Networking for Communications Challenged Communities:
Architecture, Test Beds and Innovative Alliances
Contract no: 223994

N4C
Generic DTN Documentation

Trinity College Dublin/Intel
Stephen Farrell
www.tcd.ie
stephen.farrell@cd.tcd.ie

ABSTRACT (Max 400 word)

This document describes how to set up a DTN node from TCD's N4C code repository and how to integrate applications with the DTN infrastructure. It is intended for systems administrators, developers and advanced end-users. Feedback on this version of the document is welcome and should be sent to the authors.

Document history		
Status	**Date**	**Authors**
FINAL	June 10th 2009	Stephen Farrell

Dissemination level	
	Level
PU = Public	PU
PP = Restricted to other programme participants (including the Commission Services).	
RE = Restricted to a group specified by the consortium (including the Commission Services).	
CO = Confidential, only for members of the consortium (including the Commission Services).	

CONTENTS

1 INTRODUCTION

This document describes how to start from the N4C project's [N4C] code repository and end up with a running DTN node and how to subsequently get started on integrating an application with the DTN stack on the node. The reader is assumed to be familiar with DTN generally [DTN] and with the DTN2 reference implementation [DTN2] in particular.

The contact point for the N4C librarian (responsible for managing all repositories) is librarian@n4c.eu and that address should be used for any queries related to this document.

2 GETTING A BOOTABLE IMAGE

The "official" URL is http://code.n4c.eu/ but for now we use the TCD mirror in the examples since that is currently more up to date.

In the examples below, we use the dtnmule image to build up a working DTN data mule on an Asus eee PC 901. Variations for other image types and other platforms will be documented as they are developed.

Assuming you want to use the a dtnmule build (the most complete) then the URL to start from is: http://basil.dsg.cs.tcd.ie/n4c/binary/ and select the binary image that best matches your system. In the following we will describe the situation when the Low Power Intel Architecture (LPIA) is selected, in this case based on the file dtnmule-installusb-090609.img.

Generation of these image files currently requires manual intervention by maintainers. In future, this will be replaced with a mixture of stable and nightly builds in future. The Moblin image-creator tool [MOBLIN] is used to create these images. There are separate images for DTN routers, mules and gateways and (later) for different platforms and with different installation options.

The images in these directories are installable Linux images that contain the operating system, the DTN stack and the applications that are currently integrated with the DTN stack. This includes various software components that may not be required for some kinds of DTN node, for example, Squid (a web cache) is often included but is only required for DTN nodes offering delayed web access (e.g. DTN routers).

It is your responsibility to know how to deal with these images – they are intended for system administrators and not for typical end users. Installing such an image on an inappropriate target platform may render the system unusable, that is: "brick" the system. Trinity College Dublin and the N4C project generally offer no warranties in this respect, in particular:

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT

HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

The reader should also pay particular attention to warnings below marked with "CAUTION!!" since those are steps that can be particularly damaging if done wrong.

3 INSTALLING THE IMAGE

In overview the process to follow is as follows:

- download the image and usually to load it onto a USB stick
- reboot the target platform from the USB stick and/or install the image from the USB stick
- boot the DTN node
- configure the DTN node
- test the DTN node

In the following we show the various steps as run from a Linux command line.

Step 1: Download the image file to a local file (note that it will be of the order of 500 MB).

```
$ wget http://basil.dsg.cs.tcd.ie/binaries/dtnmule-installusb-090609.img
$ ls -l
total 504832
-rw-r--r-- 1 user user 516947968 2009-06-03 15:14 dtnmule-installusb-090603.img
```

Step 2: Copy the image file to an at-least 2GB USB stick using the 'dd' command (usually as root or via sudo), and making sure you pick the right output file ('of=') to be your USB device, in the case below that is /dev/sdd1. The "bs=" input specifies the block size and should usually be 4096. The USB stick needs to be at least 2GB because during the install process a temporary file system is created on the disk that requires 1GB of free space.

