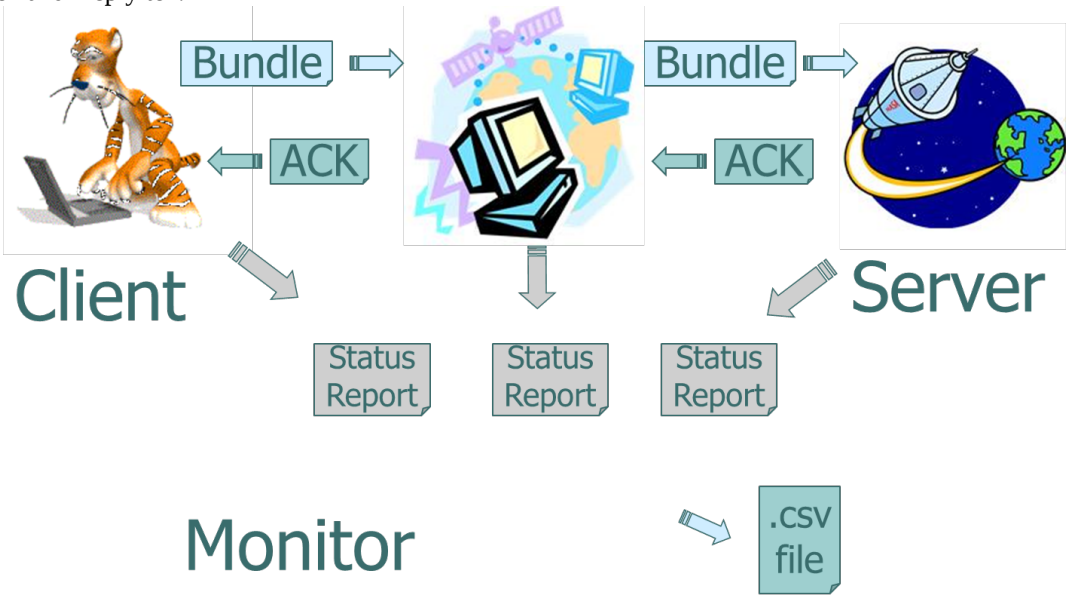

Figure 1: DTNperf operating modes: client, server and monitor.

DTNperf Client

```
SYNTAX: dtnperf --client -d <dest_eid> <[-T <s> | -D <num> | -F <filename>]>
[-W <size> | -R <rate>] [options]
```

The most important parameters are explained below. The full list of options can be obtained with the command "dtnperf --client --help".

Tx modes

The client has three mutually exclusive Tx modes to send bundles to the server. In the time-mode (-T), a series of bundles with a dummy payload of desired dimension (-P option) is generated and "sent", i.e. passed to the BP daemon for transmission, until the pre-set transmission time elapses. Data-mode (-D) is the same as time-mode, except that the bundle generation process ends after a given amount of data has been "sent" to BP, and not after a time interval. File-mode (-F) differs from data-mode because a file is transferred, instead of dummy data. A noteworthy feature is the possibility to split the file into multiple bundles of desired dimension.

Congestion control (window- or rate- based)

Independently of the Tx mode, there are two alternative congestion control policies available: window-based (-W) and rate-based (-R). In the former, the "congestion window" W represents the maximum number of bundles in-flight (i.e. sent but not acknowledged). The mechanism is similar to the TCP congestion window, but: 1) W has a fixed dimension; 2) in-flight bundles can be non-consecutive (to cope with non-ordered delivery of BP); 3) there are not retransmissions (acknowledgments are used only to trigger the transmission of new bundles). DTNperf_3 has enhanced this function

with respect to previous versions, by replacing “delivered” status reports generated by the BP of the destination node, by ACK bundles specifically created by the DTNperf_3 server, as acknowledgments of bundles sent. This innovation is essential to decouple the “reply to” from the source node, thus allowing the monitor to be launched on a node different from the source.
Although the window-based mechanism is generally useful, in some cases, e.g. to emulate streaming traffic, it would be preferable to generate traffic at constant rate. For this reason, in DTNperf_3 a rate-based congestion control has been added. In response to rate-based traffic the server does not generate any ACKs, like UDP.

DTNperf Server

The server receives bundles, acknowledges them if sent in window-based mode, and reassembles bundle payloads if a file is transferred.

```
SYNTAX: dtnperf --server [options]
```

In this third version, one instance can serve multiple clients in parallel. Moreover, the file transfer is now robust against disordered bundle delivery (incoming payloads are buffered until either the entire file is received or a timeout expires).

