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

D7.3 N4C Specification and Implementation of Integration Platform

Pedro Nunes Institute
António Cunha, Paulo Freitas
Postal address: Rua Pedro Nunes, 3030-199, Coimbra, Portugal
Tel: +351 239700934
Fax: +351 239700912
www.ipn.pt
cunha@ipn.pt
freitas@ipn.pt

ABSTRACT (Max 400 word)

Starting in May 2008, N4C is a 36 month research project funded by the Seventh Framework Programme (www.cordis.lu/fp7). This is a cooperation project between end-users, in Swedish Lapland and Kočevje region in Slovenian mountain, and technological partners (RTD performers and companies). Under the scope of the project we will design and experiment with an architecture infrastructure and applications in field trials and build two test beds.

This document is one of the outcomes of the integration and testing purposes that has been made under WP7 tasks. It is a collection of the know-how that integration teams obtained from exploring, integrating and testing the developed modules from some of the partners, in order to have tests performed by others than the developers, anticipating the scenario that will occur when the end users download and try N4C applications. The purpose of the document is to provide a document that can be read by anyone willing to configure a DTN with the applications that are being developed and will be deployed at the end of this project, in a very linear and easy to understand way. The objective is that the end users can use the applications that are being developed by N4C several partners and get, in the end, a well-behavioured system with properly integrated applications, deployed as services and that can be reached in certain places of the DTN in a centralized and standard way using, for the effect, a software application as the platform in which services can be accessed. Moreover, the integration platform should not offer incompatibilities with the different hardware types that have been used by consortium to develop their applications, and also the devices that the end users may use during the scenarios already provided on previous deliverables. Such issues could compromise the success of the integration platform due to the concept of universality and abstraction not being present.

Throughout this document, will be addressed topics such as: the objective of this deliverable; the importance for the N4C project; specification of the integration platform; future work.

Due date of deliverable: 30/04/2010 Actual submission date: 15/06/2010

Document history		
Status	**Date**	**Author**
Initial Draft based on DOW	31/05/2010	António Cunha, Francisco Barbosa, Paulo Freitas
First draft circulated to consortium	04/06/2010	António Cunha, Paulo Freitas
Feedback	07/06/2010	Karl Johan Grøttum
Integrated Material related with the Hybrid Simulation Platform	13/06/2010	António Cunha, Paulo Freitas, Krzysztof Romanowski
Submission to EC	16/06/2010	Dr Maria Udén

Dissemination level	
	Level
PU = Public	X
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).	

CONTENT

1. INTRODUCTION

1.1 PURPOSE

This document's purpose is to gather information from all the partners in N4C project and, together with the developed software and the hardware solutions defined along the project, replicate subsystems from the project, with the final goal to have all modules operating together. This way, WP7's participants intend to integrate partner's modules and perform at the same time independent tests which will reinforce the real-life tests performed and contribute to find any bugs or incorrect behaviors. These tests are important because the success of the integration process by developing a common platform largely depends on the modules being able to operate smoothly.

The current document is intended to be a "beginner's guide" for N4C applications and at the same time explain the "why" and the "how" of the integration process. It will be demonstrated that the objective of integration of the modules has many advantages, not only for user's usability but for achieving a higher modularity, components' interconnection and creating a higher abstraction of all N4C modules. The details about the specification of the work done so far, and the technical knowledge needed to install, start running and using it, will be reported. Also, some examples and scenarios regarding the usage of the integration platform will be presented in order to demonstrate what functionalities are already implemented. Concerning the technical level, for those who wants to setup the system or wants to set modifications (at code level or configuration level), all the dependencies that need to be satisfied in order to run correctly the integration platform, are described as well the respective locations for download.

People who wish to set a DTN and some of N4C applications (performed by consortium) can find in this document, which is public, every bit of information needed, like where can users download the required data from, what applications/modules do they need and how can they configure and use applications, preferently deploying DTN modules as services and using both application's frontend and the developed integration platform.

According to what has been established in the consortium meeting at Dublin, in September of 2009, a new approach to the integration and testing processes is going to be performed. When talking about the integration that was initially planned, opinions diverged about what to use as an integration platform and how to use it. So, using an integration platform like D-Bus was deprecated, and instead was decided to use IPN and ITTI efforts to take information and modules that each partner is developing

and aggregate them into specific hardware, like computers, netbooks and PDAs, which conveys two purposes: make developed modules work together in one machine and, at the same time, modules are tested in a different environment from the one in which they were developed, which can be faced as independent tests.

One last remark: one of the possible outcomes of testing and aggregating different modules in the same hardware is a guide that can later be used by general users that are doing their first steps in setting up a DTN, which can be a positive contribution to the dissemination and use of this project's final result.

1.2 INTENDED AUDIENCE AND READING SUGGESTIONS

This document is intended for all types of users. However, at the time of this document's writing there are already several DTN tutorials available in the Internet, but the majorities are written for technical developers and persons with high skills on computer-related subjects. So, with this document, the N4C consortium desires to extend DTN services to "not-so technical" computer users.

It is recommended that readers also read other documents related with N4C project so that they can be more contextualized with the technology that is being developed, as well as the technical objective of each partner.

1.3 REFERENCES

1. **Davies, Elwyn.** *D2.1 - N4C System Architecture v0.4*. Folly Consulting. 2009.
2. **Farrel, Stephen.** *D5.1 - N4C Node Design.* TCD / Intel. 2009.
3. **Cunha, António and Barbosa, Francisco.** *D7.2.1 - Laboratory Testing on Integrated Subsystems.* Instituto Pedro Nunes. 2009.
4. **Farrel, Stephen.** *D4.1 - N4C Generic DTN Documentation.* TCD / Intel. 2009.
5. **Cunha, António and Barbosa, Francisco.** *D7.2.2 - N4C Laboratory Testing on Integrated Subsystems*. Instituto Pedro Nunes. 2010.
6. **UML.** Unified Modeling Language. http://www.uml.org
7. **Python.** Python Programming Language. http://www.python.org/
8. **Maemo SDK.** Maemo SDK for Developers. http://maemo.org/development/sdks/
9. **Arne-Wilhelm Theodorsen, Sigurd Sjursen, Karl Johan Grøttum.** WP3 Plan for summer test 2010 – Hiker's PDA
10. **Norut N4C Wiki for how to setup a tablet with all the applications required:** http://trac.itek.norut.no/n4c/wiki/MaemoFormatting

11. **Hiker's app install for maemo linux on Nokia N810 (as normal user):**
http://trac.itek.norut.no/n4c/attachment/wiki/MaemoFormatting/N810_instalation_normal.sh
12. **Hiker's app install for maemo linux on Nokia N810 (as root):**
http://trac.itek.norut.no/n4c/attachment/wiki/MaemoFormatting/N810_installation_root.2.sh
13. **Configuration File for Hiker's Daemon:**
http://trac.itek.norut.no/n4c/attachment/wiki/MaemoFormatting/n4c.conf

1.4 ABBREVIATIONS

CCR	Communications Challenged Region
DTN	Delay Tolerant Networking
LI	Legacy Internet
LPIA	Low Power on Intel Architecture
SBC	Single Board Computer
IP	Integration Platform

2. NETWORK ARCHITECTURE AND MACHINES

Typically, when setting a DTN, this is composed of multiple client machines, the DTN nodes, that can communicate with a central DTN node, named DTN router. The DTN router collects the requests made by the DTN nodes and then sends the requests to a "portable" device, the data mule, that takes requests to some place where a machine with two network interfaces establishes a bridge between the DTN enclave and the Internet, the DTN gateway. On the other hand, the DTN gateway resolves the pending requests and receives the answers to those requests, answers which are carried back to the DTN router which then passes them to the prime nodes. This is the typical scenario of exchanging messages between the elements of a DTN network.

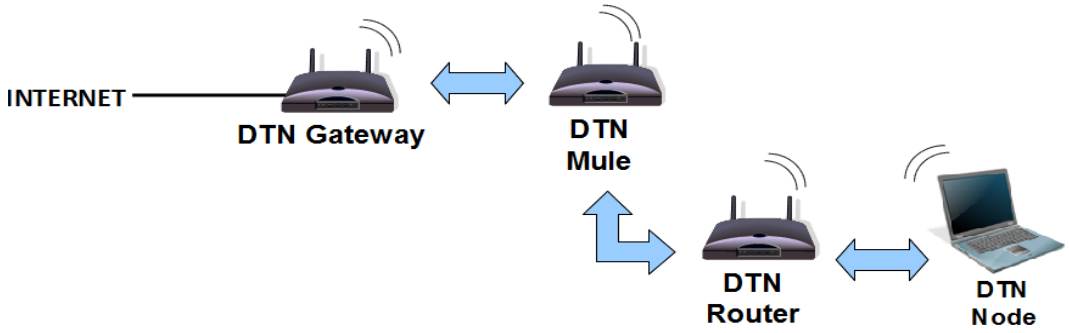

Figure 1 - A typical DTN configuration

This scenario, depicted in the previous figure, can have several variations, since DTN does not require (and it is not) a static scenario. For instance, the system is flexible enough to suppress the DTN gateway, since the data mule can suffer some modifications and starting behaving as a chameleon node, detecting whether it is in a DTN zone, and consequently it collects requests or exchanges bundles (bundles are to DTNs what packets are to IP networks) with other nodes, and once it senses that communication with Internet is available, then it resolves the requests and stores the responses.

The core of N4C is DTN, a network topology that allows computers in such network to communicate between them and with the traditional Internet users and machines independently of the delay that communications may have [3, 4, 5]. Delays in DTNs are significantly bigger than in the Internet, which can vary from a few hours to several days, as opposed to few milliseconds of nowadays Internet. Such delays are, however, a small price to pay in such areas, if it can provide a reliable and best-effort service.

The following picture Figure 2 depicts the topology of a DTN. It shows how the individuals interact with the network and how the information flows through its elements. All kinds of vehicles and travelling people are important to serve as mules and thus carry with them the information to the remotest places. These individuals are responsible for obtaining information from Legacy Internet and download it into the DTN network answering the pending requests, and also for making the link in the opposite direction, this means, bring the requests from DTN to the LI. Basically, they are the elements responsible for making the bridge between the content available on the Legacy Internet with the contents of the DTN network.

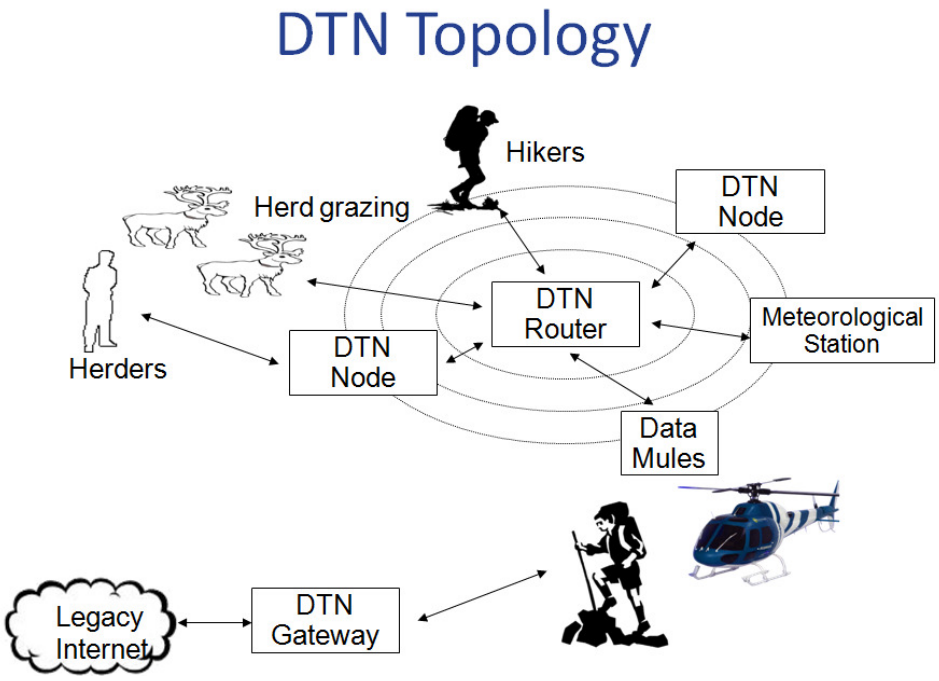

Figure 2 - DTN Topology

Whatever scenario is going to be used, the N4C consortium has suggestions on what type of machines can be used for each node, not only because the development has been made in such hardware, and therefore software could be optimized and some conflicts detected and solved, but also because the choice that has been made was properly reasoned upon several factors.

What follows are the different types of nodes that can constitute a DTN network with the respective description. The definition of each module that are being developed and updated by the consortium, and the discussion about possible hardware platforms that might be used and different nodes types, are exposed on [5]. Then, it is discussed the different types of hardware that is being used by the consortium.

2.1 TYPES OF NODES

In what concerns to the nodes, in the interest of performing tests that get as close as possible of the real-life use, some of the networks nodes thought for N4C have to be replicated. According to [1] and [2], we can find the following types of nodes in N4C:

• Pure Legacy Node

These types of nodes are standard nodes used within Legacy Internet (LI). Despite not being able of using DTN functionalities directly, they can operate in CCR enclaves without any modification in its operating system and/or protocols, although being restricted to existing applications that can operate unchanged in DTN regions, like the email application.

• DTN-Only Node

Nodes expected to be in pure DTN regions, which do not need LI connectivity. These nodes use the DTN infrastructure and can be deployed as a router node, providing store-and-forward capabilities. These nodes can be applied in isolated sensor stations, or in places which travelers almost certainly will pass by. To get a running DTN node, testing team can follow the instructions in http://info.n4c.eu/sympa/d_read/n4c-tech/tcd/n4c-tcd-007-d4.1-FINAL.pdf. An example of a DTN-only node is a DTN router, a node that is designed to be used in summer villages as the core of a local network, and that may be powered through a battery or a solar panel.

• Chameleon Node

These nodes are able to operate in LI, DTN or even in CCR enclaves and its behavior varies according to the place where the node is operating, but in a way that is transparent to users. According to its location, these types of nodes will use Internet protocols to communicate with other LI or CCR nodes whenever they are in LI or CCR enclaves, and will use DTN mechanisms when operating in CCR regions. An example of a chameleon node is the mules that are going to be used to carry information, which can be a PDA or a netbook.

• Gateway Node

These gateway nodes include nodes operating in the boundaries between LI and CCR regions where DTN protocols are used and nodes in the boundaries between CCR enclaves and the rest of CCR regions. Gateways will require one or more logical pair of interfaces: one for LI protocols and the other for DTN protocols. Together with the pair of interfaces, it is also needed a mechanism to mediate communications between the different zones and different interfaces. The gateways nodes are standard computers, which do not have to face connection or power supply problems or limitations.