CAUTION!! Picking the wrong output file can destroy the target device! Be sure to pick the correct USB device. You can check which it is by using 'df -h' with the USB stick inserted and missing and noting which device maps to the USB stick.

```
$ sudo umount /dev/sdd1
$ sudo dd bs=4096 if=dtnmule-installusb-090609.img of=/dev/sdd1
126208+0 records in
126208+0 records out
516947968 bytes (517 MB) copied, 138.533 seconds, 3.7 MB/s
$
```

You may need to extract and re-insert the USB stick before its contents show up properly. When copying has worked, and the USB stick has been remounted, you should see contents like the following:

```
$ ls -l /media/E\:/
total 441400
-rw-rw-rw- 1 root root   5312512 2009-06-03 14:36 bootfs.img
-rw-rw-rw- 1 root root       203 2009-06-03 14:36 boot.msg
-rw-rw-rw- 1 root root   3347301 2009-06-03 14:33 initrd0.img
-rw-rw-rw- 1 root root       150 2009-06-03 14:36 install.cfg
-rw-rw-rw- 1 root root      6385 2009-06-03 14:36 install.sh
-r--r--r-- 1 root root     13941 2009-06-03 13:36 ldlinux.sys
-rw-rw-rw- 1 root root 441475072 2009-06-03 14:36 rootfs.img
-rw-rw-rw- 1 root root       119 2009-06-03 14:36 syslinux.cfg
-rw-rw-rw- 1 root root   1798808 2009-06-03 14:36 vmlinuz
```

Note that this results in an unexpected capacity (432MB, 88% used) being shown for the USB stick (via 'df -h'). This is a result of the way the .img file is built – essentially that file may be a full file system image and includes the total size of the file system. This is normal and need not be of concern.

Step 3: Install the OS on the DTN device.

CAUTION!! You may brick your device if you do this wrong.

Insert the USB stick in your DTN device and reboot. Depending on the existing operating system on the DTN device you may need to tell the BIOS to boot from the USB stick. Some devices (including the Asus eee PC 901) may not boot from the USB stick every time. Extracting the USB stick and doing a cold-boot seems to resolve this though you may need more than one attempt. You may also have to try various USB ports and to hit the “ESC” key in order to trigger a boot selector.

On rebooting from the USB stick the system will display the following warning:

```
Welcome to N4C: I'm a data mule! (for an eee PC 901)

Install USB Image. This will DESTROY all content on your hard drive!!
 - To boot default vmlinuz-2.6.24-16-lpia kernel, press <ENTER>

boot:
```

CAUTION: As the warning says, this is where you will at least temporarily brick your device so be VERY careful.

Once you press return you will see lots of Linux command output eventually followed by:

```
Install Successfully
Unplug USB Key, System Will Reboot Automatically
```

But, sometimes, even so, you get right back to where you started! This can happen if the device and the image used don't match. For example, if your device's internal disk layout is sufficiently different from that assumed by the image, then copying the OS from the USB stick may fail, and the original OS will still be in place on the device. To fix this, select the appropriate image. ***We'll have to build up knowledge here of what works and what doesn't.***

Once the install finally works then you will see the Ubuntu splash screen on reboot and can login to the system.

The default password for 'root' is 'password'. There is also a 'dtnuser' account created (with the same password) that is used for running DTN processes.

CAUTION: You should change the default root and dtnuser password and replace the SSH keys that are included in the image with ones that make sense for you.

To make a basic check that your system is properly installed you can simply rund the DTN2 daemon 'dtnd' – you will get an error that we'll fix in a while but at least know that the binaries are installed and can run on your system.

```
$dtnuser: dtnd
...various errors...
[1242132568.455735 /dtnd error] error initializing data store, exiting...
$dtnuser:
```

On a newly installed system like this all N4C content is stored in the “/data” directory which contains three subdirectories: 'apps,' 'dtn' and 'scripts' and consumes (as of May 2009) about 250 MB of disk storage (as reported by 'du -sh /data') as shown below. Other than this directory, the installed image consists of a fairly standard Linux install.

```
$ cd /data/
$ tree -d -L 2
.
|-- apps
|   |-- DTN2
|   |-- LTPlib
|   |-- ProPHET-2.6
|   |-- cgic205
|   `-- oasys
|-- dtn
|   |-- cache
|   |-- conf
|   |-- log
|   `-- storage
`-- scripts
```

Add more here later about the various interesting scripts there etc.

4 INITIALIZING AND CONFIGURING DTN2

You will next probably want to configure networking on the device. For N4C testing we are currently using the following IPv4 ranges for different classes of device. Contact the N4C librarian if you wish to use an IPv4 address in one of these ranges within an existing N4C testbed. As can be seen these addresses are all private, everything “behind” the gateway device (which is also connected to the Internet) is in the range 10.125/16. At this time, we do not use IPv6 addressing in N4C.

Node Type	Address Range
DTN Gateway	10.125.10/24
DTN Router	10.125.11/24
DTN Mule	10.125.12/24
DTN Router AP	10.125.13/24

Figure 1 – IPv4 addresses used in the N4C tests

In some cases, the use of the address ranges above is merely a convenience (though quite helpful for debugging), however in the case of DTN gateways and routers it is important that addresses do not collide, so be sure to request a specific address in that case. If the addresses above aren't suitable for some reason, then it is possible to use others, simply contact the N4C librarian with your specific request if necessary.

