Validation of a Simulation Model of the Delay Tolerant Network Architecture

Martin Edsfors
Isidro Nistal

Luleå University of Technology
MSc Programmes in Engineering
Department of Computer Science and Electrical Engineering
Division of Computer Communication
Validation of a simulation model of the Delay Tolerant Network Architecture

Martin Edsfors
Isidro Nistal Garrido
Abstract

The purpose of this thesis was to study and compare a simulation model and a reference model of the Delay Tolerant Network architecture. The simulation model was done during a previous thesis at Luleå university of technology and the reference model was created as a project by the DTN research group. Both models were incomplete and had to be modified to serve the purpose of this thesis. Several tests were then run on both models with as equal topology as possible.

The conclusion of these tests was that the models behave quite similar and that the simulator is a good choice for testing DTN.
Acknowledgments

We would like to thank our supervisor Kaustubh Phanse for all the help and guidance he has given us during this project.
Contents

1 Introduction .................................................. 5
  1.1 Problem .................................................. 5
  1.2 Purpose .................................................. 5
  1.3 Demarcations ............................................. 5

2 Background .................................................. 6
  2.1 DTN Concepts ............................................ 7
    2.1.1 Bundling protocol ................................... 7
    2.1.2 Services ............................................. 7
    2.1.3 Custody transfer ................................... 8
    2.1.4 Nodes ............................................... 9
    2.1.5 Regions ............................................. 9
    2.1.6 Routing ............................................. 10
    2.1.7 Contact types ..................................... 10
    2.1.8 Security ........................................... 10
    2.1.9 Headers ............................................ 11
    2.1.10 Fragmentation ..................................... 11

3 Methodology ................................................ 12
  3.1 Simulation model ....................................... 12
    3.1.1 NS-2 ............................................... 12
    3.1.2 NAM ............................................... 12
    3.1.3 Simulation model ................................... 13
    3.1.4 Modifications ..................................... 13
    3.1.5 Routing ............................................ 14
  3.2 Reference model ......................................... 14
  3.3 Setting up the test bed ................................ 14
    3.3.1 Bundle daemons .................................... 14
    3.3.2 Dynamic switch .................................... 15
    3.3.3 Equipment ......................................... 16
  3.4 Data Acquisition ........................................ 16
    3.4.1 Tests .............................................. 16
    3.4.2 Constraints ....................................... 16
    3.4.3 Measures .......................................... 17
4 Results

4.1 End-to-end delay
   4.1.1 No expiration time
   4.1.2 20 seconds expiration time
   4.1.3 5 seconds expiration time

4.2 Delivery ratio
   4.2.1 No expiration time
   4.2.2 20 seconds expiration time
   4.2.3 5 seconds expiration time

4.3 Buffer occupancy
   4.3.1 No expiration time
   4.3.2 20 seconds expiration time
   4.3.3 5 seconds expiration time
Información

Titulo del proyecto: Validación de un modelo de simulación de la arquitectura tolerante a retrasos.
Autores del proyecto: Isidro Nistal Garrido, Martin Edsfors
Tutor del proyecto: Kastubh Phanse
Institución: Luleå Tekniska Universitet
Coordinador Erasmus: Christina Hamsch
Fecha de Lectura: 7 de Noviembre de 2005
Lugar de Lectura: Luleå Tekniska Universitet

Introducción

En las redes actuales, es cada día más habitual observar topologías donde los protocolos de red más extendidos en Internet, TCP y UDP, no son válidos. Este tipo de redes está definido por la aparición de alguna de las siguientes características:
- largos y variables retrasos de los enlaces
- contactos intermitentes
- elevadas tasas de error
- tasas de transmisión altamente asimétricas

En este tipo de entornos los protocolos conversacionales no funcionan como es debido y aparecen protocolos de transporte aptos para estas topologías. A fin de conectar redes que utilizan diferentes protocolos de transporte, la arquitectura de red DTN (Delay Tolerant Networkk) ha sido definida en [1]. Este documento define un protocolo llamado Bundling Protocol que se sitúa en la capa mas alta de la pila de protocolos y que presenta diferentes particularidades.

1Red tolerante a retrasos
2(Bundle puede ser traducido como fajo o fardo, y hace referencia a los paquetes que se envían desde el protocolo definido)
Objetivo del proyecto


Introducción teórica

La arquitectura DTN tiene como principal objetivo dar cobertura a redes donde los tradicionales protocolos de Internet (UDP, TCP) no pueden. Aunque en un principio esta arquitectura fue aplicada en redes interplanetarias, las aplicaciones de la misma pueden tener amplios rangos, desde aplicaciones militares donde las tropas pasan solamente en determinados momentos por zonas con cobertura, hasta todo tipo de redes inalámbricas con conectividad limitada.

Bundling protocol

Es el protocolo que define la arquitectura, se sitúa en la pila de protocolos, debajo del nivel de aplicación y sirve como puente entre diferentes protocolos de transporte de diferentes topologías de red.

La característica principal de dicho protocol se basa en lo que se llama “store and forward” 3 que permite a cada nodo de la red almacenar los bundles en almacenamiento permanente. Este tipo de transmisión llamado en el draft [1] transferencia custodiada (custody transfer), tiene como principal ventaja el hecho de que, dada una pérdida de un bundle en alguno de los links entre el nodo fuente y destino, la retransmisión se hace solo desde el nodo intermedio que tiene la custodia del bundle perdido. De esta forma se evita que las retransmisiones crezcan en forma exponencial con el número de links con baja conectividad. La fiabilidad del protocolo esta unida directamente a esta característica, sin esta transferencia custodiada activada, la fiabilidad de la transmisión recae en el protocolo de transporte.