DTNperf Monitor

The monitor receives and collects status reports and some DTNperf control bundles. It can be launched either as an independent process on an external node (external monitor), in which case it can serve many concurrent clients, or as a “child” process of a client (dedicated monitor), in which case it serves only its “parent” client. The syntax to launch an external monitor is the following:

```
SYNTAX: dtnperf --monitor [options]
```

The monitor creates a different .csv log file for each DTNperf client session (i.e. one client “launch”). This file contains all status reports generated by all nodes during the session and it can be directly imported into a spreadsheet for further analysis.

The Abstraction Layer

To facilitate interoperability tests, DTNperf_3 has been designed to be independent of the BP implementation. It relies on the “Abstraction Layer”, i.e. a library expressly designed to provide a common interface for DTNperf towards the APIs of different BP implementations.

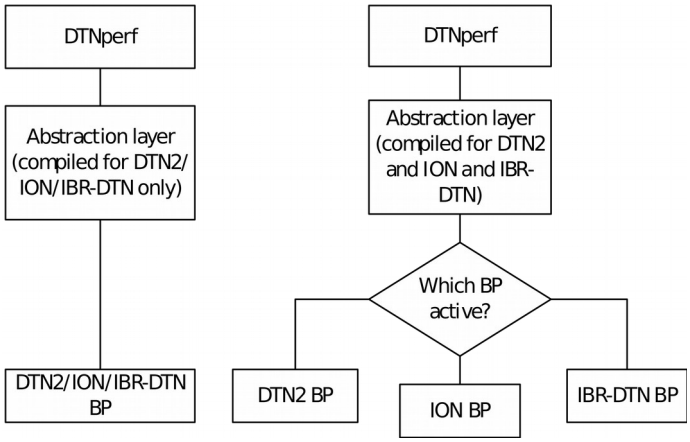

Figure 2: DTNperf compatibility. a) DTN2/ION/IBR-DTN only, b) all of them with API call selection at run time.

The present version of the AL supports DTN2, ION and IBR-DTN, but it could be further extended to other implementations. If the AL is compiled either for DTN2 or ION, the compiled version of DTNperf, with postfix _vION, vDTN2, or vIBR-DTN, can run only on top of DTN2 or ION or IBR-DTN (Figure 2a). If the AL is compiled for all, DTNperf (no postfix) can run on top of all, as the switch between DTN2, ION and IBR-DTN API calls is performed at run time (Figure 2b). For further information on the AL see the companion guide.

DTNperf_3 Use

Although derived from authors’ experience on space communications, DTNperf_3 has a general scope aiming at embracing most DTN applications. In the examples below, the use of DTNperf_3 in some typical DTN applications is presented, assuming that we have three DTN nodes, vm1, vm2 and vm3 when the dtn scheme is adopted, or nodes 1, 2

and 3 when the ipn scheme is preferred, as source, destination and external monitor. We will focus on client syntax, as the server and the monitor can always be launched with the following commands:

```
dtnperf - -server - -debug=1
dtnperf - -monitor - -debug=1
```

Basic applications

Ping

To ping vm2 from vm1 (if the server is registered with the dtn scheme)

```
dtnperf - -client -d dtn://vm2.dtn -T 15 –W 1 - -debug=1
```

or (if the server is registered with the ipn scheme; note that the demux token of the server is always 2000)

```
dtnperf - -client -d ipn: 2.2000 -T 15 –W 1 - -debug=1
```

With this command vm1 will send bundles of 50 kB (default) to vm2 for 30s (-T15), one by one (-W1 allows just one bundle in flight). The short default lifetime (60s) is useful to force the deletion of undelivered bundles, which could interfere on subsequent experiments. As no external monitor is indicated, the .csv log file will be created on vm1 by a dedicated monitor.

Trace

To trace the route of a bundle:

```
dtnperf - -client –d dtn://vm2.dtn –D100kB -P100kB –W1 –C - f –r
```

This command sends a single bundle of 100 kB as the total amount of data (-D100kB) coincides with the bundle payload (-P100kB). The custody option is on (-C) and some optional status reports are requested (forwarded, -f, received, -r). From the .csv file it is straightforward to trace the route of the bundle sent.

File transfer.