If you have been allocated one or more IPv4 addresses you must configure this as the static IP address for the node by editing the usual '/etc/network/interfaces' file in the normal manner.

For a DTN gateway or router, most likely you will need to configure the external address on its wireless interface which is done by modifying the /etc/network/interfaces file.

Once basic networking is set up on the device then we are ready to start to initialize and configure the DTN2 daemon.

The generic DTN2 manual [MAN] is really intended for application developers but includes manual pages for DTN2 configuration files and is recommended reading.

Since DTN2 creates some default endpoint identifiers (EIDs) based on the UNIX host name, you should use the hostname command to set the host name to a name of the form:

node-NNN-MMM.village.n4c.eu

where NNN is the network number (e.g. 11 for routers, 12 for mules) and MMM is the host number, that is, for a node with the IP address '10.125.11.101' set its name to 'node-11-101.nodes.n4c.eu'. The automatically configured EID for that node will then be 'dtn://node-11-101.nodes.n4c.eu.dtn'

Note that this naming scheme is subject to change as DTN naming work progresses in the project. For the present, all we need is uniqueness and a way to communicate the EID based on the assigned IPv4 address.

The DTN2 configuration file, '/etc/dtn.conf' automatically makes use of the hostname (via the 'hostname' command) and so doesn't need to be changed for the DTN2 daemon to know its own EID.

The default dtn.conf file is set up to auto-discover other nodes and mules so that should be all the configuration required in order to start working in the DTN. General information on dtn.conf settings may be found by running the DTN2 daemon (dtnd) in interactive mode and using the help command. The dtn.conf file itself also contains comments that are useful.

5 APPLICATION INTEGRATION

There are three main ways in which developers might integrate their applications with the DTN2 implementation of the bundle protocol. The first and typically simplest is to use the dtnsend and dtnrecv applications that are part of the DTN2 release. The second is to use the dtntunnel application in order to tunnel bundles over the DTN but where the application is unaware of the use of the DTN. The last is to use the DTN2 application programming interface (API) to directly link their application to the DTN2 library.

5.1 Using dtnsend and dtnrecv

We will use the example of playing correspondence chess via the DTN. Bear in mind that the purpose of this is to demonstrate the steps required to use the DTN in an application and is not to produce a perfect gaming experience! This example demonstrates how to use DTN for applications that are essentially mail like and that provide good interfaces to allow scripts to be called around and from within, the application.

The application that we will use is xboard [XBOARD] with a modification of its cmail scripts to use dtnsend and dtnrecv instead of using standard email. In the example below we are using xboard-4.4.0.alpha5. [XB4.4]

A version of xboard with the modifications described below is available from:

http://down.dsg.cs.tcd.ie/misc/dtnchess.tgz

Note that this is not a “release” in any sense, at this stage it really only illustrates how to use the DTN applications in a fairly basic manner. This version has some hardcoded pathnames embedded in the code, which are needed in order not to disturb any existing xboard installation. (The current stable version of xboard is 4.2.7 so its best not to do a 'make install' with the 4.4.0 alpha version at this stage.) You will need to fix those hardcoded pathnames.

To start, first install gnuchess so that you can test xboard. The following command should work:

```
$ apt-get install gnuchess
```

You may need to set an http-proxy environment variable in order to get apt-get to work if you are behind a firewall.

Next download (from [X44] or the above URL) and build xboard, the build process goes as follows:

```
$ autogen.sh
$ ./configure
$ ./make
```

Note that you may need to install X11 headers and various other development packages in order to build xboard. Follow the usual ./configure process until you are done. To verify that xboard is working, simply run the xboard command, telling it to use the gnuchess engine, and you should see a chess board on your X11 desktop.

```
$ ./xboard -fcs gnuchessx
```

Note: if you are ssh'ing in to a development machine you may need to use:

```
$ ssh -Y username@host
```

This allows the development machine to securely open an X window on your desktop.

The cmail tool [CMAIL] is a perl script that calls xboard and sends and receives (and parses) emails that contain chess games. We have produced a small wrapper for cmail (called dtnchess) and some a script that uses dtnsend as the program cmail uses to send “email.” Instead of sending an email to a mail address this simply uses dtnsend to send the chess game file to an endpoint identifier (EID). Similarly, dntvchess uses dtnrecv to receive game moves. We now describe this set up so that you can do the same kind of integration for other applications.