Aunque la mencionada característica es la más importante, éste también incluye otras particularidades como:

- Recibo End-to-End4: cuando el bundle llega al nodo destino, este manda un recibo a la fuente de dicho bundle.

- Notificación de transferencia custodiada: cada vez que un bundle es tomado en custodia por otro nodo, dicho nodo responde con un mensaje a la fuente del bundle.

- Notificación de envió: cada vez que un bundle es enviado hacia el destino por un nodo, éste envía una señal a la fuente de dicho bundle.

3 Traducido directamente como almacena y envía
4 Traducido como fin a fin, hace referencia al camino entre la fuente y el destino.
Cualquier combinación de estas características puede ser activada en cada bundle.

Otra de las peculiaridades del protocolo consiste en la autenticación de los nodos, lo que le hace un protocolo muy seguro, cada nodo tiene que autenticarse con encriptación asimétrica para poder mandar cualquier tipo de datos. Esta característica no ha sido tomada en cuenta en esta tesis, por no estar implementada todavía en los modelos en cuestión y considerarse de poca aplicación para el objetivo de la misma.

**Enrutamiento**

Para tal función el protocolo estudiado utiliza un enrutamiento similar al utilizado en las redes IP. Los nodos se identifican por un nombre de región y otro de identidad de nodo (ver figura 2.6). De esta forma cada bundle si no está dirigido a un lugar dentro de la región es enviado a la puerta de enlace de dicha región, esta pertenece a la vez a dos regiones. Desde ahí se va pasando por diferentes regiones hasta llegar a la región destino donde el bundle es enviado a su nodo final.

**Metodología**

Para la realización de esta tesis se utilizaron 5 PC. En uno de ellos se hicieron las simulaciones con el simulador de red NS-2 [10]. Y con los otros 4 se creo una red de 3 nodos interconectados entre sí mediante el cuarto PC. Este PC tenía instalado un programa llamado Dynamic Switch [12] que permite emular desconexiones controladas entre los nodos de la red.

**Mediciones**

- Retardo: el tiempo que un bundle dura en llegar de un extremo a otro de la red.
- Tasa de entrega de bundles: el porcentaje de bundles que llegan a su destino
- Ocupación en disco: máximo valor de bundles guardados en disco en cada uno de los nodos de la red

**Variables**

- Disponibilidad del enlace: porcentaje de tiempo que el enlace esta activo
- Tiempo de reenvío: tiempo que transcurre hasta que se retransmite un bundle que tiene transferencia custodiada activada y ha sido perdido en el enlace. A pesar de que fue fijo en todas las simulaciones (2 segundos), este valor podría ser cambiado, no obstante para la topología descrita se eligió el valor que se vio que era mas correcto para la tesis, ya que eliminaba
las diferencias en tiempos de ejecución de la simulación y el modelo de referencia.

- Tiempo de expiración: máximo tiempo que un bundle puede pasar en la red antes de ser descartado. Tres valores fueron probados, sin tiempo de expiración, 20 segundos (10 retransmisiones) y 5 segundos (2 retransmisiones).

Pruebas

Se realizaron 3 pruebas con cada valor de variable. Las pruebas consistieron en durante un periodo de 100 segundos a una tasa de 20 bundles/segundo y 100 bytes de carga en cada bundle, enviar datos desde un nodo fuente hacia un nodo destino a través de un nodo intermedio, obteniéndose 6000 valores para cada medición, lo que ofrece un buen resultado estadísticamente hablando.

Conclusiones

Tras las pruebas realizadas sobre ambos modelos se puede afirmar que se comportan de forma muy similar, lo suficiente como para poder hacer futuros experimentos sobre la simulación. La simulación supone una opción acertada dada la juventud del programa de referencia, en el que la mayoría de las características del protocolo no están implementadas. Así mismo, para investigaciones futuras, es mucho más fácil el simular grandes redes con un simulador que montar complicadas topologías de red físicamente.

Futuro trabajo

- Adecuar la simulación al nuevo draft [3]
- Implementar en la simulación algún tipo de enrutamiento dinámico que evite el tener que escribir la tabla de enrutamiento completa en cada nodo
- Implementar las características de seguridad en la simulación

Limitaciones

Durante el transcurso de la elaboración de esta tesis, un nuevo draft de internet ha sido publicado sobre la arquitectura DTN [2]. Puesto que tanto la simulación como el modelo de referencia están basados en el draft anterior [1], y los cambios sobre la arquitectura no afectan a la estructura base, se consideró inutil trabajar con el draft mas moderno.
Chapter 1

Introduction

1.1 Problem

There exists some areas in networking today where traditional network protocols have trouble coping. It could be due to intermittent connectivity, long or variable delay, asymmetric data rates or high error rates. Therefore a new architecture called DTN has been suggested. This architecture was analyzed in another thesis made at Luleå University of Technology [7]. That thesis also produced a simulation model of DTN which is one of the two models used as a basis for this thesis. The other one being a reference model made for Linux available on the DTN Research Groups home page[8]

1.2 Purpose

The purpose of this thesis was to study the performance of the DTN architecture under different DTN scenarios for cross-validating the simulation model of the protocol with the DTN reference implementation. The differences and bugs in the implementations were documented and modifications to the simulation and the reference model were done to conform to the DTN architecture specification.