2.2 TYPE OF HARDWARE

In N4C project, the hardware varies according to the type of node that is intended to be deployed.

• DTN ROUTER (DTN NODES)

In [2] is proposed a design to a DTN router, composed of board (SBC), network, power supply and enclosure modules:

- SBC:
  - Eurotech Proteus, or
  - Mikrotik RB411
  - Kronton nanoETXexpress-SP
- Network: Engenius EMP-8603 wireless card
- Power Supply:
  - Sunshine Solar FastFIX 20w12v (Solar Panel), and
  - Camden 5085329 7AH 12V gel
- Enclosure: Eurobox IP66 cabinet

This node may run Ubuntu Linux 8.04 with the kernel 2.6.24-lpia[1] (for the PROTEUS board) or OpenWRT Kamikaze (for instance, with MikroTik board), and its main role it to act as hotspot to village users. In what concerns to applications, these nodes will primarily provide e-mail (act as MTA, with Postfix/Exim and Dovecot) and web (Apache and Squid) services. The DTN router will also provide DNS through Bind9. Configurations for a DTN router running Ubuntu 8.04 are available at http://basil.dsg.cs.tcd.ie/code/DTN-gateway-doc/file/566861cbee4d/build/dtnrouter%20build%20version%201.01. An interesting approach to another DTN node could also be to use the Asus WL-500G router with OpenWRT firmware (MEIS already mentioned it, and they could be contacted in order to know if they were succeed or not). In a later stage, testing teams can think of replicating the power supply, since these nodes, ideally, would have minimum needs of human interaction in what concerns to power.

• STANDARD PC (GATEWAY OR PURE LEGACY NODES)

Standard PCs can serve two purposes in the network: act as a gateway or as a pure legacy node.

[1] Low Power Intel Architecture

In order to configure a computer as a gateway, a document is available with some configuration at http://basil.dsg.cs.tcd.ie/code/DTN-gateway-doc/file/566861cbee4d/build/dtngateway%20build%20version%201.00.

• ASUS EEEPC 901 (CHAMELEON NODE)

These netbooks can act as data mules or as DTN nodes. The way these nodes can be configured is described in [3] and in the wiki, at the 2009 summer tests section (at the present time, the wiki is offline, so it is impossible to check the correct URL). Some configurations to Asus eeePC 901 data mule are available at http://basil.dsg.cs.tcd.ie/code/DTN-gateway-doc/file/566861cbee4d/build/dtnmule%20build%20version%201.00.

• NOKIA N810 (CHAMELEON NODE)

Nokia tablets will have combined behavior, since it will operate as a DTN node and as a mule. Its operating system in Maemo, a linux distribution, and will have installed the "Hiker's Applications" and NORUT AutoDiscovery module, for ad-hoc communication between hikers. In the section 6 it is exposed further information on how to setup a tablet with all the applications.

• OTHER DATA MULE (CHAMELEON NODE)

Much similar to the devices being used in helicopters during the tests, a SBC-like device can be used as a data mule. For this purpose, a configuration like the one presented in section 0 or even the Asus WL-500 option can be used.

3. INTEGRATION PROCESS

The change that was decided to perform in the integration process leads to a modification in the strategy that was initially defined. As it has been explained in [5], this change implies that, when performing the integration, previously it is necessary to execute tests and elaborate the documentation of the installation steps for each module or service. It is only possible to achieve successfully an integration among the various components that are being developed during the N4C project, when each of them can be installed and start running without issues. This is a basic step and at the same time crucial. The Figure 3 depicts the design of the approach for the integration process.

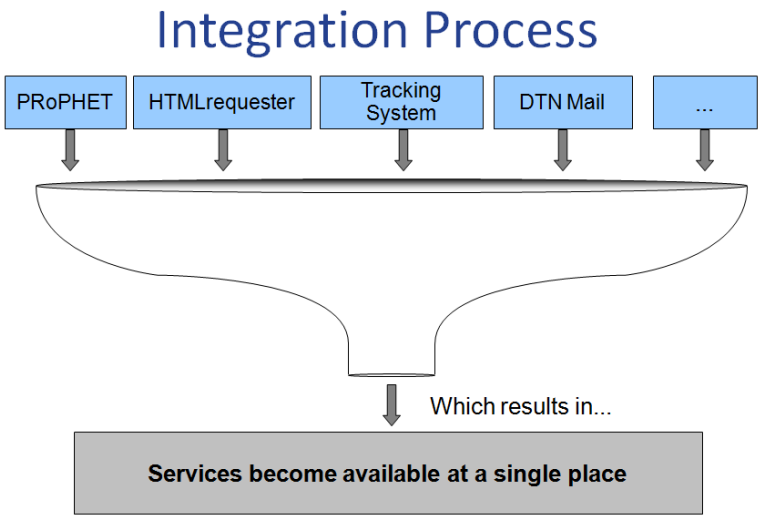

Figure 3 - Integration Process Design

Referring to section 6 from [3] – "Subsystem Test Planning" – and, more specifically, to the methodology that is going to be used, it is required to change the integration perspective from black-box to gray-box perspective, i.e., the integration teams will need to have some knowledge, albeit limited, of each module that is going to be integrated.

In order to perform integration, the two responsible entities for the integration, IPN and ITTI, will retrieve the modules that are being developed by N4C's partners, perform independent tests and, at the same time, will try to gather different modules from different developers into single platforms. We think that it might also be a good idea to duplicate certain equipment, like chameleon or DTN-only nodes, and try to imitate the behavior of users who pass from legacy internet to a CCR, exchange data with other users in the DTN, and return to the legacy internet. This process requires however certain type of hardware that has to be bought, and IPN and ITTI will combine to see who buys which hardware, and consequently, who tests what.

The information collected from partners is going to be used in the process of getting things to work in hardware pieces. To achieve this objective, it is required to define first what kind of nodes are going to exist in laboratory tests, what is their role in the network, which hardware will they require and which software they need in order to run the modules properly, in an attempt to replicate, in laboratory, the nodes that are going to be deployed, but this time with some integration made.

In the next section, will be exposed the information about the modules that will be integrated in the beta version of the common platform that will be, for the first time, presented for N4C consortium and for the general community. Later on this deliverable (section 6) will be explained, step by step, the dependencies and the actions needed to install and execute correctly each of these modules / services. Following the principles of the Integration Process that is now implemented, as final objective, we plan to achieve higher modularity and abstraction in a way that allow end users might develop their services and integrate them on the platform.

3.1 COMPONENTS INFORMATION

The tables below show the modules that are currently integrated in the integration platform. The missing services and modules developed by the consortium that are not available yet on the platform are going to be integrated in future versions of it.

Module Number	**1**
Module	DTN Stack
Responsible Partner(s)	LTU / TCD / Folly (?)
Source Location	*http://code.n4c.eu/code/DTN2/*
Compile notes	Instructions in Annex II
Software Requirements (Dependencies)	Compiling in linux: libraries *tcl-dev*, *libdb-dev* and *libxerces-c2-dev*.
Platform in which the software is being developed/used	Linux (Pure Legacy, DTN Only, Chameleon and Gateway Nodes)
Platform in which the module is going to be tested	Regular PCs with Ubuntu 8.04 Hardy and in Xephyr Nokia N900 simulator with Maemo

Module Number	**2**
Module	PRoPHET
Partner(s)	LTU; TCD
Source Location	*http://info.n4c.eu/code/PRoPHET*
Compile notes	Instructions in Annex I
Software Requirements (Dependencies)	Module 1 – DTN or DTN2
Platform in which the module is going to be tested	Regular PCs with Ubuntu 8.04 Hardy and in Xephyr Nokia N900 simulator with Maemo

Module Number	**3**
Module	DTNmail
Responsible Partner(s)	Folly
Source Location	*http://info.n4c.eu/sympa/d_read/n4c-tech/folly/dtnmail.tar.gz*
Compile notes	Instructions in Annex III

Software Requirements (Dependencies)	Module 1 – DTN or DTN2
Planned changes	Create a email client embedded into the application itself, instead of launching the predefined email client in pop-up style
Platform in which the module is going to be tested	Regular PCs with Ubuntu 8.04 Hardy and in Xephyr Nokia N900 simulator with Maemo

Module Number	4
Module	Hikers, Hunters, Herders and Rangers Applications
Responsible Partner(s)	NORUT
Source Location	*http://trac.itek.norut.no/n4c/wiki/maemoformatting*
Hardware Requirements	Nokia Tablet
Software Requirements (Dependencies)	Maemo Linux Distribution
Platform in which the software is being developed/used	Mules (Nokia N810/N900)
Platform in which the module is going to be tested	PDA

Module Number	**5**
Module	Web Caching
Responsible Partner(s)	Folly, LTU
Compile notes	Instructions in Annex IV
Software Requirements (Dependencies)	Module 1 – DTN or DTN2, pythonlibwebkit (for the HTML render in the Nokia simulator)
Platform in which the software is being developed/used	Regular PCs with Ubuntu 8.04 Hardy and in Xephyr Nokia N900 simulator with Maemo
Platform in which the module is going to be tested	PCs (Asus eeePC and/or desktops); Routers (ARM) (which will act as nodes or mules); PDA

Module Number	6
Module	Meteorological Application
Responsible Partner(s)	MEIS
Software Requirements (Dependencies)	Module 1 – DTN or DTN2, pythonlibwebkit (for the HTML render in the Nokia simulator)
Platform in which the software is	Regular PCs with Ubuntu 8.04 Hardy and in Xephyr Nokia

being developed/used	N900 simulator with Maemo
Platform in which the module is going to be tested	PCs (Asus eeePC and/or desktops); Routers (ARM) (which will act as nodes or mules); PDA

4. A HYBRID SIMULATION PLATFORM

This section describes a simulation part of the integration platform that can aid in integration testing.

4.1 MOTIVATION

The integration process involves substantial amount of building and testing the various system and application components, on several hardware/software systems, forming parts of diverse (mostly DTN) networks. In order to achieve as broad and extensive testing as possible, it is advantageous to have the systems and networks replicated to many instances, on which different testing/integration teams can perform independent tests. Replication of individual nodes is usually feasible, requiring as much as obtaining – in some cases actually building – hardware equipment and software components, followed by appropriate configuration. For the common case of nodes hosted on the Linux operating system, all that is needed to get a node ready might be just software installation, optionally compiling, and configuration, performed on a commodity PC.

Replicating a network – or just setting up a similar one to a reference site – is more complicated. Apart from the necessity of networking hardware, there is a need to ensure the network operation shows some characteristics typical of CCRs (Communications Challenged Regions), like random or scheduled disruptions of links, ad hoc connections, long and variable delays etc. - without requiring resources (time, space, manpower) comparable to an actual CCR site. While it certainly is possible to simulate such network behaviour in a laboratory setting, the scope of integration and test work can still be limited by the number of physical nodes that can be provided and the time required to manipulate them.

This is a situation where software simulation can prove useful. As the objective of the testing is integration of actual systems and application software, a typical simulation setup - with both nodes and the network being modelled within the simulator - is of limited use, since it is the actual software implementations rather than ideas and mechanisms that is of interest at this stage. Instead, a useful simula-

tion platform should only model what is more resource-consuming to provide: the network and maybe the hardware of the nodes, while using real instances of what is the core of the software integration effort: the actual DTN software.

4.2 POSITIONING

A simulation platform is proposed which can make use of a DTN network software model and actual software implementations of the nodes. It is hybrid in that it can operate on a simulated network and real nodes. The reality of the nodes may concern only their software or extend also to the hardware.

This platform is complementary to the user-level integration platform. While the latter provides users with a common interface framework and can be seen as positioned on top of the various applications being integrated, the former is positioned at the bottom, below the DTN stack and the operating system of the nodes, providing a simulated network plane. It should be noticed that the DTN stack itself is functionally an integration platform; the complete picture of the integration platform comprises those three components, with the specifics of the bottom one – the simulation platform – being its purpose is integration work rather than everyday operations. The relative position of the basic subsystems discussed is shown in Figure 4, with the integration subsystems highlighted.

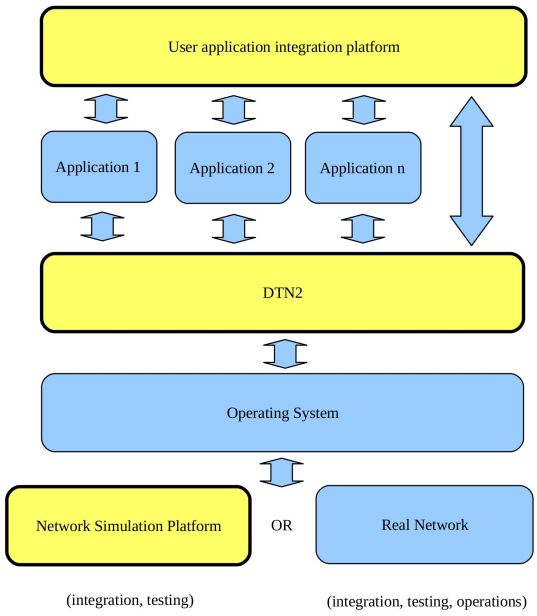

Figure 4 - Scheme of the Integration of Subsystems

4.3 ARCHITECTURE

The simulation platform in its basic form is a collection of software components running on a single host. The components include the host operating system, a network simulator running as a user space application on that operating system, a virtual machine engine also running in user space, and virtual machines running by the virtual machine engine; each of the virtual machines runs a guest operating system and, on top of it, DTN2 network software stack and DTN applications that are subject to testing and integration.

The virtual machines create a native environment for the DTN software, i.e. they provide virtualized hardware on which operating systems can be installed and configured according to the DTN software requirements, with minimal impact of the specifics of the underlying host. The network connecting those virtualized nodes is simulated by the network simulator, with only minimum engagement of the host operating system's network stack. The latter provides only basic connectivity between each individual virtual machine and the simulator, not between the virtual machines. This might be illustrated as an analogy to a switch or router connecting workstations: the simulator plays the role of the switch or router, while the host operating system provides the patchcords (cables) only between the workstations and the switch (router).

Using the network simulator rather than the networking capabilities of the host operating system (which could include e.g. bridging and routing between the virtual machines) gives more flexibility in configuring the simulated network and its operation. The host operating system's networking stack can be used to set up and tear down connections, and – if a sufficiently advanced system is involved – shape the network traffic (e.g. with Linux advanced routing subsystem); with some effort these functions could be scheduled on a time scale. On the other hand network simulators make it possible – in addition to the above functionality – to model communication channel properties (e.g. delays, transmission rates, error and packet loss distributions), mobility patterns of wireless stations, networking device properties at the physical and link layer, etc. - all with detailed scheduling, logging and packet tracing. If required, new models can be constructed and used in simulations.