The are two scripts required (dtnchess and dtnchess_send), the first one starts the game and the second is called from within cmail in order to actually send the bundles containing chess positions. To start, you simply run 'dtnchess', or if you are black you run 'dtnchess black'.

As black, dtnchess first registers with its local DTN2 daemon (dtnd) using the registration EID dtn://$hostname.dtn/chess_black. The game board is not displayed and the script simply waits until white has made, and sent, a move. One the first move is received then the script starts xboard and the game is displayed. The actual call that dtnchess makes to dtnrecv for this is:

```
$ dtnrecv -n 1 -N -o $tmpfile -d dtn://$hostaname.dtn/chess_black
```

This call establishes the connection with the dtnd, so that bundles for the EID will be delivered as required. The other inputs instruct the dtnrecv application to exit after one bundle has been received. Once the bundle arrives, the temporary file is moved into the $HOME/Chess directory where xboard can pick it up.

As white, dtnchess displays the board and allows the user to make a move. Once you have moved, then you choose the “Mail Move” option from the file menu. At that point, the dtnchess_send script is called and sends a bundle containing the game position from the EID dtn://$hostname.dtn/chess_white to the destination EID which is dtn://$hostname.dtn/chess_black. The actual call made to send the bundle is:

```
$ dtnsend -s dtn://$hostname.dtn/chess_white -d dtn://$hostname.dtn/chess_black \
        -t f -p $posfile
```

The astute reader will notice that this will only work out-of-the-box when both users are on the same host. To change the EID of the other player, you must currently modify the script with the correct other player's EID. The scripts have various variables to allow such changes to be made.

Once a player has received a bundle, then they must currently choose the “Reload Cmail Message” from the xboard file menu in order to load the new position into xboard. Each time a position is sent, then a new instance of dtnrecv is started to wait for the next bundle.

All of the above required about 1 man-day's effort including selecting the chess application, setting up the DTN from scratch on the host and integrating the application and the DTN. We plan to make a proper release of DTN Chess in future, which should be the first DTN game release, one interesting part of which will be to properly integrate the chess clocks supported by xboard!

5.2 Using the dtntunnel Application

For applications that are already delay tolerant or are running in environments that are subject to disrupted connections (and where the application doesn't gracefully handle those disruptions), the dtntunnel application provides a possible way of using DTN in order to make the application work.

This integration method, while more limited than the scripting based one above in various respects, can be more suited for closed-source or hard to modify applications, especially if those use standard protocols so that the application's behaviour can be verified.

As an example of this kind of integration, we set up DTN tunnels between various hosts as shown in the diagram below and ran the HTTP protocol over those between instances of the Squid HTTP proxy. This set up would be suitable for use for a testbed where the remote nodes were unreliably connected via some frequently-disrupted wide area network, for example, a WiMAX rural network working at excessive distances that might be often disrupted by weather.

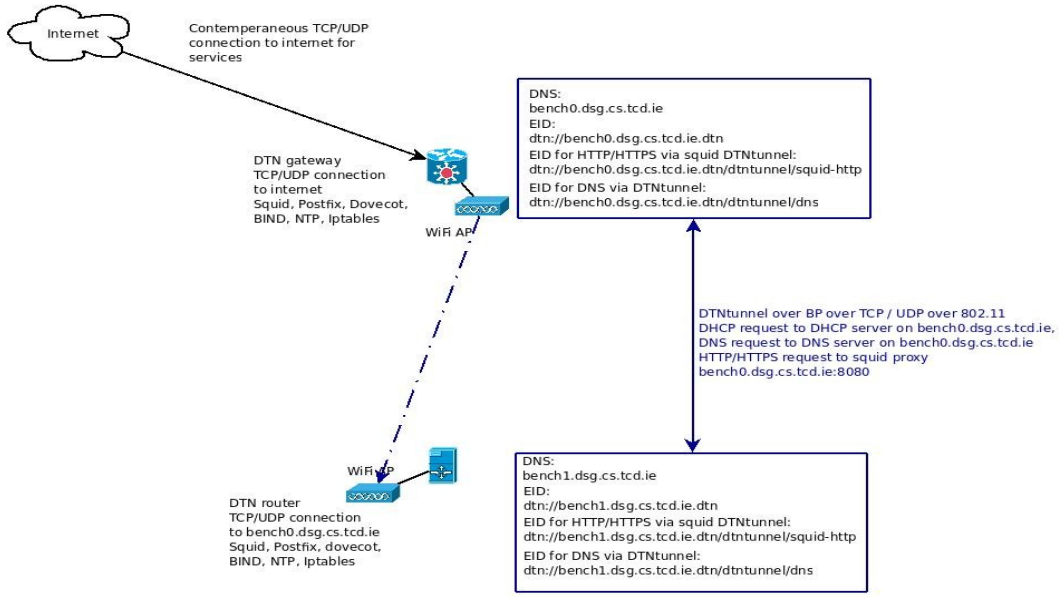

Figure 2 – A dtntunnel configuration.