1.3 Demarcations

The security bit in the DTN architecture was not well defined at the start of this thesis. This coupled with the fact that neither of the models had this part implemented was the reason for leaving it out of the thesis.

Any new versions of the drafts used for this thesis will not be used for this work, since both models used are based on a certain draft it would be a time consuming effort to rewrite them to comply to newer versions.
Chapter 2

Background

In the Internet today, the TCP/IP suite is most commonly used. While this suite is good, it is not sufficient for certain unconventional networking scenarios. The Internet is expanding into these scenarios where constant connectivity can not be assured and/or the delay is extremely high, for example wireless networks. (When a wireless device loses connection TCP will drop any packets currently in transfer and need to re-send them from the originating node while DTN would only need to re-send from the last node to receive the data.)

Other areas that DTN can be applied to are, satellite networks, sensor networks or interplanetary networks. When NASA were communicating with their Mars rover they had two problems: the latency was quite high, and the link was severed when Mars rotated so that the rover ended up on the far side of the planet. These kinds of environments are not very suitable for existing protocols. With these problems in mind the DTN Research Group (DTNRG)[8] was created in 2002. They used the work from the Interplanetary Internet Research Group as a base to work from. DTNRG are sponsored by the Internet Research Task Force (IRTF). They have drafted a Delay Tolerant Network architecture which uses the Bundling protocol which is the basis of this thesis. The main difference between this architecture and the Internet architecture is that the DTN architecture works on a hop-to-hop basis rather than on an end-to-end basis, which is made possible because of the bundling protocol. The existing Internet protocols also need an open end-to-end path between the source and destination during the whole time communication is made. A DTN can be more similar to the postal system. It generally has non-interactive traffic and no strong guarantee of timely delivery. A postal office does not really need to know where the letter is sent from, only where to deliver it.
2.1 DTN Concepts

2.1.1 Bundling protocol

When using the bundling protocol a bundle layer is added to the protocol stack. This layer exists between the application and transport layer and packs all data that passes through into bundles (see figure 2.1). Afterwards, these bundles are sent using the underlying transport protocol. However, it is only sent to the next node in the route. This node receives the packet and gives the bundle to the bundling layer which determines if it should send the bundle to the next node or if it should be delivered to the application layer. This makes it possible to send the bundle through several networks using different protocols, as long there are gateway nodes in between the different networks. However, sending it like this does not ensure end-to-end delivery. Therefore there are a number of services in the proposed architecture which can be used to do this [1].

![Protocol stack without and with the extra bundle layer](image)

Figure 2.1: The protocol stack without and with the extra bundle layer

2.1.2 Services

The DTN architecture provides several types of services that can be activated individually. To ensure guaranteed end-to-end reliability both custody transfer and at least one kind of receipt must be active.

- *Custody Transfer:* This service is thoroughly described in section 2.1.3

![Custody Transfer](image)

Figure 2.2: Custody Transfer
• **Return Receipt:** When this service is active a message will be sent back to the source when the bundle is received to confirm that it has been delivered.

![Figure 2.3: Return Receipt](image)

• **Custody-transfer Notification:** This sends a message back to the source node when a new node takes custody of the specific bundle.

![Figure 2.4: Custody Transfer Notification](image)

• **Bundle-forwarding Notification:** This sends a notification back to the sender every time the bundle is forwarded.

![Figure 2.5: Bundle-Forwarding Notification](image)

• **Priority of Delivery:** A bundle can have three different types of priority: bulk, normal and expedited, where expedited is the highest.

• **Authentication:** With this enabled the sender's identity and the integrity of the bundle will be verified.

### 2.1.3 Custody transfer

When custody transfer is active one node will always be the custodian of the bundle. This means that that node has a stored copy of the bundle and will
re-send it if needed. When a node receives a bundle and custody transfer is active, it will send a message to that bundle's current custodian saying that it has taken custody. This means that as the bundle progresses in the network, the custodian is in a way moving with it. This way, if the bundle is lost or if it times out, it does not have to be resent from the source node but rather from the custodian. This is a key concept in the DTN architecture, since retransmissions does not have to traverse the entire distance of the link. This is very useful when trying to send data over intermittent contacts. When the data has successfully traversed this contact it does not have to do it again, even if it needs to be resent.

2.1.4 Nodes

DTN nodes work as senders and receivers of bundles. An application uses DTN nodes to set Endpoint IDs, which consists of two parts: a region naming part and an administrative naming part. Every node is uniquely identified by its Endpoint ID. The administrative part is unique for every node within a region. There might however, exist two nodes with different region parts that have the same administrative part.

2.1.5 Regions

DTN divides larger networks into regions. A node belongs to a certain region depending protocol, connection dynamics, administrative reasons or for other reasons. Each region has their own unique name, which is recorded in the region naming part of the nodes Endpoint ID. Some nodes may belong to more than one region and can therefore work like a gateway between regions. These nodes have a Endpoint ID for each region they belong to.

![Figure 2.6: Endpoint ID and regions](image)

Figure 2.6: Endpoint ID and regions
2.1.6 **Routing**

A bundle is routed in the network using the Endpoint ID of the receiving node. DTN utilizes late binding when doing this, which means that the administrative part of the Endpoint ID is not interpreted until the bundle has arrived in the correct region.