The proposed simulation platform generally does not use models of nodes that participate in simulated networks, although this is possible and could complement the setup made of virtual machines, e.g. to generate some specific traffic or model non-DTN nodes. Virtualization of node hardware gives the possibility of regular operating system installation and an almost native environment on which to install DTN software; thus it has been preferred to modelling nodes within the simulator or to employing light virtualization solutions (like Linux containers – although this option might be interesting if a large number of nodes are to be run simultaneously).

An example configuration of two virtual nodes connected by a simulated network is shown in Figure 5.

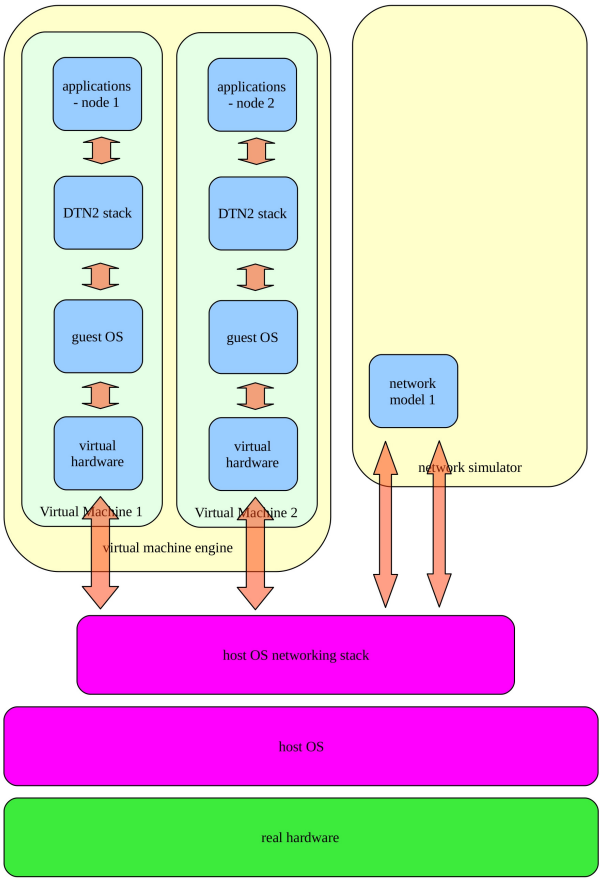

Figure 5 - Two virtual nodes connected by a simulated network

4.4 IMPLEMENTATION

In accordance with most of the other software work in the project, Linux has been selected as the operating system both for the host and the guests. The host OS could have been any other system capable of running a suitable network simulator, while the guest OS choice is clearly determined by the DTN2 implementation. In particular, the host OS variant is Fedora (currently version 12, 64-bit, is used, to be upgraded to the current 13); this choice has been dictated by it being reportedly a primary development platform used for the network simulator chosen (to be described in a sequel). The guest OS variants are Ubuntu (currently – version 9.10, 32-bit, to be upgraded to the current 10.4), as this is the variant predominantly used in the project. This choice limits the node types that can be virtualized to those based

on Linux and Intel x86 or x86_64 architecture. The virtualization engine used makes it possible to install other operating systems as guests (e.g. Microsoft Windows or Sun Solaris), however, the very nature of virtualization (as opposed to emulation) leaves no possibility for other hardware. There are a few substantially different hardware platforms of interest in the N4C project (e.g. ARM used in Nokia N900 smartphones, Broadcom or Atheros used in WiFi routers); those cannot be modelled by a virtualized node on the proposed platform, however, there are other possibilities to integrate them that will be mentioned further in the text. The hardware of the host can be any standard PC or notebook, preferably with a processor with hardware virtualization support, and sufficient size of RAM and disk to allow for many virtual machines.

Of the various virtualization platforms available Sun VirtualBox has been selected as the primary one (version 3.1.8), although other possibilities do exist (e.g. VMWare Player or VMWare Server) and have also been used.

The network simulator selected is ns-3 (http://www.nsnam.org, veresion 3-dev from its Mercurial repository), which comes with a number of channel and device models, including CSMA (akin to Ethernet), WiFi, and WiMax, with the possibility of writing one's own. The simulation setups are being written in C++ and use components from the simulator library; an optional language is Python.

Finally, the DTN2 systems and application software comes from the current N4C tree at http://basil.dsg.cs.tcd.ie.

All the software code mentioned is available as open source under various free licenses (e.g. GPL, Apache), with the exception of VirtualBox, which is available in two versions: closed-source binary, released under the VirtualBox Personal Use and Evaluation License, and Open Source Edition, released as source code under the GNU General Public License, Version 2. The closed-source version has some additional features, the most interesting being VirtualBox Guest Additions (which includes time synchronization, shared folders, shared clipboard, mouse pointer integration) and USB support. The personal use as defined in the closed-source license includes use of the product on the host computer where the user has installed it himself and there is no more than one simultaneous remote client connecting to the host in order to display the guest computers remotely (there is no need for such a feature here). Such use fits a developer's needs more than an end-user's. Should it present a problem, the open-source version or another comparable virtualization platform can be substituted.

4.5 CONFIGURATION

4.5.1 HARDWARE

As already mentioned, the hardware required is a standard PC or notebook, capable of running a Linux operating system with X Window System. A processor with hardware support for vitualization and multiple cores is preferred, as well as large RAM. A 64-bit processor may be required if more than 4GB of RAM is installed, and it can also enable testing of both 32-bit and 64-bit virtual nodes. The system needs a few GB of disk space for the Linux distribution (Fedora 12 in the case described) and 1-2GB for the ns-3 software. The VirtualBox needs tens of MB itself, but the virtual machine disks – one for each machine – must be large enough to install an Ubuntu (a couple of GB) distribution plus DTN systems and application software (several hundreds of MB). In practical terms, 5GB would be enough for a single virtual hard disk, so a typical present-day disk should suffice.

The precise requirements depend on the number of nodes to be run at the same time. A single virtual machine needs 256-512MB RAM and ca. 5GB disk size. Tests have been performed on a notebook with a dual-core processor with hardware virtualization support, 4GB RAM and 250GB disk, running 5 simultaneous virtual machines of 512MB each and 20GB virtual disk (dynamic, with about 4-5GB actually allocated), and have shown that this hardware configuration is sufficient for the case, although the system begins to swap programs from RAM. For more virtual nodes to be tested, a larger RAM and/or tight tuning of the guest configuration would be needed.

As far as basic simulation is concerned, there is no need for actual networking hardware (apart from downloading software or keeping it up to date). However, for testing Internet-DTN gate-way functionality and some advanced network setups, a network interface or two, including a WiFi one – is necessary.

The more simultaneous virtual machines are involved, the more important it is to equip the hardware platform with as large a display monitor (or two) as possible.

There is no special configuring required at the hardware level, besides making sure that the pro-cessor features (multi-core, hardware virtualization support) are enabled in the system BIOS.

4.5.2 HOST OPERATING SYSTEM

A Linux distribution from http://fedoraproject.org does not require much extra configuration. It is necessary to ensure that basic networking works. Also, the following rpm packages are necessary or useful (the required command-line program provided is given in parentheses):

- bridge-utils (brctl)
- tunctl (tunctl)
- net-tools (ifconfig, arp, route, netstat)
- wireless-tools (iwconfig)
- ip-route (ip)

For testing and trouble shooting the following is useful:

- iputils (ping, arping)
- nc (nc)
- tcpdump (tcpdump)

as well as (at least to check whether packets are blocked somewhere):

- iptables (iptables)
- arptables (arptables)
- ebtables (ebtables)

As a number of virtual machines may be involved, the critical resource is RAM on the host: the system should be configured so that the size of RAM used is kept at minimum.

4.5.3 VIRTUAL MACHINE ENGINE

This can be downloaded from http://www.virtualbox.org. A yum repository is available at http://download.virtualbox.org. There is some additional software required:

- qt
- SDL
- dkms

After installing the package, additional virtual network interfaces should be defined for the virtual machines that are going to be created. This can be done via the VirtualBox GUI by selecting Preferences from the File menu; a 'VirtualBox – Settings' pop-up window opens, with a list of categories of settings. Selecting 'Network' from the list produces a 'Host-only Networks' list. Initially the list holds only one item, 'vboxnet0 network'. An item should be added here for each of the prospective virtual nodes, named (automatically) 'vboxnet1 network', 'vboxnet2 network', etc. The IP addresses can be changed here from the default, as the default is common for all the interfaces and – like DHCP server settings, if any – is not appropriate for the simulations planned.

Virtual machines, one per node, should then be created by use of the VirtualBox GUI or the command-line VBoxManage program. A virtual machine should preferably have 512MB RAM, a 10GB dynamic disk, and two network interfaces:

- 'Adapter 1': type 'Host-only'; name 'vboxnet1' for the first virtual node, 'vboxnet2' for the second etc.; adapter type – any of the emulated NICs; the actual test setup uses the one named 'Intel PRO/1000 T Server (82543GC)'; 'cable connected'; this interface shows up as eth0 in the guest operating system and is the one to be connected in the simulated network;

- 'Adapter 2': type 'NAT'; adapter type – same as above; no 'cable connected'; this one is visible as eth1 and is used to 'out-of-band' networking, mainly to connect from time to time to the Internet to download or update software (this is not strictly necessary, as file transfer can be made via folders shared with the host – an operation that does not require network interfaces); during normal testing and simulation this interface should be down.

It is convenient to initially define the virtual disk for the first virtual machine only, which can then be cloned. In order to do this, the first virtual machine should be started and have all the common software (including the operating system and DTN software) installed and initially configured.

After that, with the virtual machine down, the virtual disk can be cloned and the new disks registered with the remaining virtual machines, as described in the 'Sun VirtualBox User Manual'.

There have been reports of problems caused by some Linux distributions employing UUID numbers instead of disk partition names in their references to disk partitions. This type of refe-rencing can be changed to plain partition name references by editing the /etc/fstab and (with Ubuntu) /boot/default/grub files; further details can be found in the /boot/grub/grub.cfg file and on the Ubuntu forums.

In effect, all the virtual machines have their individual virtual disks with the same contents. What needs to be done then is to change their hostnames and IP addresses from within their guest operating systems.

4.5.4 GUEST OPERATING SYSTEM

The detailed configuration of the guest should be done as on a real Intel x86 node. The only specifics are the network interfaces, as described above in the section on the virtual machine engine; eth0 should be assigned address that is appropriate from the point of view of the DTN setup to be tested; eth1 should be normally down, brought up only when necessary – with the address to be acquired from DHCP service on the virtual machine engine. If virtual disk cloning is planned, as much configuration as possible should be done before the cloning.

4.5.5 DTN SOFTWARE ON THE GUEST

The configuration of the DTN software (and any accompanying or prerequisite software) should be done as on a real Linux / Intel x86 node. If virtual disk cloning is planned, as much configuration as possible should be done before the cloning.

4.5.6 NS-3

The installation and configuration (on the host system) should proceed according to the current documentation of the system. The prerequisite software include at least:

- `gcc`
- `gcc-c++`
- `python`
- `mercurial`

and a number of other optional packages. Simulation programs are wrotten in C++ (Python is an option) and run by the waf script (a wscript file may need to be updated with new source file names).

4.5.7 VIRTUAL CONNECTIONS

The idea of the network connecting the virtual nodes is as follows. Each virtual machine has a virtual Ethernet interface eth0, which is visible to its (guest) operating system, and has been allocated a unique (in the simulation address space) IP address. This Ethernet interface is bridged by the virtual machine engine to a virtual interface on the host computer, which the host operating system sees as vboxnet1, vboxnet2, etc. - a dedicated name for each of the virtual machines. These interfaces do not need IP addresses.

The host operating system maintains another set of virtual network interfaces, of a 'tap' type, created by the tunctl command. These interfaces also do not need IP addresses. In the test setup described their names have been defined as i1, i2, etc. Each of them corresponds logically to a virtual machine, although technically they do not have anything common with the virtual machines, until specific commands are executed by the host operating system.

On the other hand, the simulation program run in the simulator creates (for a specific runtime) virtual network devices as internal data structures, which it associates with the host operating system interfaces i1, i2, etc. as so-called 'Tap Bridge' objects, such that each of the i1, i2, etc. forms a bridge with a corresponding network device in the simulator.

In the networking subsystem of the host operating system, the virtual interfaces vboxnet1, vboxnet2, etc. are bridged by the brctl command with the interfaces i1, i2, etc. Thus effectively the interfaces eth0 that the guest operating systems see are bridged to the corresponding network devices in the simulator.

Finally, the task of the simulator is to connect its internal networking devices to an internal communication channel. This makes the virtual nodes (residing on the virtual machines) to be connected.

Only the interfaces eth0 on the virtual nodes need IP addresses; the rest of the configuration simulates a layer-2 connection.

The commands required to build a network of that kind could be the following:

- create tap virtual interfaces and bring them up without IP addresses:

```
tunctl -t i1
```

```
tunctl -t i2
ifconfig i1 0.0.0.0
ifconfig i2 0.0.0.0
```

- create bridges on the host and bring them up without IP addresses:

```
brctl addbr b1
brctl addbr b2
ifconfig b1 0.0.0.0
ifconfig b2 0.0.0.0
```

- connect the tap interfaces to the virtual machine interfaces via the bridges:

```
brctl addif b1 i1
brctl addif b1 vboxnet1
brctl addif b2 i2
brctl addif b2 vboxnet2
```

- create tap bridge objects by running the ns-3 simulator with the following code (fragment only shown):

```
CsmaHelper csma
Ptr<CsmaChannel> chan;
TapBridge Helper tbh;
tbh.SetAttribute("Mode", StringValue("UseBridge"));
NodeContainer nodes;
NetDeviceContainer devs;
nodes.Create(2);
devs = csma.Install(nodes, chan);
tbh.SetAttribute("DeviceName", StringValue("i1"));
tbh.Install(nodes.Get[0], devs.Get[0]);
tbh.SetAttribute("DeviceName", StringValue("i2"));
tbh.Install(nodes.Get[1], devs.Get[1]);
```

The resulting network configuration is shown in Figure 6. For the network frames to actually reach their destinations, they must not be impeded by packet filtering mechanisms in the kernel. Either the rules (manipulated by tools like iptables, arptables, and ebtables) have to accept them to be forwarded, or using the rules by the kernel has to be disabled for bridges by zeroing the relevant entries in /proc/sys/net/bridge/bridge-nf-call-*. Note that although a bridge is a layer-2 device, the FORWARD chain of the IP kernel packet filter (accessed by 'iptables') can block packets traversing the bridge.