In order create such a set up, one must of course run DTN daemons on the relevant nodes, and in addition, one must start dtntunnel instances that listen for application traffic on the non-disrupted interfaces and that then encapsulate application traffic (one UDP packet or the output of a read on the socket per bundle) into bundles for transmission over the disrupted link. Upon bundle-reception, the other side of the tunnel decapsulates the bundle and then forwards the traffic to the receiving application. The commands below show how such tunnels are established.

```
# start tcp squid listener
sudo -u dtnuser /usr/bin/dtntunnel -L -d -o /data/dtn/log/dtntunnel_squid_tcp_L.log
--local_eid dtn://bench0.dsg.cs.tcd.ie.dtn/dtntunnel/squid_tcp

# start tcp squid dtntunnel
sudo -u dtnuser /usr/bin/dtntunnel -T bench0.dsg.cs.tcd.ie:18080:bench1.dsg.cs.tcd.ie:8080
dtn://bench1.dsg.cs.tcd.ie.dtn/dtntunnel/squid_tcp -d -o
/data/dtn/log/dtntunnel_squid_tcp_T.log
```

There are clearly some limitations as to when such tunnels can usefully be used. Firstly, applications that require privileged ports (1-1024) require the bundle daemon to be run as root, or with setuid permission, which is undesirable for security reasons. There are ways to avoid this by using complex IP handling in the UNIX kernel, but the situation rapidly becomes complex and sometimes creates hard-to-diagnose problems.

The other main limitation with this set up is that it can only handle significant delays to the extent that the application itself handles those. One N4C developer characterised this approach as a way to “stretch” the application rather than really integrating the application the DTN.

Having said that, the clear benefit of tunnels, where they work, is that they require no changes to the application itself.

Readers who are interested in this approach can find a number of sample configurations at:

http://down.dsg.cs.tcd.ie/misc/dtntunnel-squid-http.zip

5.3 Using the DTN2 API

Finally, if one is developing a new application, or is able to devote more significant effort to integration with the DTN, the DTN2 implementation has a DTN API that developers can use in order to directly send and receive bundles. That C++ API may be found in the 'applib' folder of the DTN2 implementation and for example defines C structures and C++ classes like the following:

```
/**
 * Bundle metadata. The delivery_regid is ignored when sending
 * bundles, but is filled in by the daemon with the registration
 * id where the bundle was received.
 */

struct dtn_bundle_spec_t {
        dtn_endpoint_id_t source;
        dtn_endpoint_id_t dest;
        dtn_endpoint_id_t replyto;
        dtn_bundle_priority_t priority;
        int dopts;
        dtn_timeval_t expiration;
        dtn_timestamp_t creation_ts;
        dtn_reg_id_t delivery_regid;
        dtn_sequence_id_t sequence_id;
        dtn_sequence_id_t obsoletes_id;
        struct {
                u_int blocks_len;
                dtn_extension_block_t *blocks_val;
        } blocks;
        struct {
                u_int metadata_len;
                dtn_extension_block_t *metadata_val;
        } metadata;
};
```

There are also a number of applications built using this API in the DTN2/apps directory, including the dtnsend and dtnrecv applications referred to earlier.

5.4 Other Possibilities

At the time of writing the above are the methods for integrating applications into the DTN with which the authors are familiar. There are however a number of other DTN implementations that may be better suited for other applications or other environments. A (somewhat) up-to-date list of those is maintained on the DTN research group's web site at:

http://www.dtnrg.org/wiki/Code

6 REFERENCES

[N4C] http://www.n4c.eu/

[DTN] Farrell, S., & Cahill, V., "Delay- and Disruption-Tolerant Networking," ISBN 1-59693-063-2, Artech House, 2006.

[DTN2] http://www.dtnrg.org/

[MAN] http://www.dtnrg.org/docs/code/DTN2/doc/manual/

[MOBLIN] http://moblin.org/

[XBOARD] http://tim-mann.org/xboard.html

[XB4.4] http://savannah.gnu.org/projects/xboard/

[CMAIL] http://www.tim-mann.org/xboard/xboard_10.html

[NODE] N4C Deliverable D5.1 – DTN node design.