2.1.7 **Contact types**

In the DTN architecture several types of contacts are defined:

**Persistent Contacts**

A persistent contact is always available. DSL lines are good examples of this.

**On-Demand Contacts**

On-demand contacts require some procedure to be done to open a connection, but then it works precisely as a persistent contact until disconnected. A good example of this would be a dial-up modem.

**Intermittent - Scheduled Contacts**

Scheduled contacts are when you know that a connection will be established a certain time, for a certain duration. A link with an orbiting satellite or a rover on Mars exemplifies this.

**Intermittent - Opportunistic Contacts**

Opportunistic contacts are non-scheduled contacts that present themselves unexpectedly. This could for example happen when a PDA with WLAN enters the range of a wireless hot spot.

**Intermittent - Predicted Contacts**

Predicted contacts are those that, with a large enough confidence, can have contact time statistically calculated. An example would be walking a certain way every day, connecting to different opportunistic contacts. The knowledge of this could be used to predict when each of these hot spots should be used.

2.1.8 **Security**

The security in DTN is controlled by use of asymmetric cryptography. Each bundle will, if the security features are enabled, include a signature containing a verifiable identity of the sender. Each node in the network has a key-pair used for authentication. Each node the bundle passes will authenticate the previous nodes identity which will stop false messages quickly, freeing bandwidth for real messages.
2.1.9 Headers

DTN utilizes chained headers in the same manner as IPv6. Each bundle must have at least a primary bundle header, a dictionary header and a payload header. The primary bundle header must be first followed by the dictionary header. Depending on the type of bundle other headers can be added after these. The bundle payload header must always be the last one. There are separate headers for bundle authentication, payload security and bundle fragment. All these can be chained in no specific order. The bundle payload header either contains the data transported by the bundle or an administrative payload which is used for sending bundle status reports and custodial signals.

2.1.10 Fragmentation

At some point a bundle might have to be divided into smaller parts. It might for example be that the next hop is through an intermittent contact or that the network has a low MTU. When this is the case, the bundle will be divided into several fragments which will be reassembled again at the final destination. These fragments all have the same primary and dictionary header but will also have a fragmentation header that works as a kind of blueprint, showing how they fit together.
Chapter 3

Methodology

3.1 Simulation model

3.1.1 NS-2

NS-2 started out as a variant of the REAL Network Simulator in 1989, but has grown into a program in its own right. It is written in C++ and has support for a number of different network protocols and routing in both wired and wireless networks. It utilizes a TCL script interface, which makes it possible to write the networks scenarios you want to simulate in pure TCL. This scripting language is easy to learn and makes the use of NS-2 much easier. It does however depend on a number of externally available components which can make it a bit sensitive. However, this is also a strength, since different components can be changed independent of each other. The design is very modular it is rather easy to add newly developed protocols to the simulator without the need to reinstall the whole package. One of the benefits of using NS-2 instead of a real network is that you can control every aspect of the simulation. The environment is also easy to setup and does not require a large number of computers. The main drawback is that you get a very idealized environment which is limited by the assumptions made and the correctness of the simulation models [10].

3.1.2 NAM

NAM or Network AniMator is a program which can be used to visualize and animate packet trace data. This data can come from a simulator like NS-2 or from real network measurements by using tcpdump. It displays this network and the flow of packets throughout this network graphically, making it much easier to interpret the results than reading long log files. The packets and nodes can be color coded to further help with the interpretation and the speed of the simulation can be adjusted dynamically during its run [11].
3.1.3 Simulation model

The simulation model[7] is written in C++ and is modeled on the draft released in September 2004[3]. It includes all aspects of DTN except the security part. It was written as a module for NS-2 and like all protocols in NS-2 it can be accessed through the TCL script interface. It was written by Henrik Eriksson and Patrik Jönsson who designed it as part of their master’s thesis [7].

3.1.4 Modifications

When the simulation model was thoroughly tested it was evident that there were some problems with it and some features were missing, e.g. a number of reason codes and status flags for the administrative payload that were unimplemented. These were added during this thesis to make the model able to cope with more advanced test scenarios and to conform more closely to the draft.

To make the "no known route" reason code work, the bundles needed to have some info about where they had been before, otherwise they would be sent back to the node who originated the error. The solution to this was to add a new object in the bundle which keeps track of previously visited nodes, so that it would conform to the draft [1].

A problem with the resend timer made the underlying UDP protocol behave badly. When a link went down it kept all bundles scheduled for sending until the link was back up again. A true UDP protocol does not always have knowledge of link status and therefore this was rectified, so only bundles with custody feature
activated were scheduled to be resend.

All these problems were fixed both for normal bundles and fragmented ones.

### 3.1.5 Routing

The simulation model uses static routing where the user has to populate the routing tables manually. This makes it very cumbersome to create larger networks, however adding functions like dynamic routing was not in the scope of this thesis. The routing table is created in two steps. First each node is added to a region and then links are created between the nodes. In each link the metric (cost), the MTU and whether the next node is a custodian or not can be adjusted.

### 3.2 Reference model

The reference model, DTN2, is an open source project which was created to provide a good experimentation platform. It can be found on the DTNRG home page. Basically it runs as a daemon in the background for which, you can write applications based on its API to send and receive traffic from one daemon to next one [9].