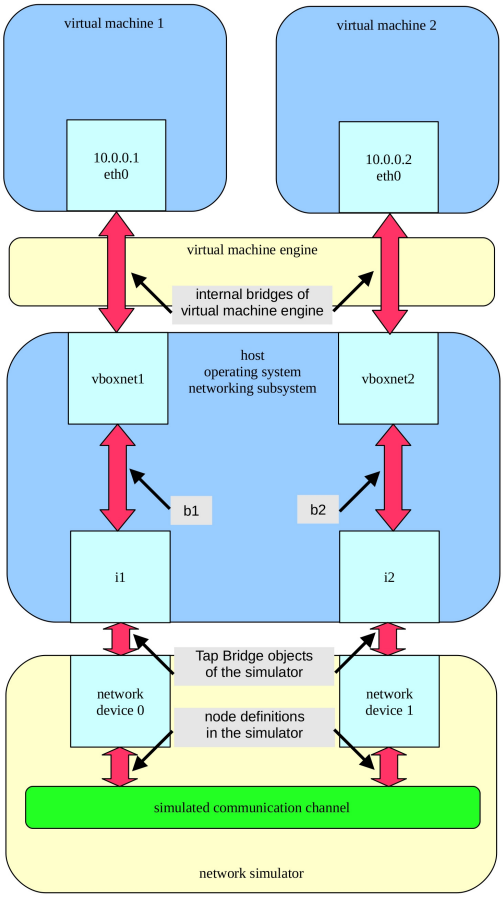

Figure 6 - Network Configuration

4.5.8 CONNECTING REAL HARDWARE

It is possible to connect a real network node to a system of virtual nodes and simulated network. This can prove useful in cases where a node of interest has architecture that cannot be modelled by a virtual machine, perhaps because of a different hardware/system platform, e.g. a Nokia N900 (which is build on an ARM platform).

To integrate such a node in the simulated network, a virtual connection has to be made between a tap virtual interface connected to a Tap Bridge object of the simulator and a host network interface that represents a physical interface (e.g. eth0) rather than a virtual-machine one (e.g. vboxnet1). There are configurations where it is enough to configure a bridge on the host operating system appropriately; such cases follow the idea described in the previous section. However, there are others where bridging

like this does not work, e.g. because of the physical host networking interface not supporting MAC spoofing, and in consequence being incapable of sending to the real node a layer-2 frame with the original source address (which could have been expected from a bridge). In those situations a similar connection can be build by using a pseudo-bridge instead.

The pseudo-bridge changes layer-2 information and only layer-3 packets are passed intact. The two key techniques it uses are proxy ARP – to communicate with the adjacent links without the need to spoof source MAC of the frames it sends – and routing, to actually deliver the packets from one side of the bridge to the other. Rule based routing together with the simple bridge configuration, consisting of only two nodes per bridge, make it possible to forward the packets without considering the layer-3 information, and without assigning IP addresses to the interfaces that would not need them in a classical bridge setup.

The steps required to build such a pseudo-bridge are given below, assuming the interfaces to be bridged are i1 (created by tunctl as described in a previous section) and wlan0 (a WiFi interface):

- turn on proxy ARP functionality on the bridged interfaces:

```
echo 1 > /proc/sys/net/ipv4/conf/i1/proxy_arp
echo 1 > /proc/sys/net/ipv4/conf/wlan0/proxy_arp
```

- turn on packet forwarding so that routing can work:

```
echo 1 > /proc/sys/net/ipv4/ip_forward
```

- define two routing tables by adding the following lines to file /etc/iproute2/rt_tables:

```
1 wlan0-to-i1
2 i1-to-wlan0
```

- populate the new routing tables:

```
ip route add table 1 to default dev i1
ip route add table 2 to default dev wlan0
```

- define rules that determine when to use the new routes:

```
ip rule add iff wlan0 pref 1 table 1
ip rule add iff i1 pref 2 table 2
```

The resulting network configuration is shown in Figure 7.

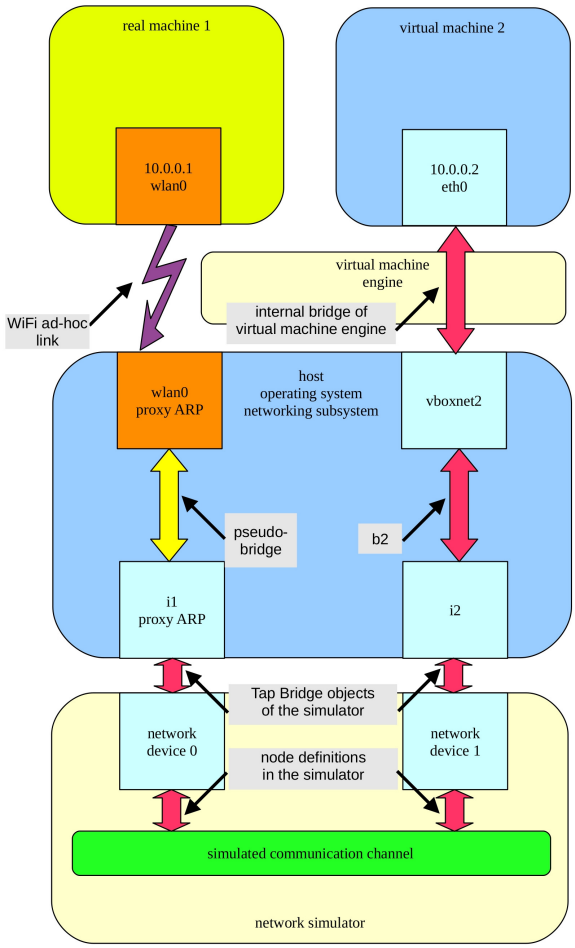

Figure 7 - Network configuration after bridge setup

4.5.9 NETWORK MODEL

Among the various network models already present in the ns-3 source code distribution the most interesting seem to be WiFi and CSMA. The former corresponds to a type of physical link often encountered in real DTN testbeds; the latter, being a simplification of the Ethernet, is a good reference for comparisons.

One of the basic characteristics of a DTN network one would try to model is an ad hoc link, which can go up or down at different times. There is no simple general way of modelling it as a dynamic attribute

at the link level (i.e. not higher than layer 2). In the WiFi case the on/off link state changes can be modelled by defining an appropriate mobility model; when a distance between node locations becomes sufficiently large, the effect is loss of connectivity, which quite adequately models the real-world link-down event. In the CSMA case, there are two specific attributes of NetDevice objects, SendEnable and ReceiveEnable, which can be manipulated to get the on/off effect.

There are other network characteristics that can be modelled by using attributes of the existing ns-3 classes, e.g. delay in CSMA models, or by constructing models of specific phenomena, e.g. propagation error models.

It is possible to develop a custom model for DTN simulations, and this possibility may be fol-lowed in further work if the project needs exceed the capabilities of the existing models.

4.6 TESTING

4.6.1 SIMPLE CONNECTIVITY

In order to test the feasibility of using the simulated network to connect virtual and real nodes, simple network models have been defined based on the existing CSMA and WiFi classes.

The following sample code shows main C++ programs, to be run by invoking the waf tool of ns-3 on the Linux command line as in the following examples:

```
./waf "--run=n4ccsmaonoff --interval=5 --runtime=3600"
./waf "--run=n4ccsmadelay --interval=5 -delay1=1 --delay2=200 \ --
runtime=3600"
./waf "--run=n4cwifi --interval=30 --distance=100 \ --runtime=3600"
```

The arguments following the first one (--run) are optional. The full list of arguments defined in the main program can be obtained by the following command:

```
./waf "--run=n4ccsmaonoff --PrintHelp"
```

It is possible to set a number of other attributes of specific ns-3 objects at program invocation time; the above command produces information on command-line arguments that can be used to find the relevant information.

The CSMA model consists of nodes (in the simulator space) connected to channels. There can be more than one channel, and the number of nodes connected to each are configurable when starting the simulation. This model corresponds to an Ethernet switch with ports grouped onto disjoint sets, with no connectivity between the sets (like in port-based VLANs). The channel delays (common for all transmissions) and on/off times for the individual nodes can be defined at runtime.

The first example defines a simulation of two channels, one with three and one with two nodes (the numbers may be changed at program invocation), with two delays (common to both chan-nels) alternating in time. The delay and change interval can be specified at program invocation.

```
// n4ccsmadelay.cc: an example simulation program for ns-3 network
//   simulator
// simulates a CSMA network with channel delays
// usage: waf "--run=n4ccsmadelay [other options]"
// usage: waf "--run=n4ccsmadelay --PrintHelp"
#include <iostream>
#include <sstream>
#include <string>
#include <ctime>
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/core-module.h"
#include "ns3/helper-module.h"
NS_LOG_COMPONENT_DEFINE("CTest");
using namespace ns3;
const uint32_t KMAX = 10;
void csma_chan_delay (const Ptr<CsmaChannel> c, const Time t)
{
  c->SetAttribute("Delay", TimeValue(t));
}
void csma_chan_delay (const Ptr<CsmaChannel> c, const Time t,
  const std::string msg)
{
  csma_chan_delay(c, t);
  NS_LOG_INFO(msg);
```

```
}
int main (int argc, char *argv[])
{
  LogComponentEnable("CTest", LOG_LEVEL_INFO);
  GlobalValue::Bind("SimulatorImplementationType",
    StringValue("ns3::RealtimeSimulatorImpl"));
  GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
  uint32_t n[KMAX] = {3,2};
  std::string iprefix = "i";
  uint32_t ioffset = 1;
  std::string tbm = "UseBridge";
  double rt = 600.0;
  double ti = 10.0;
  double delay1 = 1.0;
  double delay2 = 100.0;
  CommandLine cmd;
  std::ostringstream ossn, ossmsg;
  uint32_t k;
  for (k=0; k<KMAX; k++)
  {
    ossn.str("");
    ossn << "nodes" << k;
    ossmsg.str("");
    ossmsg << "number (>=0) of nodes on channel " << k << " ["
      << n[k] << "]";
    cmd.AddValue(ossn.str(), ossmsg.str(), n[k]);
  }
  cmd.AddValue("iprefix", "interface name prefix ["+iprefix+"]",
    iprefix);
  ossmsg.str("");
  ossmsg << "first interface number (>=0) [" << ioffset << "]";
  cmd.AddValue("ioffset", ossmsg.str(), ioffset);
  cmd.AddValue("mode",
    "TapBridge mode: ConfigureLocal/UseLocal/UseBridge ["+tbm+"]",
    tbm);
```

```
ossmsg.str("");
ossmsg << "total simulation run time in seconds (>=0) [" << rt
  << "]";
cmd.AddValue("runtime", ossmsg.str(), rt);
ossmsg.str("");
ossmsg << "time interval [s] between changes [" << ti << "]";
cmd.AddValue("interval", ossmsg.str(), ti);
ossmsg.str("");
ossmsg << "channel delay1 [ms] [" << delay1 << "]";
cmd.AddValue("delay1", ossmsg.str(), delay1);
ossmsg.str("");
ossmsg << "channel delay2 [ms] [" << delay2 << "]";
cmd.AddValue("delay2", ossmsg.str(), delay2);
cmd.Parse(argc, argv);
CsmaHelper csma;
Ptr<CsmaChannel> chan[KMAX];
NodeContainer nodes[KMAX];
NetDeviceContainer devices[KMAX];
TapBridgeHelper tapBridge;
tapBridge.SetAttribute("Mode", StringValue(tbm));
NS_LOG_INFO("TapBridge mode: " << tbm << ". Run time limit: "
  << rt << " seconds");
uint32_t istart, kn;
uint32_t i = 0;
for (k=0; k<KMAX; k++)
{
  if (n[k] <= 0) continue;
  chan[k] = CreateObject<CsmaChannel>();
  nodes[k].Create(n[k]);
  devices[k] = csma.Install(nodes[k], chan[k]);
  istart = i;
  for (kn=0; kn<n[k]; kn++,i++)
  {
    ossn.str("");
    ossn << iprefix << i+ioffset;
```

```
      tapBridge.SetAttribute("DeviceName", StringValue(ossn.str()));
      tapBridge.Install(nodes[k].Get(kn), devices[k].Get(kn));
    }
    NS_LOG_INFO("  channel " << k << ": rate "
                << chan[k]->GetDataRate() << " delay "
                << chan[k]->GetDelay() << " " << n[k]
                << " nodes; interfaces "
                << iprefix << istart+ioffset << " to "
                << iprefix << i-1+ioffset);
  }
  for (k=0; k<KMAX; k++)
  {
    if (n[k] <= 0) continue;
    for (i=0; i<rt/ti; i+=2)
    {
      Simulator::Schedule(Seconds(i*ti), &csma_chan_delay, chan[k],
        MilliSeconds(delay1), "delay1");
      Simulator::Schedule(Seconds((i+1)*ti), &csma_chan_delay,
        chan[k], MilliSeconds(delay2), "delay2");
    }
  }
  csma.EnablePcapAll("ctest", true);
  Simulator::Stop(Seconds(rt));
  time_t ct;
  std::time(&ct);
  std::cerr << "SIMULATION START " << std::ctime(&ct);
  Simulator::Run();
  std::time(&ct);
  std::cerr << "SIMULATION END   " << std::ctime(&ct);
  Simulator::Destroy();
  return 0;
}
```

The second example defines a simulation of the same two channels of three and two nodes, this time with nodes being turned off sequentially one after another (the sequences are separate for each channel), alternating in time. The change interval can be specified at program invocation. The resulting sequence of nodes being on and off (which gives the effect of the corresponding links being on and off) is:

Time step	Node 1	Node 2	Node 3	Node 4	Node 5
0	on	on	on	on	on
1	off	on	on	off	on
2	on	on	on	on	on
3	on	off	on	on	off
4	on	on	on	on	on
5	on	on	off	off	on
6	on	on	on	on	on
7	off	on	on	on	off
8	on	on	on	on	on
9	on	off	on	off	on
10	on	on	on	on	on
11	on	on	off	on	off
12	on	on	on	on	on

etc. The code follows:

```
// n4ccsmaonoff.cc: an example simulation program for ns-3 network
//   simulator
// simulates a CSMA network with devices on/off
// usage: waf "--run=n4ccsmaonoff [other options]"
// usage: waf "--run=n4ccsmaonoff --PrintHelp"
#include <iostream>
#include <sstream>
#include <string>
#include <ctime>
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
```

```
#include "ns3/core-module.h"
#include "ns3/helper-module.h"
NS_LOG_COMPONENT_DEFINE("CTest");
using namespace ns3;
const uint32_t KMAX = 10;
void csma_dev_enable (const Ptr<NetDevice> d)
{
  d->SetAttribute("SendEnable", BooleanValue(true));
  d->SetAttribute("ReceiveEnable", BooleanValue(true));
}
void csma_dev_enable (const Ptr<NetDevice> d, const std::string msg)
{
  csma_dev_enable(d);
  NS_LOG_INFO(msg);
}
void csma_dev_disable (const Ptr<NetDevice> d)
{
  d->SetAttribute("SendEnable", BooleanValue(false));
  d->SetAttribute("ReceiveEnable", BooleanValue(false));
}
void csma_dev_disable (const Ptr<NetDevice> d, const std::string msg)
{
  csma_dev_disable(d);
  NS_LOG_INFO(msg);
}
int main (int argc, char *argv[])
{
  LogComponentEnable("CTest", LOG_LEVEL_INFO);
  GlobalValue::Bind("SimulatorImplementationType",
    StringValue("ns3::RealtimeSimulatorImpl"));
  GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
  uint32_t n[KMAX] = {3,2};
  std::string iprefix = "i";
  uint32_t ioffset = 1;
  std::string tbm = "UseBridge";
```

```
double rt = 600.0;
double ti = 10.0;
CommandLine cmd;
std::ostringstream ossn, ossmsg;
uint32_t k;
for (k=0; k<KMAX; k++)
{
  ossn.str("");
  ossn << "nodes" << k;
  ossmsg.str("");
  ossmsg << "number (>=0) of nodes on channel " << k << " ["
    << n[k] << "]";
  cmd.AddValue(ossn.str(), ossmsg.str(), n[k]);
}
cmd.AddValue("iprefix", "interface name prefix ["+iprefix+"]",
  iprefix);
ossmsg.str("");
ossmsg << "first interface number (>=0) [" << ioffset << "]";
cmd.AddValue("ioffset", ossmsg.str(), ioffset);
cmd.AddValue("mode",
 "TapBridge mode: ConfigureLocal/UseLocal/UseBridge ["+tbm+"]",
 tbm);
ossmsg.str("");
ossmsg << "total simulation run time in seconds (>=0) [" << rt
  << "]";
cmd.AddValue("runtime", ossmsg.str(), rt);
ossmsg.str("");
ossmsg << "time interval [s] between changes [" << ti << "]";
cmd.AddValue("interval", ossmsg.str(), ti);
cmd.Parse(argc, argv);
CsmaHelper csma;
Ptr<CsmaChannel> chan[KMAX];
NodeContainer nodes[KMAX];
NetDeviceContainer devices[KMAX];
TapBridgeHelper tapBridge;
```

```
tapBridge.SetAttribute("Mode", StringValue(tbm));
NS_LOG_INFO("TapBridge mode: " << tbm << ". Run time limit: "
  << rt << " seconds");
uint32_t istart, kn;
uint32_t i = 0;
for (k=0; k<KMAX; k++)
{
  if (n[k] <= 0) continue;
  chan[k] = CreateObject<CsmaChannel>();
  nodes[k].Create(n[k]);
  devices[k] = csma.Install(nodes[k], chan[k]);
  istart = i;
  for (kn=0; kn<n[k]; kn++,i++)
  {
    ossn.str("");
    ossn << iprefix << i+ioffset;
    tapBridge.SetAttribute("DeviceName", StringValue(ossn.str()));
    tapBridge.Install(nodes[k].Get(kn), devices[k].Get(kn));
  }
  NS_LOG_INFO("  channel " << k << ": rate "
              << chan[k]->GetDataRate() << " delay "
              << chan[k]->GetDelay() << " " << n[k]
              << " nodes; interfaces "
              << iprefix << istart+ioffset << " to " << iprefix
              << i-1+ioffset);
}
uint32_t j;
for (k=0; k<KMAX; k++)
{
  if (n[k] <= 0) continue;
  for (i=1; i<rt/ti; i+=2)
  {
    j = ((i-1)/2) % n[k];
    Simulator::Schedule(Seconds(i*ti), &csma_dev_disable,
      devices[k].Get(j), "off");
```

```
      Simulator::Schedule(Seconds((i+1)*ti), &csma_dev_enable,
        devices[k].Get(j), "on");
    }
  }
  csma.EnablePcapAll("ctest", true);
  Simulator::Stop(Seconds(rt));
  time_t ct;
  std::time(&ct);
  std::cerr << "SIMULATION START " << std::ctime(&ct);
  Simulator::Run();
  std::time(&ct);
  std::cerr << "SIMULATION END   " << std::ctime(&ct);
  Simulator::Destroy();
  return 0;
}
```

The WiFi model consists again of nodes connected to channels in a similar fashion as with the CSMA model. This corresponds to two ad-hoc WiFi networks operating in different channels (frequencies). Defining disjoint channels does not have a practical application in the examples presented; it is shown here for the illustration of simulation possibilities.

The example defines a simulation of two channels, one with three and one with two nodes (the numbers may be changed at program invocation). Within each channel, the nodes are located sequentially in a line, separated initially by a common distance from one another. At runtime the nodes move in turns. A node that takes its turn to move changes its location to that of its neighbour, then returns to its original place. The location patterns of the nodes for the initial time steps are as shown in the following table. The groups of locations 1-3 and 4-5 are independent (they belong to different channels). In each group, the locations are equidistant, so their coordinates might be:

- location 1: (0, 0, 0)
- location 2: (d, 0, 0)
- location 3: (2d, 0, 0)
- location 4: (0, 0, 0)
- location 5: (d, 0, 0)

Where d is the distance given to the simulator. The table below shows the ID numbers of the mobile nodes which are at specific fixed locations.

Time step	Location 1	Location 2	Location 3	Location 4	Location 5
0	1	2	3	4	5
1	-	1, 2	3	-	4, 5
2	1	2	3	4	5
3	1	-	2, 3	-	4, 5
4	1	2	3	4	5
5	-	1, 2	3	-	4, 5
6	1	2	3	4	5
7	1	-	2, 3	-	4, 5
8	1	2	3	4	5

etc. The distance separating the locations and the change interval can be specified at program invocation. If the distance specified is sufficiently large, the effect is no connectivity between the nodes except in the time slots when two nodes are collocated. This can model connectivity provided in DTN networks by data mules. The specific example shown can be thought of as illustrating (for the first channel) a data shuttle carrying packets from location 1 to location 3 and then back to location 1. The example code follows:

```
// n4cwifi.cc: an example simulation program for ns-3 network
//   simulator
// simulates a WiFi network
// usage: waf "--run=n4cwifi [other options]"
// usage: waf "--run=n4cwifi --PrintHelp"
#include <iostream>
#include <sstream>
#include <string>
#include <ctime>
#include "ns3/simulator-module.h"
#include "ns3/node-module.h"
#include "ns3/core-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-model.h"
```

```
#include "ns3/helper-module.h"
NS_LOG_COMPONENT_DEFINE("WTest");
using namespace ns3;
const uint32_t KMAX = 10;
const uint32_t IMAX = 20;
const uint32_t SPMAX = 20;
const uint32_t TSMAX = 100;
uint32_t ioffset = 1;
uint32_t istart[IMAX];
std::string iprefix = "i";
Vector3D* nodepos[IMAX][TSMAX];
void move_nodes (const NodeContainer* nc, const uint32_t k,
                 const uint32_t ts,
                 const bool logpositions)
{
  Vector3D* v;
  uint32_t i, j;
  if (logpositions)
    NS_LOG_INFO("  movement " << ts << " channel " << k);
  for (i=0,j=istart[k]; i<nc[k].GetN(); i++,j++)
  {
    v = nodepos[j][ts];
    ((nc[k].Get(i))->GetObject<MobilityModel>())->SetPosition(*v);
    if (logpositions)
      NS_LOG_INFO("    " << iprefix << j+ioffset << " at " << *v);
  }
}
void move_nodes (const NodeContainer* nc, const uint32_t k,
  const uint32_t ts)
{
  move_nodes(nc, k, ts, false);
```

```
}
int main (int argc, char *argv[])
{
  LogComponentEnable("WTest", LOG_LEVEL_INFO);
  GlobalValue::Bind("SimulatorImplementationType",
    StringValue("ns3::RealtimeSimulatorImpl"));
  GlobalValue::Bind("ChecksumEnabled", BooleanValue(true));
  uint32_t n[KMAX] = {3,2};
  std::string tbm = "UseLocal";
  double rt = 600.0;
  double dist = 1.0;
  double ti = 10.0;
  CommandLine cmd;
  std::ostringstream ossn, ossmsg;
  uint32_t k;
  for (k=0; k<KMAX; k++)
  {
    ossn.str("");
    ossn << "nodes" << k;
    ossmsg.str("");
    ossmsg << "number (>=0) of nodes on channel " << k << " ["
      << n[k] << "]";
    cmd.AddValue(ossn.str(), ossmsg.str(), n[k]);
  }
  cmd.AddValue("iprefix", "interface name prefix ["+iprefix+"]",
    iprefix);
  ossmsg.str("");
  ossmsg << "first interface number (>=0) [" << ioffset << "]";
  cmd.AddValue("ioffset", ossmsg.str(), ioffset);
  cmd.AddValue("mode",
   "TapBridge mode: ConfigureLocal/UseLocal/UseBridge ["+tbm+"]",
```

```
 tbm);
ossmsg.str("");
ossmsg << "total simulation run time in seconds (>=0) [" << rt
  << "]";
cmd.AddValue("runtime", ossmsg.str(), rt);
ossmsg.str("");
ossmsg << "initial distance [m] between nodes [" << dist << "]";
cmd.AddValue("distance", ossmsg.str(), dist);
ossmsg.str("");
ossmsg << "time interval [s] between movements [" << ti << "]";
cmd.AddValue("interval", ossmsg.str(), ti);
cmd.Parse(argc, argv);
WifiHelper wifi = WifiHelper::Default();
wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager",
 "DataMode", StringValue("wifia-54mbs"));
YansWifiChannelHelper chanhlp = YansWifiChannelHelper::Default();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default();
NqosWifiMacHelper mac = NqosWifiMacHelper::Default();
mac.SetType("ns3::AdhocWifiMac");
Ptr<YansWifiChannel> chan[KMAX];
Ptr<YansWifiChannel> dummychan = chanhlp.Create();
Ptr<Channel> dch = dummychan;
NodeContainer nodes[KMAX];
NetDeviceContainer devices[KMAX];
MobilityHelper mobility;
mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel");
Ptr<ListPositionAllocator> pos[KMAX];
TapBridgeHelper tapBridge;
tapBridge.SetAttribute("Mode", StringValue(tbm));
NS_LOG_INFO("TapBridge mode: " << tbm << ". Run time limit: "
  << rt << " seconds");
```

```
Vector3D spacepoint[SPMAX];
uint32_t j, kn, kninc, sp, ts;
uint32_t i = 0;
for (sp=1; sp<SPMAX; sp++)
  spacepoint[sp] = Vector(sp*dist, 0.0, 0.0);
for (ts=0; ts<TSMAX; ts++)
  for (j=0; j<IMAX; j++)
    nodepos[j][ts] = &spacepoint[j];
for (k=0; k<KMAX; k++)
{
  if (n[k] <= 0) continue;
  chan[k] = chanhlp.Create();
  phy.SetChannel(chan[k]);
  nodes[k].Create(n[k]);
  devices[k] = wifi.Install(phy, mac, nodes[k]);
  pos[k] = CreateObject<ListPositionAllocator>();
  istart[k] = i;
  NS_LOG_INFO("    channel " << k << " initial node locations:");
  for (kn=0; (kn<n[k])&&(i<IMAX); kn++,i++)
  {
    pos[k]->Add(*(nodepos[i][0]));
    NS_LOG_INFO("      " << iprefix << i+ioffset << " at "
      << *(nodepos[i][0]));
    ossn.str("");
    ossn << iprefix << i+ioffset;
    tapBridge.SetAttribute("DeviceName", StringValue(ossn.str()));
    tapBridge.Install(nodes[k].Get(kn), devices[k].Get(kn));
  }
  mobility.SetPositionAllocator(pos[k]);
  mobility.Install(nodes[k]);
  kn = istart[k];
```

```
  kninc = -1;
  for (ts=1; ts<TSMAX; ts+=2)
  {
    nodepos[kn][ts] = nodepos[kn+1][ts];
    j = kn - istart[k];
    if (n[k]>2)
    {
      if ((j >= n[k]-2) || (j<=0))
        kninc = -kninc;
      kn += kninc;
    }
  }
  NS_LOG_INFO("  channel " << k << ": " << n[k]
              << " nodes; interfaces "
              << iprefix << istart[k]+ioffset << " to "
              << iprefix << i-1+ioffset);
}
for (ts=0; ts<TSMAX; ts++)
  for (k=0; k<KMAX; k++)
  {
    if (n[k] <= 0) continue;
    ossmsg.str("");
    ossmsg << "channel " << k << " nodes moved";
    Simulator::Schedule(Seconds(ti*ts), &move_nodes, nodes, k, ts,
      true);
  }
phy.EnablePcapAll("wtest", true);
Simulator::Stop(Seconds(rt));
time_t ct;
std::time(&ct);
std::cerr << "SIMULATION START " << std::ctime(&ct);
```

```
  Simulator::Run();
  std::time(&ct);
  std::cerr << "SIMULATION END    " << std::ctime(&ct);
  Simulator::Destroy();
  return 0;
}
```

Tests have been performed on the example configurations, with virtual nodes (hosted in virtual machines) as well as real-world ones (external computing devices connected to the simulation host computer via an ad-hoc WiFi network). The results show the proposed platform is capable of modelling various connectivity patterns in a manner transparent to IP packets.

4.6.2 DTN APPLICATIONS

As the default DTN configuration uses a TCP or UDP convergence mechanism, to the simulation platform the traffic generated looks just as any IP traffic. Test have been performed on the simple configurations described above, both the mostly connected (as in the CSMA examples) and the mostly disconnected ones (as in the WiFi case). Tests of simple applications (including dtnping, dtnsend, and dtnrecv) show the platform can model DTN network connectivity.

For further integration testing, larger models have to be built. It is straightforward to enlarge the DTN network modelled by defining additional virtual nodes. As noted above, with 4GB RAM on the simulation host at least 5 virtual nodes can be run without problem; 8GB – an amount that can be easily available – should make it feasible to run 12-14 nodes simultaneously; this makes it possible to run a simulation of a scenario shown in Figure 8.

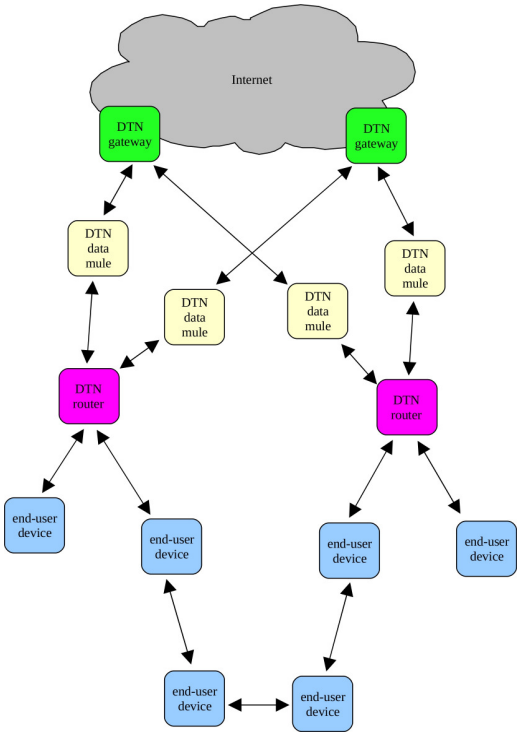

Figure 8 - Representation of Simulation's Scenario

Any virtual node can be substituted for by a real-world device, provided that device has a working DTN stack and applications together with the lower networking stack layers installed, with the data link layer being compatible to the corresponding layer of the simulation platform host.