To transfer a file segmented into multiple bundles of desired dimension:

```
dtnperf - -client –d dtn://vm2.dtn –F picture.jpg –P 100kB –W4
```

Here a file is sent (-F) instead of dummy data. The dimension of the bundle payload (-P100kB) is the dimension of segments into which the file will be split. This feature is useful to match limited contact volumes as an alternative to proactive fragmentation. Note however that file segmentation is carried out at application layer (by DTNperf), while bundle fragmentation is performed at BP layer (by the BP daemon).

Performance evaluation in continuous and disrupted networks

Goodput (macro-analysis)

Goodput (data_ACKed/time_elapsed) evaluation makes sense especially if the DTN network is not partitioned (e.g. in GEO satellite communications, where the challenges are long delay and losses). To this end, it is necessary to send dummy bundles with the window-based congestion control for a reasonable amount of time to reach and maintain a steady state. The following command could be suitable in the case of a hypothetical GEO satellite hop:

```
dtnperf - -client -d dtn://vm2.dtn -T 30 –P 1MB –W 4 –l 60
```

With this command vm1 will send bundles to vm2 for 30 s (-T30), i.e. for a much longer interval than the typical GEO RTT (600ms including processing time). To fill the (likely) large bandwidth-delay product and to reduce the impact of bundle overhead, bundles are large (-P1MB) and the congestion window W is greater than one (-W4). The same experiment should be repeated increasing W until the goodput reaches a maximum. Note that goodput evaluation should always be complemented by the analysis of status reports, collected in the example by the internal monitor (default), to control the regularity of the bundle flow and to recognize the reasons of the macro-results achieved.

Status report analysis (micro-analysis).

The evaluation of goodput is useful when links are continuous or only moderately disrupted (e.g. in GEO satellites with mobile terminals). As the chances of disruption increase (e.g. LEO satellites, deep space communications), the study of individual bundles (i.e. micro-analysis) becomes more important than goodput. A possible command in the presence of disruption is:

```
dtnperf - -client -d dtn://vm2.dtn –m dtn://vm3.dtn –D30MB –P 1 MB –W4 –l
200 (server and monitor registered with dtn scheme)
```

```
dtnperf - -client -d ipn:2.2000 –m ipn:3.1000 –D30MB –P 1 MB –W4 –l 200
(server and monitor registered with the ipn scheme; the demux tokens are
always 2000 and 1000 respectively)
```

This command dispatches 30 bundles of 1 MB each, with a limit of 4 bundles in flight. Status reports are collected (possibly in real time by means of dedicated links) by an external monitor (-m option). Note that the lifetime has been increased to 200s (-l200) to cope with disruption. Moreover, Data mode (-D30) is preferable because disruption makes uncertain the actual duration of bundle transfer.

Status report analysis of streaming traffic.

To emulate a streaming source a possible command is:

```
dtnperf - -client -d dtn://vm2.dtn -T30 –P100kB –R2b
```

This command generates a stream of 100 kB bundles for 30s, at 2 bundles per second (-R2b). No ACKs are generated by the server, as the congestion control is rate-based.

Performance evaluation in partitioned networks: “data mule” communications

One of the most evident advantages of DTN architecture is the possibility to cope with network partitioning. The extreme case is that of “data mule” communications, where an intermediate node (the “mule”, or “ferry”) is alternatively connected, thanks to its movement, either to the sender or to the destination. In this case, the evaluation of goodput is useless, while the microanalysis is essential. A possible command is:

```
dtnperf - -client -d dtn://vm2.dtn –m dtn://vm3.dtn –D 10 MB –P1 MB –W10 –l
200 -- debug=1
```

The external monitor (-m option) is very useful here, especially if connected to other nodes through dedicated links, which can be easily carried out in a lab testbed, since links used to transfer data are only occasionally active. Note that by setting the window to the total number of bundles (10 in the example) these are sent in one burst, which can be preferable in this kind of experiments, in order not to have to wait for ACKs (once the burst of data is sent, the user can interrupt the client by pressing ctrl+c). Alternatively, the user can take advantage of the rate-based congestion control, which does not imply any ACKs (e.g. by setting -R10b instead of –W10).