**Code**

The DTN2 code is mainly written in C++. It uses TCL scripting language[17] to initialize and interpretate the different commands in the daemon. It also uses a database system to store the bundles and features as threads and timers are taken from oasys library [18].

As it is described in the draft [3], the bundling protocol operates on top of transport protocol which could be TCP or a filesystem (a USB memory you move between computers). For this duty the reference implementation has some convergence layers implemented in C++ as TCP and UDP.

### 3.3 Setting up the test bed

#### 3.3.1 Bundle daemons

The reference code did not include any tracing facility. For the purpose of the thesis, tracing messages were added to the code and registered in files that further on, would be used for the analysis.

The reference code is in an early stage so many features of the bundling protocol are not implemented yet. In order to compare the implementations, custody transfer and bundle expiration was implemented in the reference model. Two timers were implemented in the Bundle class, one for resending and one for expiration.
Figure 3.2: Test Bed

All nodes in the experimental test bed were time synchronized using Network Time Protocol (NTP)[13]. The used server was inside the university network(http://bart.sm.luth.se) so that the difference between the clocks were insignificant. The NTP messages were sent over dedicated links, isolated from the experimental traffic (bundles). With these premises, expiration time was applied correctly in any of the nodes, and end-to-end delays could be measured precisely with the trace file from the end node.

3.3.2 Dynamic switch

The dynamic switch is a program that can make a computer switch packets between different network interfaces. It can dynamically turn specified interfaces on and off and it can also set the weights of the links. This gives a much more controlled way of emulating links going up and down than connecting and disconnecting cables in a normal switch, but it has its limitations too, the node that is sending data can not check for link availability just by listening in the wire since this wire is always up and the packets are lost in dynamic switch machine kernel.

The major problem with dynamic switch was actually installing it. It requires a specific Linux kernel which can find in Red Hat Linux 9. To install it the kernel had to be recompiled to include the dynamic switch. This was a source of much trouble in the beginning of this thesis.

The simulation uses a feature of NS-2 that puts links up and down according to an exponential distribution. To achieve this kind of architecture a timed script was created to control the dynamic switch similarly to the simulation. (See appendix C for further details)
3.3.3 Equipment

Four standard PC’s with Red Hat Linux 9 were used in the test bed. One had a multi port network card and dynamic switch installed to connect the remaining computers into a three nodes test network architecture for the reference model testing.

3.4 Data Acquisition

3.4.1 Tests

All tests generated log files for every run made, the simulation used only one and the reference created one for each node. Each testrun sends data during 100 seconds, at a rate of 20 bundles per second and 100 bytes payload per bundle. All tests were run three times, resulting in 6000 different values for each link availability, so a good statistical result was obtained.

Links

In order to make the simulation and reference test bed similar, the delay between nodes was measured in the three nodes network so that the NS-2 links could be tuned to similar values. The values used were 100Mb/s bandwidth and 0.1ms delay.

Parsing

Both the simulation model and reference model generated a log file every run, only one in simulation case, and three, one for each node, in reference case. To process this data, Java programs were used to plot different graphs with the help of gnuplot[14].

3.4.2 Constraints

Draft

A couple of months after the work on this thesis had begun, a new draft of the DTN architecture was released[4]. However, since both the simulation and reference model were constructed according to the old draft, the new one was not used for this thesis.

There are several key differences in the two versions of the draft. First and foremost the new draft is much more mature. It describes some parts more completely. The security part has also been moved into its own document and been more properly described[5]. Following the new draft there are only two headers a bundle is required to have: the primary bundle header and the payload header. The dictionary header referred to in the old draft is now incorporated into the primary bundle header. Many of the length and offset fields in different headers are now to be encoded with the SDNV (Self-Delimiting Numeric Values) encoding scheme. This saves memory and bandwidth since the influenced fields size is dynamically adjusted after the size of the content. There has also been some changes to the administrative payloads and the fragmentation scheme.
Simulation and reference code  The reference model is still in an early stage of development and therefore it has not all features implemented yet and it has many bugs that are fixed every day in the CVS version[9]. A new version of the DTN code was about to be published when this thesis was finished. It would include many new features not available in the version used for this thesis. Some of the features not implemented yet, are custody transfer (section2.1.3), all report types except for end to end receipt, bundle expiration and security.

Although both models had fragmentation implemented in them, we decided not work on that part, since it would have made the implementation of custody transfer in reference model much more complicated and time consuming.

3.4.3 Measures

End-to-end delay
The end-to-end delay is the time spent from a bundle arrives at the bundling daemon from the application, until it is delivered to the destination daemon.

Delivery ratio
The delivery ratio is the percentage of the bundles that are actually delivered.

Maximum buffer size
The buffer size is a measure of the maximum space that the bundles occupy in the memory for a given node.

3.4.4 Variables

Link availability(%)  
Link availability is the time a link is up and running in percentage.

The links are connected and disconnected using an exponential distribution given the following variables:

\[
Uptime = \frac{LinkAvailability}{100} \text{ Seconds}
\]

\[
DownTIme = 1 - Uptime \text{ Seconds}
\]

Link availability will be in all graphs at the x axis.

Expiration time
The time a bundle can wait to be delivered before being discarded. The test have been done for 5, 20 and 2000 seconds.