5. INTEGRATION PLATFORM

5.1 INTEGRATION OBJECTIVES

After the testing phase that IPN was responsible to perform within the scope of WP7, it is time to integrate all the software that was developed by the different project partners in a common ground where all the applications and services can coexist in a way that allows the final user to take advantage of all the potential on the N4C project. This common ground will permit also that the user can enjoy with all the services deployed without being an expert technician. This kind of users will just need to start the platform on their preferred device, and on it, they will find the deployed modules, having the control

over the services by just pushing buttons and setting up preferences with a usable and nicely interface. On the other hand, thinking about the general community where there are always technicians who want to develop their own services, or even to improve the actual components provided by the N4C project, the integration platform might be modular and be an abstraction layer such that new modules can be integrated easily. The next deliverables will be focused on the fully integration of all the modules and provide the tools for others members and individuals contribute with new applications, technically enriching further the project. With this in mind, this document describes the integration platform that is being developed, its main goals and the reasons that lead us to choose the technology that we are using to developed the platform.

The developing process of the integration platform depends on the collaboration of all the partners involved on producing software for the project due to the gray-box perspective that we are following. Tutorials and other documentation about the architecture and implementation details of the developed solutions are important in order to speed up the understanding of the implementations, of the logical structure of the components and of the inputs/outputs to get them communicate with the upper layer (the common platform) and the other modules. This deliverable is a step forward towards this objective.

The main goal that was identified concerning the usability of all the N4C services and applications was the ability of working with the system as a whole, integrated in a unique platform where the final user can access all the features without the knowledge of how the system works or necessary configurations. With an integration platform, users are able to get not only a functional DTN but also the capability of using N4C's developed applications. However, we think that the project can benefit a lot if it has as an alternative to each module's front-end an interface that enables users to access services in a more homogenous way, preferentially one that frees users from having to use a specific hardware, operating system or even the necessity to install specific software in their laptops.

5.2 TECHNOLOGY OVERVIEW

The decision making process of what technology to use to implement the integration platform took in consideration all the software modules that must be integrated, all the different technologies and programming languages that were used in developing them, the devices that can be used in the DTN Network (refer to section 2), interoperability and speed of development.

Our first approach was to use a web-based application. The main upside of this approach is the fact that nowadays virtually every device used to communicate has the ability to connect to a data network

(regular computers, PDA's, Tablets, etc). A web-based application is cross-platform, requires no installations and is widely compatible with the majority of communication devices. But some technical questions soon came to mind when considering the implications of this type of platform, mainly because of the web-based nature itself. An important characteristic is that we can't get a web-based application to execute processes on the host device, which is a requirement. Also, there are at least 2 stand alone applications that were developed that can't be directly integrated in a web-based platform (Not So Instant Messenger and Hiker's Applications). We feel the need to integrate all the services and applications available in the project in a single platform, and a web-based application has constraints that invalidate this requirement.

Our second approach was based on the operating system used by pretty much every partner in the project: Linux. We choose to use Python Programming Language [7] as the developing programming language. The reasons are simple: it's free, it's available for every major operating system and the same source code will run unchanged across all of them. This is very important to us, since the application has to be executed in Nokia N810 or N900 tablets. The Application Programming Interface (API) of those devices are based on Python, and due to the portability and abstraction characteristics of such programming language, we can develop the integration platform under a regular computer and then port it to the Nokia Tablets, which also run a distribution of Linux (Maemo) which assures us that we can maintain compatibility with few or no changes at all. Therefore, developing an integration application in Python allows us to fulfill our main goal, which is to integrate every application and service developed within the project's scope. Also, using Python it is possible to develop web applications increasing then the possibility of performing new interfaces, returning back to the first approach, and giving to new contributors more space to their creativity.

5.3 PLATFORM OVERVIEW

The following drawing shows a diagram of the architecture of the integration platform. There are three major blocks in the second layer that represent the different technologies that were used to develop the project modules and services. Using Python it is possible to interoperate all the applications deployed until now and which are already available. All these are integrated in the application in a way that is transparent to the final user.

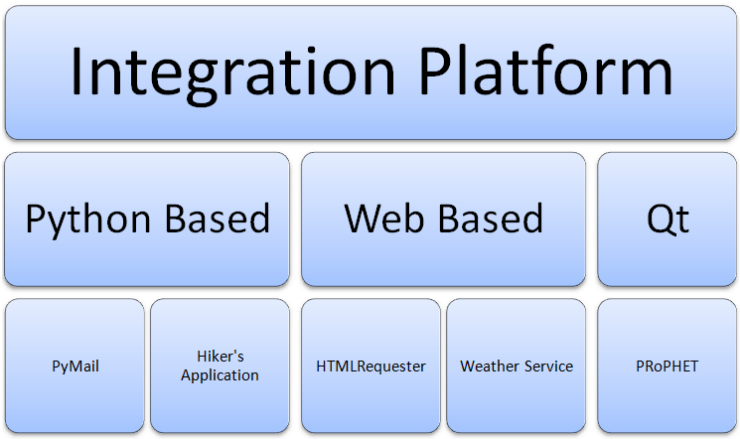

Figure 9 - Blocks Diagram relating each module of the Integration Platform

The interface of the integration platform is composed by 3 main tabs. One of the tabs, named "About N4C", exposes some information about the N4C project. The tab "Partners" has the official links to the respective partner of the project. Finally, the main tab "Services" has the control of all the modules that were already discussed on this deliverable. More tabs are dynamically created when a new service is called as exposed on Figure 14 or the external applications are quickly opened up without the user knowing where the applications are located in the system. Moreover, it is possible to configure the platform which gives the user the possibility of using other applications other than the predefined ones, for example, the user can execute Thunderbird instead of Evolution which is the predefined email client for a regular Ubuntu desktop distribution. A configuration feature was included, and it is explained in detail later on in this section.

Regarding the services, the actual version of the platform enables the usage of the web services available, the HTMLRequester and the Weather Service as depicted on Figure 14. It also permits to control the DTN process, this means, the user can stop or run it when he wants to operate on a Legacy Internet zone or on a DTN zone. More effort will be oriented in order to interoperate even more the other modules.

The following images show the first draft of the application layout:

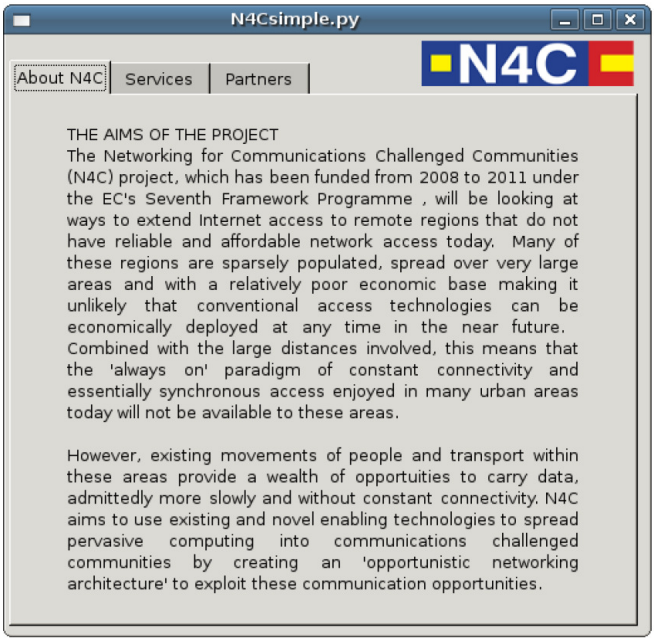

Figure 10 - About section of the application

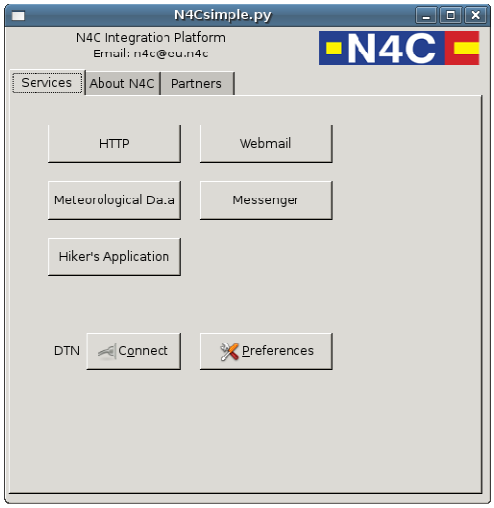

Figure 11 - Services section of the application

The Figure 11 is the main tab in the platform. It includes all the services and applications integrated in the platform as well as a configuration feature. The user can also connect and disconnect the DTN daemon if he wishes to do so. Both 'HTTP' and 'Meteorological Data' buttons will dynamically open a new tab in which a HTML page will be rendered, simulating a simple regular internet browser. In this version of the platform, the 'Webmail' service is not completely integrated in the platform, meaning that when the user chooses to use this service, the predefined email client will be executed via a system command line command. The predefined email client can be configured by the user if he wishes to do so. The preferences feature was included to allow the user to configure some parameters concerning the locations of some of the DTN applications and services, meaning that the platform can be executed in a system independently of how and where the DTN services where installed. The following Figure shows a screenshot of the preferences window:

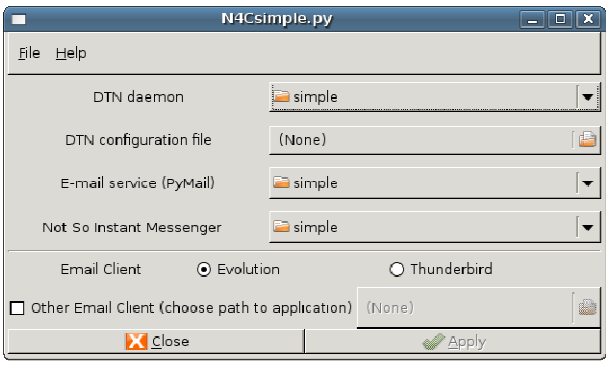

Figure 12 – Preferences Window

The preferences window allows the user to configure the location of 3 services: DTN daemon, PyMail and Not So Instant Messenger. The user just has to choose the folder where these services are installed in his system. This feature also allows the user to select between Evolution and Thunderbird email clients, or else it allows him to manually configure the system to use another email client, by selecting the check-box and choosing the executable of the email client he wishes.

Figure 13 - Partners section of the application

'Partners' tab has got a list of all the project partners, and allows the user to view their web pages by clicking in the links. A new tab is dynamically created with the respective site if this is available in the router. If not, a new HTTP request is made in the HTMLrequester service.

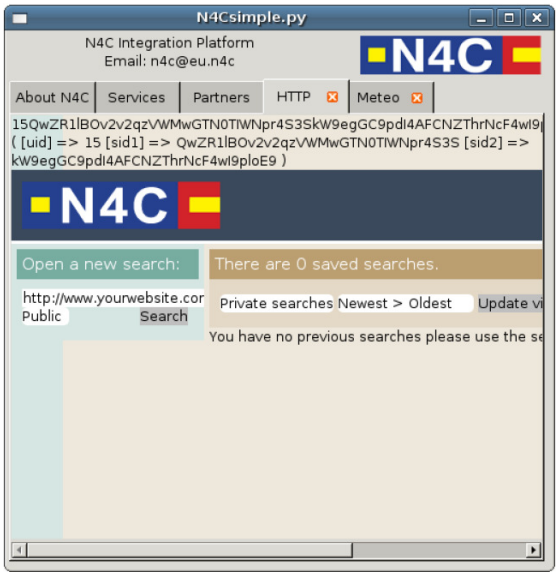

Figure 14 - HTMLrequester embebed example

Like it was already said, the HTTP service opens a new tab directly in the HTMLrequester sevice. The user can now make new web pages requests, close this new tab and open several of them simultaneously. All the HTML pages are rendered using WebKit Python libraries.

In the first stage of the development, the main concerns that are being considered are the usability of the application in both regular computers and the Nokia tablets. The simulator that is being used to test the application is Xephyr and it is available on [8]. This emulates a Nokia N900, and runs Maemo OS. Note that this is a first draft of the application. Both the layout and some features should suffer changes in the weeks to come.

We experienced some difficulties to port the application to the Nokia tablet simulator because of the visual design of it. These difficulties can be caused by the resolution of the Nokia Tablets, by different Python versions running, by the simulator not be able to handle correctly the porting of the source code. Future work will have to consider these implications, or else the platform won't look the same when it's executed in both regular computers and the Nokia Tablets. However, if these difficulties are confirmed on real scenarios, a feasible solution might be to create a separate interface design especially dedicated to the Nokia Tablets. This should not require major development effort since the interface is being developed using Glade, and very easily changes can be made without changing the core of the

platform. Please refer to section 5.4 to know more details. The following drawing shows a draft of how the platform will look in the Nokia Tablets.

Figure 15 - Screenshot of the platform running in the Nokia Tablet simulator

Future development will assure that almost every software can be integrated directly in the application, instead of executing some features in a pop-up style. This will probably be the case for the Not So Instant Messenger application. As for HTMLrequester we are using the WebKit library to embed the web-page into the application itself like we have shown in the previous chapter of this document. With PyMail, in this first version of the application we execute Evolution in pop-up style, but in the future we will develop an embed simple email client to replace the use of Evolution which may not be present on all platforms where the application will work.

5.4 DEVELOPING IN GLADE INTERFACE DESIGNER

Glade Interface Designer was the software chosen to develop the layout of the Integration Platform. Glade is a RAD tool to enable quick and easy development of user interfaces for the GTK+ toolkit and the GNOME desktop environment. It allows an interface to be designed and then exported to XML, which can later be dynamically loaded by applications due to the use of GKTBuilder GKT+ objects. To export the glade user interface to XML we use the gtk-builder-convert application. The syntax is the following:

```
gtk-builder-convert <input_glade_file> <output_xml_file>
```

This creates the XML file which can be loaded into a Python application, making it easy to control all the widgets and features created on the user interface. After the XML is been generated, the developer can now load it into his application using GKTBuilder GKT+ objects. The following excerpt of code shows this process in a python application:

```
builder = gtk.Builder()

builder.add_from_file('N4CintegrationPlatform.xml')

self.window = gtk.Window(gtk.WINDOW_TOPLEVEL)

self.window = builder.get_object("mainWindow")
```

Note: gtk-builder-convert expects a <glade-interface> tag in the XML file and Glade Interface Designer creates the XML with a <interface> tag. So the developer has to change this manually so that gtk-builder-convert can accept the glade file.