Interoperability tests

Thanks to the AL library, which now (v1.5) includes IBR-DTN support, DTNperf_3 (v3.5) can run on top of either DTN2, ION or IBR-DTN BP. If compiled for multiple implementations, the very same executable can be used, as the choice between the different implementation API is made at run time. This feature makes DTNperf_3 particularly suitable for carrying out interoperability tests among different implementations. To this end, a key requirement is the capability of coping with both the different EID URI schemes, “dtn” and “ipn”. For example, DTN2 and IBR-DTN prefer the “dtn” scheme, ION the “ipn” one. To facilitate experiments, not only can DTNperf_3 register itself with the preferential scheme, but also with the alternate URI scheme (dtn in ION, ipn in DTN2 and IBR-DTN), thus allowing full interoperability (in particular also with DTN2 “not NASA patched” nodes unable to use the ipn scheme correctly). In brief, DTNperf_3 can cope with every combination of DTN2, ION and IBR-DTN nodes, independently of the EID URI scheme adopted by these. For example, the following command can be launched on a machine running either DTN2 or ION or IBR-DTN (server registered with the “dtn2 scheme, monitor with the “ipn” scheme):

```
dtnperf - -client -d dtn://vm2.dtn –m ipn:3.1000 –D30MB –P 1 MB –W4 –l 200
```

Of course, a prerequisite for DTNperf_3 interoperability is that all DTN2, ION and IBR-DTN configuration files are correctly configured for supporting interoperability at bundle layer, which is not trivial. Moreover, it is worth noting that the “bleeding edge” version of DTN2, i.e. the development version, requires a few patches developed by NASA to improve the basic and buggy support of the ipn scheme provided by the bleeding edge. Without these patches, there is not fully interoperability at BP layer between DTN2 and ION.
To help the user interested in interoperability, a few advanced examples involving the use of the alternate scheme in the registration phase, are reported below.

Dtnperf running on top of DTN2: registration as “ipn” with node number 4.

In DTN2 it is necessary to use 2 options: “force-eid” to override the dtn default and “ipn-local” to assign the node number.
The client will register as ipn:4.x, with x the ipn demux token number automatically assigned by DTNperf. The registration is always independent of the destination scheme (dtn in the former example, ipn in the latter).

```
dtnperf --client dtn://rita.dtn --D200k --R1b --force-eid IPN --ipn-local 4
--debug=2.
```

```
dtnperf --client ipn:5.2000 --D200k --R1b --force-eid IPN --ipn-local 4
--debug=2.
```

The server will register itself as ipn:4.2000 with the following command:

```
dtnperf --server --force-eid IPN --ipn-local 4 --debug=2
```

The monitor will register itself as ipn:4.100 with the following command

```
dtnperf --monitor --force-eid IPN --ipn-local 4 --debug=2
```

Dtnperf running on top of ION: registration as "dtn".

In ION it is necessary to use only the "force-eid" option to override the ipn default. The dtn node name is derived automatically by ION starting from the node hostname (no possibility of overriding this). Thus the DTNperf client on "susy" (hostname) will register as dtn://susy.dtn/x, with x the dtn demux token. The registration is always independent of the destination scheme (ipn in the example below):

```
dtnperf --client ipn:5.2000 --D200k --R1b --force-eid DTN --debug=2 .
```

Dtnperf running on top of IBR-DTN, registration as "ipn".

In IBR-DTN it is necessary to use only the "force-eid" option to override the dtn default. In fact, by contrast to DTN2, in IBR-DTN the ipn EID, included the node number is set in the configuration file. The client will register as ipn:4.x, with x the ipn demux token number). The registration is always independent of the destination scheme (dtn in the example below):

```
dtnperf --client dtn://rita.dtn --D200k --R1b --force-eid DTN --debug=2 .
```

Disjoint use of client or monitor.

DTNperf modes (client, server, monitor) were conceived to work together. However, it is perfectly possible, and sometimes also convenient, to use the client (rate based) or the monitor in a fully autonomous way.

Client

For example, as the client in rate based congestion control does not need any feedback form the server, it can be used autonomously as a powerful and flexible source of bundles. The user can specify the bundle dimension, the total amount of data, the Tx rate (either in bundle/s or bit/s, at his choice), and all bundle flags (delivered is always on). No more need of custom complex scripts.

Monitor

The monitor can be used in an independent way as well, to collect all status reports. It is enough to set its EID as report to in the bundle source, which can be whatsoever, i.e. different from a DTNperf client. In case of multiple sources, the option "--oneCSVonly" can be useful to generate a unique CSV (Comma Separated Values) file, instead of as many files as concurrent sources. It can also be used as a real-time monitor with the option "--rt-print[=filename]". Status reports actually provides the user with a complete "telemetry" at bundle layer. In most cases DTNperf monitor allows the user to get rid of time consuming analysis of log files. The .csv file(s) generated by the monitor can be directly imported into a spreadsheet for deeper (and much faster) analysis.