Resend timer
The resend timer was set to 2 seconds in all tests. This value was selected with respect to the topology used in the tests. If other types of topologies are to be tested, this variable could be changed to reflect that.
Chapter 4

Results

4.1 End-to-end delay

With custody transfer enabled the end-to-end delay increased as the link availability decreased due to the need for more bundles to be resent. Since the resend timer is set at 2 seconds the end-to-end delay increased quite rapidly between 100% and 95% link availability. This is because of link delays being measured in milliseconds, making the resend timer a very large increase.

When the expiration time is lowered the end-to-end delay is decreased, as shown in the figures. The more packets that times out, the faster the ones who does get through goes, lowering the end-to-end delay. The bundles does at most 10 retransmissions for 20 seconds of expiration time and 2 for 5 seconds expiration time.

The big increase in end-to-end delay between 20% and 10% in Figure 4.1 corresponds to a large increase in buffer occupancy. This buffer occupancy increase is evident in both models (see Figure 4.13 and Figure 4.14).

With no custody both models will behave similar to the UDP protocol. As the figures indicate the end-to-end delay is constant independent of the link availability. This is because UDP does not store packets while waiting for a link to go up. This means that the only bundles that made it through were those that were sent when both links in the test network were up. The reference model did not give exactly constant values. Instead they varied between 0.01 seconds and 0.1 seconds. This small variation can be credited to processing times in computers, which are not always the same, as predicted.
4.1.1 No expiration time

Figure 4.1: End-to-end delay with no bundle expiration for the simulation model

Figure 4.2: End-to-end delay with no bundle expiration for the reference model
4.1.2 20 seconds expiration time

Figure 4.3: End-to-end delay with 20 seconds expiration time for the simulation model

Figure 4.4: End-to-end delay with 20 seconds expiration timer for the reference model
4.1.3 5 seconds expiration time

Figure 4.5: End-to-end delay with 5 seconds expiration time for the simulation model

Figure 4.6: End-to-end delay with 5 seconds expiration timer for the reference model
4.2 Delivery ratio

As Figure 4.7 and 4.8 shows, when there is no expiration time all bundles are delivered with custody activated. However, there is no guarantee how long the transfer will take. When the expiration timer is set to a low value the delivery ratio with custody is no longer constantly 100%, since bundles are starting to expire. The bundles have a limited number of retransmissions so the delivery ratio will be high for large values of link availability but will decrease quickly when the link availability is lowered as the figures clearly depicts.

With no custody the delivery ratio decreases as soon as the link availability is set less than 100%. This can again be compared to UDP which behave in this way.

4.2.1 No expiration time

![Graph showing delivery ratio with no bundle expiration for the simulation model.](image)

Figure 4.7: Delivery ratio with no bundle expiration for the simulation model
4.2.2 20 seconds expiration time

Figure 4.8: Delivery ratio with no bundle expiration for the reference model

Figure 4.9: Delivery ratio with 20 seconds expiration time for the simulation model
Figure 4.10: Delivery ratio with 20 seconds expiration timer for the reference model

4.2.3 5 seconds expiration time

Figure 4.11: Delivery ratio with 5 seconds expiration time for the simulation model
Figure 4.12: Delivery ratio with 5 seconds expiration timer for the reference model

4.3 Buffer occupancy

When custody transfer is active the buffer occupancy tends to increase as the link availability is decreased. This is due to bundles being stored while waiting for a link to go up. When the expiration timer is lowered the buffer occupancy decreases since bundles are deleted from the buffer when they expire.

As Figure 4.15, 4.19, 4.16 and 4.20 shows the buffer size in the first and middle node differs quite much sometimes. This happens especially when the expiration time is set to a finite value, the buffer in the source node deletes bundles that never arrive to the middle node. When the expiration time is set to 20 or 5 seconds the theoretical limits of the maximum buffer occupancy are $20 \text{ bundles/second} \times 20 \text{ seconds} = 400 \text{ bundles}$ and $20 \text{ bundles/second} \times 5 \text{ second} = 100 \text{ bundles}$. As Figure 4.17 shows the simulator actually exceeds this limit. This can be traced to the implementation of the simulator. It only checks if any bundle has expired when it receives a new bundle or sends one. The reference implementation follows this limit much tighter, as it actually throws an interrupt when a bundle expires and deletes it immediately.

Without custody transfer the buffer is constant at one bundle, since UDP sends the bundles regardless of link status.
4.3.1 No expiration time

Figure 4.13: Maximum buffer occupancy with no bundle expiration for the simulation model

Figure 4.14: Maximum buffer occupancy with no bundle expiration for the reference model
4.3.2 20 seconds expiration time

Figure 4.15: Buffer occupancy with 20 seconds expiration timer and custody transfer for the simulation model

Figure 4.16: Buffer occupancy with 20 seconds expiration timer and custody transfer for the reference model
Figure 4.17: Maximum buffer occupancy with 20 seconds expiration timer for the simulation model

Figure 4.18: Maximum buffer occupancy with 20 seconds expiration timer for the reference model
4.3.3 5 seconds expiration time

Figure 4.19: Buffer occupancy with 5 seconds expiration timer and custody transfer for the simulation model

Figure 4.20: Buffer occupancy with 5 seconds expiration timer and custody transfer for the reference model
Figure 4.21: Maximum buffer occupancy with 5 seconds expiration timer for the simulation model

Figure 4.22: Maximum buffer occupancy with 5 seconds expiration timer for the reference model
Chapter 5

Conclusions

Validation
In this thesis enough results have been shown to see that both models, reference and simulation, behave quite similar in the tested environment. The small differences that can be seen in both models are caused by the ideal environment that the simulation uses, with no processing times and no delays in the interfaces between nodes.