6. APPLICATIONS

All the tutorials in this section were produced around February and March 2010, so the instructions contained in them are relative to the versions used at that time. Some problems may already been solved, so whoever uses the tutorials should have this in mind when following the instructions. The tutorials were produced while installing the project different modules and services in regular desktop computers with the following characteristics:

• Kernel

```
Linux jmachado-laptop 2.6.33-020633-generic #020633 SMP Thu Feb 25 10:59:18 UTC
2010 i686 GNU/Linux
```

• GCC:

```
Using built-in specs.
Target: i486-linux-gnu
Configured with: ../src/configure -v
--enable-languages=c,c++,fortran,objc,obj-c++,treelang --prefix=/usr --enable-
shared
--with-system-zlib --libexecdir=/usr/lib --without-included-gettext
```

```
--enable-threads=posix --enable-nls --with-gxx-include-dir=/usr/include/c++/4.2
--program-suffix=-4.2 --enable-clocale=gnu --enable-libstdcxx-debug --enable-
objc-gc
--enable-mpfr --enable-targets=all --enable-checking=release --build=i486-linux-
gnu
--host=i486-linux-gnu --target=i486-linux-gnu
Thread model: posix
gcc version 4.2.4 (Ubuntu 4.2.4-1ubuntu4)
```

• Apache2:

```
2.2.8-1ubuntu0.14
```

• MySQL:

```
mysql-client-5.0 5.0.51a-3ubuntu5.5
libmysqlclient15-dev 5.0.51a-3ubuntu5.5
mysql-server-5.0 5.0.51a-3ubuntu5.5
```

• Qt:

```
libqt4-dev 4.3.4-0ubuntu3.1
libqthreads-12 1.6.8-6ubuntu1
QMake 1.07a (Qt 3.3.8b)
```

6.1 HOW TO INSTALL PRoPHET IN UBUNTU

First we install subversion:

```
sudo apt-get install subversion
```

In a home folder we create folder structure with name n4c:

```
mkdir n4c
mkdir n4c/dtn
mkdir n4c/apps
mkdir n4c/apps/dftp
mkdir n4c/logs
mkdir n4c/scripts
```

```
mkdir n4c/dtn/prophet
```

We move ourself to dtn folder and download latest prophet code:

```
cd n4c/dtn/
svn checkout http://grasic.net/prophet
```

We download all the needed qt libs (it should request to install all other required packages as well):

```
sudo apt-get install libqt4-dev
sudo apt-get install build-essential
```

Then edit/configure the prophet.pro file:

```
gedit prophet.pro
```

You can set/clear 4 flags here:

Enable/disable GUI (enable)

Enable/disable PDAGUI (disable,use this only for N810)

Enable/disable DTN_INTERFACE (disable, you will not be running DTN2 beside prophet)

Enable/disable DFTP_INTERFACE (enable, used for transferring files)

Compile prophet using commands:

```
qmake prophet.pro
make
```

Before starting prophet you need to set up the prophet.ini file (most important lines to set):

NODEID=10 (usually we use the last .xxx number of ip)

NODENAME=lime.dtn (name of the machine, we have our own list of machines and ip numbers at LTU, if you are planning to join the test with own machine let me know and I will assigned name and ip for the machine)

NODEIP=192.168.2.10

NODEBROADCAST=....

DFTPPATH=/home/<*user*>/n4c/apps/dftp/

STORAGEPATH=/home/<*user*>/n4c/dtn/prophet/storage/

MSGPATH=/home/<*user*>/n4c/dtn/prophet/list/

LOGPATH=/home/<*user*>/n4c/logs/prophet/

Run prophet using the command:

```
./prophet
```

6.2 COMPILING DTN REFERENCE IMPLEMENTATION

Create a N4C folder structure:

```
/home/<user>/n4c (n4c root directory)
/home/<user>/n4c/apps (applications directory)
/home/<user>/n4c/dtn (dtn platform directory)
/home/<user>/n4c/dtn/db (dtn database directory)
/home/<user>/n4c/logs (log directory)
/home/<user>/n4c/scripts
```

Compiling DTN reference implementation 2.6.0(Ubuntu 8.10)

- Add package tcl-8.4

```
sudo apt-get install tcl8.4
```

- Add older gcc and g++ (version 4.1)

```
sudo apt-get install build-essential
```

- Add package libdb4.6, libdb4.6++ and libdb4.6dev

```
sudo apt-get install libdb4.6 libdb4.6++ libdb4.6-dev
```

- Add package libxerces-c28

```
sudo apt-get install libxercesc28
```

- Quick fix: compiling dtn-2.6.0 release version fails on a fresh install of Ubuntu 8.04. Add `#include <sys/types.h>` in oasys/util/SafeRange.h

Compile oasys-1.3.0:

```
./configure --with-cc=gcc-4.1 --with-xerces-c
make
sudo make install
```

Compile dtn-2.6:

```
./configure --with-cc=gcc-4.1
```

```
make
sudo make install
```

Configure DTN:

Open n4c/dtn/DTN-2.6.0/deamon/dtn.conf

Set dbdir to: "/home/<*user*>/n4c/dtn/dtn-2.6.0/db"

Set route local_eid to: "dtn://nodename.dtn"

Now you can run dtnd for the first time:

```
./dtnd -c dtn.conf -initdb -t
```

Next time use:

```
./dtnd -c dtn.conf
```

6.3 COMPILING AND CONFIGURING DTN MAIL

Gateway:

Install the following packages:

```
sudo apt-get install python python-dev python2.5 python2.5-dev python-pysqlite2
tcl8.5 tcl8.5-dev
```

Create a folder structure:

```
mkdir /home/dtn
mkdir /home/dtn/dtn
mkdir /home/dtn/bundlestore
mkdir /home/dtn/bundlestore/bundles
mkdir /home/dtn/maildir
mkdir /home/dtn/python
```

Install postfix:

```
sudo apt-get install postfix (system mail name: dtn.las.ipn)
```

Configure postifx:

```
--- sudo gedit /etc/postfix/main.cf ---
```

Add to <relay_domains> the following:

```
relay_domains = dtn.las.ipn, dtn.las.ipn.relay
```

Add the following lines:

```
dtnout_destination_recipient_limit = 1
# Transport maps for DTN transports
transports_maps = regexp:/etc/postfix/transport_regexp
```

Add to <mynetworks>:

```
[network IP]/[CIDR notation (if you are in a sub-network)]
--- sudo gedit /etc/postfix/master.cf ---
```

Add the following:

```
# DTN mail outgoing interface
dtnout unix - n n - - pipe -v flags=D user=[hostname]
argv=/home/[hostname]/dtn/python/pypop_smtpout.py $(recipient)
```

Copy the PyMail software folder (gateway version) to a selected directory (/home is used in this example) and change the owner to your system user:

```
sudo cp -R /[PyMail_module]/gateway/dtn/ /home/
sudo chown -R [user] /home/dtn/
```

Configure PyMail:

```
--- gedit /home/[dtn_user]/python/dp_main.py ---
```

Change domain in line 128 to:

```
"dtn.las.ipn"
```

Compiling software:

```
sudo apt-get install python-dev python-pysqlite2
```

```
cd /home/[hostname]/n4c/DTN2-N4C/applib
make pythonapi
sudo make pythonapi_install
```

Configure DTN2:

```
--- gedit /home/[dtn_user]/dtn/dtn.conf ---
```

Change the following lines accordingly:

```
storage set dbdir /home/[dtn_user]/db
storage set payloaddir /home/[dtn_user]/bundlestore/bundles
route set type prophet
```

Change route local_eid to the appropriate one and add the following lines:

```
prophet set age_period 43200
prophet set kappa 86400000
interface add tcp0 tcp
discovery add ip_disc ip port=4301
discovery announce tcp0 ip_disc tcp cl_port=4557 interval=10
```

Configure PyMail:

```
--- gedit /home/[dtn_user]/python/dp_pfmailin.py ---
```

On line 174 change the domain to the system mail name

```
--- gedit /home/[dtn_user]/python/dp_dtn.py ---
```

On line 531 change nomadic domain to the system mail name

```
cd /home/dtn/
```

Start PyMail:

```
sudo ./start_nomadic_mail.sh
```

Stop PyMail:

```
sudo ./stop_nomadic_mail.sh
```

Outstation:

Install the following packages:

```
sudo apt-get install python python-dev python2.5 python2.5-dev python-pysqlite2
tcl8.5 tcl8.5-dev
```

Create a system user and enter it's main folder:

```
cd /home/[user]
```

Create a folder structure:

```
mkdir bundlestoredtn_pymail.log
mkdir maildir
mkdir python
mkdir log
mkdir db
mkdir /dtn/log
```

Install Berkeley Database: (download Berkeley DB 4.7.25 (ZIP version))

```
http://www.oracle.com/technology/software/products/berkeley-db/db/index.html
```

Extract the file, enter the extracted folder and perform the following commands:

```
cd db-4.7.25
cd build_unix
../dist/configure --prefix=/usr/local/berkeleydb --enable-compat185 --enable-cxx
--enable-debug_rop --enable-debug_wop --enable-rpc
```

```
make
sudo make install
sudo su
echo '/usr/local/berkeleydb/lib/' >> /etc/ld.so.conf
echo '/usr/local/lib/' >> /etc/ld.so.conf
exit
sudo ldconfig
```

Compiling software:

```
cd /home/[DTN_user]/n4c/DTN2-N4C/applib
make pythonapi
sudo make pythonapi_install
```

Configure DTN Daemon:

```
--- gedit /home/[user]/dtn/dtn/dtn_outstation_static.conf ---
```

Change the following lines accordingly:

```
storage set payloaddir /home/[user]/bundlestore
storage set dbdir /home/[user]/db
route set type <prophet / static>
```

Change route local_eid to the appropriate one and add the following lines:

```
prophet set age_period 43200
prophet set kappa 86400000
interface add tcp0 tcp
discovery add ip_disc ip port=4301
discovery announce tcp0 ip_disc tcp cl_port=4557 interval=10
```

Configure logging:

```
--- gedit /home/beta/dtn/python/dtn_pymail_log.conf ---
```

On line 31 change the first argument to:

```
"/home/[user]/log/dtn_pymail.log"
```

On line 38 change the first argument to:

```
"/home/[user]/log/dtn_postfix_pipe.log"
```

Configure PyMail:

```
--- gedit /home/[user]/dtn/start_pymail.sh ---
```

Add to line 15:

```
/usr/bin/dtnd -c /home/[user]/dtn/dtn/dtn_outstation_static.conf -d -o
/home/[user]/log/dtnd.log -l info
```

Add to line 22:

```
python /home/[user]/dtn/python/dp_outstation.py &
```

```
--- gedit /home/[user]/dtn/python/dp_outstation.py ---
```

Change line 115 to:

```
mail_domain = [system_mail_domain]
```

Change line 224 to:

```
main("/home/[user]/maildir", "/home/[user]/log",
"/home/[user]/dtn/python/dtn_pymail_log.conf")
```

```
--- gedit /home/[user]/dtn/start_pymail.sh ---
```

Change the last line to:

```
python /home/[user]/dtn/python/dp_outstation.py &
```

```
--- gedit /home/[user]/dtn/stop_pymail.sh ---
```

Change line 4 to:

```
pid=`ps | awk –f /home/[user]/dtn/dp_out.awk`
```

```
cd /home/[user]/dtn
```

Start PyMail:

```
sudo ./start_pymail.sh
```

Stop PyMail:

```
sudo ./stop_pymail.sh
```

6.4 COMPILING AND CONFIGURING WEB CACHING SERVICE

Gateway:

Move the DTN daemon configuration file:

```
mv ~/dtn/dtn.conf ~/dtn/dtn_router_mule_gateway.conf
```

Edit the configuration file:

```
--- gedit ~/dtn/dtn_router_mule_gateway.conf ---
```

Replace the file content with the following:

```
-----------------------------------------------------------------------
log /dtnd info "dtnd parsing configuration..."
console set addr 127.0.0.1
console set port 5050
set shorthostname [lindex [split [info hostname] .] 0]
console set prompt "$shorthostname dtn% "
storage set type berkeleydb
```

```
storage set server_port 62345
storage set schema /etc/DS.xsd

set dbdir ""
foreach dir {/data/dtn/storage} {
if {[file isdirectory $dir]} {
set dbdir $dir
break
}
}

if {$dbdir == ""} {
puts stderr "Must create /data/dtn/storage storage directory"
exit 1
}

storage set payloaddir /home/[user_name]/dtn/bundles
storage set dbname DTN
storage set dbdir /home/[user_name]/dtn/db
route local_eid "dtn://[hostname].dtn"

route set type static
route set add_nexthop_routes true
route set open_discovered_links true
interface add tcp0 tcp local_port=4556
discovery add tcp0d ip port=9556
discovery announce tcp0 tcp0d tcp interval=10
param set link_min_retry_interval 1
param set link_max_retry_interval 10

#################
```

```
##### routes ####
#################
## gateway routes via TCD mule

#define routes
log /dtnd info "dtnd configuration parsing complete"

## emacs settings to use tcl-mode by default
## Local Variables: ***
## mode:tcl ***
## End: ***
```

Compile and run PRoPHET:

```
qmake prophet.pro
make
./prophet (just for testing purposes)
```

Build the database:

```
mysql -u www-data -p
password = www-data
```

Copy the following commands into the mysql terminal:

```
CREATE USER 'www-data'@'localhost' IDENTIFIED BY 'www-data';
CREATE DATABASE members;
GRANT ALL ON members.* TO 'www-data'@'localhost';
CREATE TABLE `members`.`creds` (
`uid` int(8) unsigned NOT NULL auto_increment,
`username` varchar(20) NOT NULL default 'anon',
```

```
`password_md5` varchar(32) NOT NULL default '5f4dcc3b5aa765d61d8327deb882cf99'
COMMENT 'default password = password',
PRIMARY KEY (`uid`),
KEY `new_index` (`uid`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
CREATE TABLE `members`.`groups` (
`gid` int(8) unsigned NOT NULL,
`groupname` varchar(20) character set utf8 NOT NULL,
PRIMARY KEY (`gid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `members`.`requests_tbl` (
`req_ID` varchar(32) collate utf8_unicode_ci NOT NULL,
`uid` int(8) unsigned NOT NULL,
`req_val` varchar(1024) collate utf8_unicode_ci NOT NULL,
`req_response` tinyint(1) NOT NULL,
`req_created` date NOT NULL,
PRIMARY KEY (`req_ID`),
KEY `new_index` (`uid`),
CONSTRAINT `new_fk_constraint10` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`)
ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
CREATE TABLE `members`.`sessions` (
`sid` varchar(32) NOT NULL default '06D5VlLQTbM57LL7IBMW38yHkFpb1XVa',
`sid_dir` varchar(32) NOT NULL default '06D5VlLQTbM57LL7IBMW38yHkFpb1XVa',
`uid` int(8) unsigned NOT NULL,
`signature` varchar(20) NOT NULL,
`timeout_date` datetime NOT NULL,
`expiration_date` datetime NOT NULL,
`enum` int(8) unsigned NOT NULL auto_increment,
PRIMARY KEY (`enum`),
KEY `new_fk_constraint` (`uid`),
```

```
CONSTRAINT `new_fk_constraint` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`) ON
UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC;
CREATE TABLE `members`.`tracking` (
`uid` int(8) unsigned NOT NULL,
`client_ip_address` varchar(15) NOT NULL,
`download_date` datetime NOT NULL,
`download_path` varchar(255) NOT NULL,
`download_size` int(8) unsigned NOT NULL,
KEY `new_fk_constraint3` (`uid`),
CONSTRAINT `new_fk_constraint12` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`)
ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
CREATE TABLE `members`.`uid_gid` (
`uid` int(8) unsigned NOT NULL,
`gid` int(8) unsigned NOT NULL default '0',
`enum` int(8) unsigned NOT NULL auto_increment,
PRIMARY KEY USING BTREE (`enum`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
quit
```