It must be taken into account that the same tests made in different computers could give different results since the processing times involve a big part of the tests. For all tests with custody transfer activated, a big resend timer was selected, so that the computer times were negligible against this long time. If a lower resend timer value had been chosen, the results might not have been as conclusive as they were.

We can conclude in this thesis that the simulation model is a good option to work with the DTN architecture since it works as intended in the draft and includes some features that are not implemented in the reference model, as all different kinds of reports, which can be good for future tests and research.

Further work
As a continuation to this thesis there are some points that could be improved. A new draft has been published during the elaboration of this thesis, it would be a good idea to include the changes in the simulation model. As exposed before in this draft, there is a new paper for security issues that defines this feature more thoroughly. It would be a nice continuation of this thesis to implement all the security options into the simulator.

In order to be more efficient in different network topologies, it would be a good improvement to include some kind of dynamic routing into the simulator, to make it easier to simulate much larger networks.
Bibliography

   http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-arch-02.txt

   http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-arch-03.txt

   http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-spec-02.txt

   http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-spec-03.txt

   http://www.dtnrg.org/docs/specs/draft-irtf-dtnrg-bundle-security-00.txt

   http://www.dtnrg.org/docs/tutorials/warthman-1.1.pdf


Appendices
Appendix A

Reference Model

Code

The reference code version used is written in C++. The program has its main part in BundleDaemon.cc, which handles the basic event / action mechanism. All events are queued in an actions queue where afterwards they are consumed by BundleRouter class, the core class that calls functions to recognize, forward, store, generate and delete bundles when needed.

To follow the changes done in this thesis the reader should start in the huge class BundleRouter in /DTN2/servlib/routing/BundleRouter.cc. All events are handled in this class. The reader can in this class track the bundles way through the daemon, how signals are identified and bundles are stored and deleted.

A general scheme of the events used in this class is:

- Bundle received: handles all the actions that must be taken since a bundle arrives such as:
  - identify the kind of bundle it is (data, administrative)
  - generate signals or signal acknowledgements if needed
  - forward the bundle if the node is not the final consumer
  - initialize the expiration timer to correct value
  - answer to administrative bundles, cancelling resend timers and deleting bundles from permanent storage.

- Bundle expired: handles the bundle expiration event, deleting it from the custody list of bundles.

- Bundle retransmission: handles the retransmission of the bundles in custody when the resend timer expires. It also schedules this timer again.

- Bundle transmitted: after bundle received and retransmission event this routine is posted in the events queue. It decides if the bundle sent or
resent must be kept in custody list or should be deleted permanently. It also schedules the resend timer for new bundles.

In addition to this big class it is recommended for future developers to read changes in:

**Classes found at DTN2/servlib/bundling/ directory.**

- BundleList: a new find method was implemented to find bundles in custodian list with the only reference being its timestamp.
- CustodySignal: a class where a signal is implemented with all the fields described in the draft[3]
- Bundle: new elements as timers and a boolean variable called is_signal were added to bundle object. is_signal has tracing functions in order to measure the max buffer size. (subsection3.4.3)
- CustodyTimer and ExpirationTimer: classes that post bundle expired and bundle retransmission events when timeout.
- BundleActions, BundleEvent: expiration and retransmission events where added in these classes.

**Classes found at DTN2/applib/ directory.**

- ApiServer: (line 503) expiration time was added to the API so applications can specify this value.

**Classes found at DTN2/servlib/conv_layers directory.**

- UDPConvergence: was changed to deal correctly with the Dynamic Switch link availability.

**Users manual**

Before starting any tests with the reference application some preliminary points must be considered and configured:

- Before installing the program a database and a TCL interpreter must be installed in the computer. Follow[19] for information about installation.
- Routing tables of the kernel. Computers in the network must be able to ping each other.
- Limit of files open. This can be set in each console with the command `ulimit -n [number]`. When the number of bundles sent per second is high and the link availability is low, this value must be set to a high value, 20000 or similar so the program runs without problems.
• **Dtn.conf**, this is the file that daemon read when initializes. Routes, interfaces, directories and the name for the daemon machine must be set here. A good tutorial about how to set this file can be found in[9]

• For the daemons to work properly, the clocks of each node in the network must be synchronized, so bundles can expire in the correct moment in each node and trace files of the end node are useful at all. This can be done with a NTP server[13]

Now that all computers in the network are configured we can start all the daemons in the computers with the command

\[ DTN2/daemon/dtnd -c“your dtn.conf file” -tidy -o outputFile \]

The -tidy command restarts the database and deletes all the bundles received before, this option is recommended in the web page of the reference. If there is not specified any output file, all the messages will be printed in the console, which slows the program considerably. The daemon has different levels of logging. The level can be specified in ~/dtndebug. (More information available in[19]) For tracing purposes in this thesis the debug messages were selected in each node in a different way, so the information required could be obtained. (Check BundleRouter.cc where some messages have been added to trace bundles).

Once all the daemons are running applications for them can be written using the API of the reference. Some examples can be found in /DTN2/apps/. For this thesis dtnsend and dtnrecv were used(read[19] for further information). We created a 100 bytes file and these commands were used:

**Source node**


**End node**

\[ DTN2/apps/dtnrecv/dtnrecv test >> fileout \]

For the dtnsend application the -s option indicates the source, the -d the destination, -t the type of data is going to be sent (file in this case), -p the payload sent and -n the number of copies of the payload. Some changes were made in the dtnsend application in order to send 20 bundles per second a \texttt{usleep(35700)} was added to the send loop.