Execute the DTN daemon and the HTMLgateway:

```
cd ~/n4c/DTN2-N4C/daemon/
```

On the first time:

```
./dtnd -c ~/dtn/dtn_router_mule_gateway.conf --init-db
```

After the first time:

```
./dtnd -c ~/dtn/dtn_router_mule_gateway.conf
```

```
cd /
sudo -u www-data ~/n4c/DTN2-N4C/apps/dtnN4Cmiddle/dtnN4Crecv -S
dtn://[hostname].dtn/HTMLgateway
```

Router:

Create folder structure:

```
mkdir ~/dtn
cd dtn
mkdir bundles
mkdir db
mkdir list
mkdir log
cd dtn/
cp ../n4c/DTN2-N4C/daemon/dtn.conf .
```

Configure DTN daemon:

```
--- gedit ~/dtn/dtn.conf ---
```

Replace the file content with the following:

```
-----------------------------------------------------------------------
log /dtnd info "dtnd parsing configuration..."
console set addr 127.0.0.1
console set port 5050
set shorthostname [lindex [split [info hostname] .] 0]
console set prompt "$shorthostname dtn% "
storage set type berkeleydb
```

```
storage set server_port 62345
storage set schema /etc/DS.xsd

set dbdir ""
foreach dir {/data/dtn/storage} {
if {[file isdirectory $dir]} {
set dbdir $dir
break
}
}

if {$dbdir == ""} {
puts stderr "Must create /data/dtn/storage storage directory"
exit 1
}

storage set payloaddir /home/[user_name]/dtn/bundles
storage set dbname DTN
storage set dbdir /home/[user_name]/dtn/db
route local_eid "dtn://[hostname].dtn/HTMLrouter"

route set type static
route set add_nexthop_routes true
route set open_discovered_links true
interface add tcp0 tcp local_port=4556
discovery add tcp0d ip port=9556
discovery announce tcp0 tcp0d tcp interval=10
param set link_min_retry_interval 1
param set link_max_retry_interval 10

log /dtnd info "dtnd configuration parsing complete"
```

```
## emacs settings to use tcl-mode by default
## Local Variables: ***
## mode:tcl ***
## End: ***
```

Compile PRoPHET:

```
qmake prophet.pro
make
./prophet (just for testing purposes)
```

Configure Squid:

```
--- sudo gedit /etc/squid/squid.conf ---
```

Add on line 3049:

```
visible_hostname testmachine

sudo /etc/init.d/squid restart

--- sudo gedit /etc/php5/apache2/php.ini ---
```

Add on line 711:

```
extension=mysql.so
sudo /etc/init.d/apache2 restart

--- gedit ~/n4c/DTN2-N4C/apps/dtnN4Cmiddle/middle_api.c ---
```

Change lines 24, 25 e 26 to:

```
char bndlDest [EID_MAX_LENGTH + 1] = "dtn://[gateway_hostname].dtn/HTMLgateway";
char bndlSrc [EID_MAX_LENGTH + 1] = "dtn://[router_hostname].dtn/HTMLrouter";
```

```
char bndlRply_to [EID_MAX_LENGTH + 1] =
"dtn://[router_hostname].dtn/HTMLrouter";


cd ~/n4c/HTMLrequester/mod_auth_form-2.05/
```

Export the following environment variables:

```
export APACHE2_INCLUDE=/usr/include/apache2

export APR1_INCLUDE=/usr/include/apr-1.0

./configure -C
```

Configure mod_auth_form:

```
--- gedit ~/n4c/HTMLrequester/mod_auth_form-2.05/src/Makefile.am ---
```

add to the end of the file the output of the command: apr-1-config --cppflags, preceded by «CPPFLAGS = »

```
CPPFLAGS = -DLINUX=2 -D_REENTRANT -D_GNU_SOURCE -D_LARGEFILE64_SOURCE

sudo make install

libtool --finish /usr/local/lib

sudo su

echo "LoadModule auth_form_module /usr/lib/apache2/modules/mod_auth_form.so" >>
/etc/apache2/mods-available/auth_form.load

exit


cd /etc/apache2/mods-enabled
```

Create the following symbolic links:

```
sudo ln -s ../mods-available/auth_form.load auth_form.load

sudo ln -s ../mods-available/rewrite.load rewrite.load
```

Build the database:

```
mysql -u www-data -p

password = www-data
```

Copy the following commands into the mysql terminal:

```
-------------------------------------------------------------------------
CREATE USER 'www-data'@'localhost' IDENTIFIED BY 'www-data';
CREATE DATABASE members;
GRANT ALL ON members.* TO 'www-data'@'localhost';
CREATE TABLE `members`.`creds` (
`uid` int(8) unsigned NOT NULL auto_increment,
`username` varchar(20) NOT NULL default 'anon',
`password_md5` varchar(32) NOT NULL default '5f4dcc3b5aa765d61d8327deb882cf99'
COMMENT 'default password = password',
PRIMARY KEY (`uid`),
KEY `new_index` (`uid`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
CREATE TABLE `members`.`groups` (
`gid` int(8) unsigned NOT NULL,
`groupname` varchar(20) character set utf8 NOT NULL,
PRIMARY KEY (`gid`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
CREATE TABLE `members`.`requests_tbl` (
`req_ID` varchar(32) collate utf8_unicode_ci NOT NULL,
`uid` int(8) unsigned NOT NULL,
`req_val` varchar(1024) collate utf8_unicode_ci NOT NULL,
`req_response` tinyint(1) NOT NULL,
`req_created` date NOT NULL,
PRIMARY KEY (`req_ID`),
KEY `new_index` (`uid`),
CONSTRAINT `new_fk_constraint10` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`)
ON
UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci;
```

```
CREATE TABLE `members`.`sessions` (
`sid` varchar(32) NOT NULL default '06D5VlLQTbM57LL7IBMW38yHkFpb1XVa',
`sid_dir` varchar(32) NOT NULL default '06D5VlLQTbM57LL7IBMW38yHkFpb1XVa',
`uid` int(8) unsigned NOT NULL,
`signature` varchar(20) NOT NULL,
`timeout_date` datetime NOT NULL,
`expiration_date` datetime NOT NULL,
`enum` int(8) unsigned NOT NULL auto_increment,
PRIMARY KEY (`enum`),
KEY `new_fk_constraint` (`uid`),
CONSTRAINT `new_fk_constraint` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`) ON
UPDATE CASCADE
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8 ROW_FORMAT=DYNAMIC;
CREATE TABLE `members`.`tracking` (
`uid` int(8) unsigned NOT NULL,
`client_ip_address` varchar(15) NOT NULL,
`download_date` datetime NOT NULL,
`download_path` varchar(255) NOT NULL,
`download_size` int(8) unsigned NOT NULL,
KEY `new_fk_constraint3` (`uid`),
CONSTRAINT `new_fk_constraint12` FOREIGN KEY (`uid`) REFERENCES `creds` (`uid`)
ON
DELETE CASCADE ON UPDATE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=utf8 ROW_FORMAT=FIXED;
CREATE TABLE `members`.`uid_gid` (
`uid` int(8) unsigned NOT NULL,
`gid` int(8) unsigned NOT NULL default '0',
`enum` int(8) unsigned NOT NULL auto_increment,
PRIMARY KEY USING BTREE (`enum`)
) ENGINE=InnoDB AUTO_INCREMENT=1 DEFAULT CHARSET=utf8;
quit
```

Create folder structure:

```
sudo mkdir /data
sudo mkdir /data/www/
sudo mkdir /data/www/private/
sudo mkdir /data/www/router_stage/
sudo cp -R ~/n4c/HTMLrequester/ /data/www/private/
cd /data/www/private/HTMLrequester/
sudo mkdir router_stage
sudo mkdir staging
sudo chown -R www-data:www-data /data/www/
```

```
--- sudo gedit index.php ---
```

On lines 96 and 108, change 'details.php' to '/details.php'

```
--- sudo gedit add_search.php ---
```

On line 105, change 'details.php' to '/details.php'

```
--- sudo gedit details.php ---
```

On line 336, remove «'http://localhost'.» and change '/router_stage/' to 'router_stage/'

```
cd ~/n4c/DTN2-N4C/daemon/
```

On the first time:

```
./dtnd -c ~/dtn/dtn_router_mula_gateway.conf --init-db
```

After the first time:

```
./dtnd -c ~/dtn/dtn_router_mula_gateway.conf
```

```
cd ~/n4c/DTN2-N4C/apps/dtnN4Cmiddle/
./dtnN4Cmiddleware


cd /
sudo ~/n4c/DTN2-N4C/apps/dtnN4Cmiddle/dtnN4Crecv -p transaction_####.tar.gz
dtn://jmachado-laptop.dtn/HTMLrouter


firefox
http://localhost/private/HTMLrequester
```

6.5 CONFIGURING HIKER'S APPLICATION

The information required to setup the Hiker's Application is available in the following references [9, 10, 11, 12, 13]. The reference [9] reports a complete tutorial, step by step, of how to install Nokia tablets (with the latest Maemo version and with no priori configuration) with all the necessary applications. On reference [10] it is possible to access to other technical information as installing Hiker's Application on EEE PC. The references [11, 12, 13] are technical configurations and scripts to run on Linux based devices, as the Nokia tablets and the Asus EEEPC, to install the required applications automatically.

7. STAKEHOLDERS

A use case is discussed in this section. We want to demonstrate, through a diagram and a UML diagram, the levels of user interaction with the platform. The use case in Figure 16 describes the functionality of the system in a horizontal way. It shows a top-down perspective of the features that were discussed thought this document and presents an overview of the usage requirements for the integration platform.

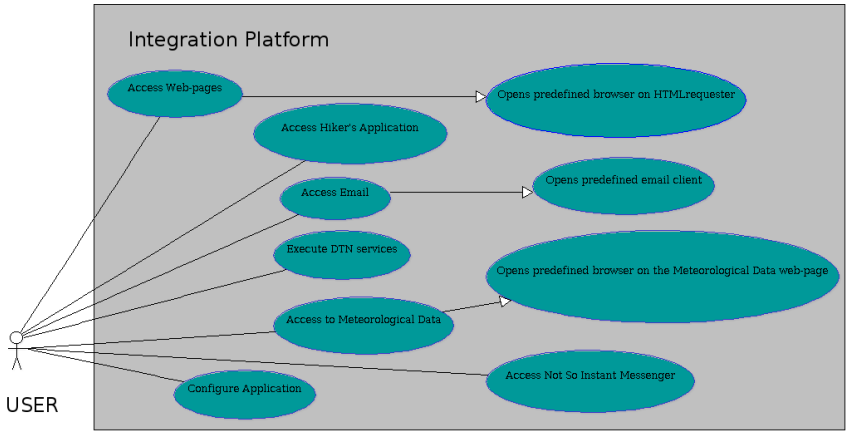

Figure 16 - Use Case diagram representing the interaction of a user with the Integration Platform

It is important to understand how the work done by N4C project could have a positive impact on certain scenarios and for some target zones. This understanding increases the importance of the project and permits to work oriented to specific purposes, to solve explicit problems and enriching the final deliverables. The Figure 17 depicts a practical applicability where DTN networks could be successfully implemented, improving the quality of life of the people enabling new ways of communication and inter-connection.

On Figure 17 are represented several cities or villages with a DTN infrastructure implemented and another one with connection to the Legacy Internet. Those cities are linked between them by roads or rivers, and using these routes of travelling the "mules", which can be cars, bus or boats, can take the information from one place to another. This means, pending requests from DTN zones, can be answered from the Legacy Internet using such "mules" to carry the data.

The Figure 17 does not represent specifically any country or city, it is just for demonstrating a possible applicability where the DTN technology and the integration platform could be used. Indeed, countries as Brasil and Africa are good examples of real scenarios where DTN can have a large and positive impact.

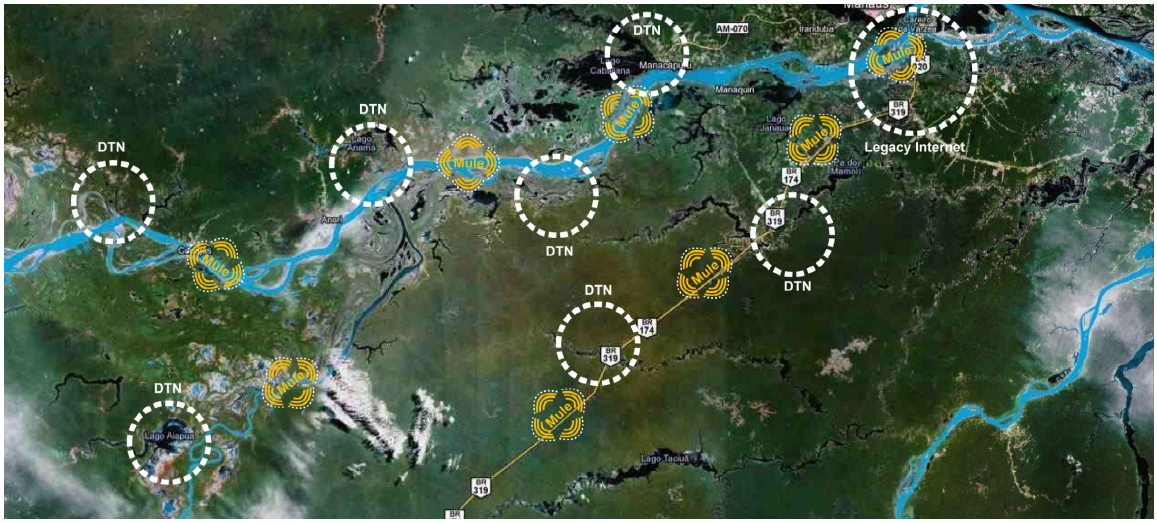

Figure 17 - Practical Case for DTN applications