If the pipe to a file is not included in dtnrecv application all the data received is printed in console what makes the program too slow.
Appendix B

Simulation Model

For a guide to how to use the simulation see [7]

Code

The source code of the simulation model came with little comments and therefore it took some time to analyze it. It is fairly complex and not always easy to follow.

Files

- **bundlemanager**: This file is where the main processing of bundles takes place. It creates, sends, receives and process bundles. It also manages all queues and generates reports.

- **bundle**: This class describes a bundle and its contents. It contains methods for extracting data and for fragmentation and defragmentation.

- **bmflags.h**: In this file a number of different flags and their value are defined.

- **debug.h**: In this file different debug options can be set and whether or not debug info should be written to file.

- **dtn**: This is the main class of the simulation. It contains the TCL linkage which makes it possible to use the simulator in different scripts. It also contains methods which connect the model directly to the rest of NS-2.

- **headers**: This file contains methods describing the different headers a bundle can have.

- **registration**: This file handles everything to do with registration and delivery to endpoint id's.
• routes: In this file methods for routing are contained. It also contain methods used to see if links are up or down.
• string records: String records are used in the simulator to store the data sent in the bundles.
• timers: Here the two timers used in the simulator are created.

Walkthrough

This walkthrough will follow a bundle sent from the first node to the last node in a three node network.

1. A send command is given in the TCL script for the network. This is caught in the command method in dtn.cc
2. The method newBundle in bundlemanager.cc is called. It creates a new bundle with all associated headears and fields.
3. The bundle is sent to the enqueue method in bundlemanager.cc and is put into the sendqueue.
4. The queue is processed by processqueue in bundlemanager.cc which eventually works its way through the queue to the bundle.
5. It is now given to sendBundle, still in bundlemanager.cc
6. From sendBundle the bundle is sent to sendPacket in dtn.cc. This method gives the data to NS-2 which in its turn sends it out onto the first link.
7. In the middle node of the network the method recv in dtn.cc receives the data from the first node from NS-2.
8. The data is given to newBundle(packet) in bundlemanager.cc which unlike the previously mentioned sendBundle only is used on data received through a link.
9. The newBundle(packet) method decides that the bundle is not destined for the local node and gives it to the enqueue method.
10. The bundle is now picked up by processqueue again and given to sendBundle
11. From there it is given to sendPacket in dtn.cc and sent on to NS-2 and the next link.
12. The recv method in dtn.cc in the last node picks up the data.
13. It is given to newBundle(packet) and is determined to be adressed to the local node.
14. It is sent to localdelivery in bundlemanager.cc which feeds the bundle to the application.

Trace Files

The trace files created by the simulation are rather self explanatory. The first line is each tracefile describes the different fields. The events given in the files are as follows:

- **NEW**: A new bundle is created
- **FORWARD**: A bundle has been forwarded to a new node
- **RECEIVE**: A bundle has been received from another node
- **CUSTODY**: The node has taken custody of a bundle. This event is given both when a node has taken custody of a bundle and when the previous custodian receives the custody transfer notification.
- **AGENT**: Confirmation of custody transfer. This event is given both at the originating node and when it is received at the destination node.
Dynamic Switch

Dynamic switch is a program that is installed in the kernel. To be able to run it the right kernel must be loaded. Once Linux is running, the program must be run in text mode, no x-windows should be executed. When the program is running execute `dynamic_switch_2.0/process` to obtain a list of commands that can be used. For the realization of this thesis, since none of the commands were appropriated for the link availability model, a simple C program with a random exponential number generator was used. This is the source code of that program:

```c
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <gsl/gsl_rng.h>
#include <gsl/gsl_ranist.h>
#define GSL_RNG_SEED = expr {rand()}*2147483647+1
#define GSL_RNG_TYPE = mr
int main( int argc, char * argv[] )
{
    float mean_on = strtod( argv[1], NULL );
    float mean_off = strtod( argv[2], NULL );
    int pid, estado;
    int flag;
    const gsl_rng_type * T;
    gsl_rng * r;
    gsl_rng * s;
    gsl_rng_env_setup();
    T = gsl_rng_default;
    r = gsl_rng_alloc( T );
    s = gsl_rng_alloc( T );
    while ( 1 )
    {
        
```
if ( flag == 0 )
  flag = 1;
else
  flag = 0;
float k = gsl_ran_exponential( r, mean_on );
k = k * 1000000;
int g = ( int )k;
float l = gsl_ran_exponential( s, mean_off );
l = l * 1000000;
int h = ( int )l;
if ( flag == 0 )
{
  if ( ( pid = fork() ) == 0 )
  {
    usleep( g );
    execl( "process", "process", "3", "4", "down", NULL );
  }
}
if ( flag == 1 )
{
  if ( ( pid = fork() ) == 0 )
  {
    usleep( h );
    execl( "process", "process", "3", "4", "up", NULL );
  }
}
pid = wait( &estado );
pid = wait( &estado );
}

It is an easy program that is used to put one of the links up and down as long as an random exponential number generator sets. In another console, an analog program is run for the second link (changing 3 and 4 after process command per 1 and 2). As random exponential number generator[20] was employed.