Virtually Delayed: An investigation into delay tolerant networks and their emulation in a virtual environment

P Tomlinson

25 October, 2009

Department of Computing
Faculty of Mathematics, Computing and Technology
The Open University

Walton Hall, Milton Keynes, MK7 6AA
United Kingdom

http://computing.open.ac.uk
Virtually Delayed: An investigation into delay tolerant networks and their emulation in a virtual environment

A dissertation submitted in partial fulfilment of the requirements for the Open University’s Master of Science Degree in Computing for Commerce and Industry

Paul N Tomlinson

28 February 2010

Word Count: 15112
Preface

The author wishes to acknowledge of the invaluable help of:

Frank Mercer, my tutor for this project for his assistance and guidance.

Those DTN researchers on the ION and DTNRG mailing lists particularly, David Young, Scott Burleigh and Hans Kruse for their helpful and always very prompt replies.

Tad Kollar and Will Ivancic, from NASA, who kindly allowed me to use the NASA Channel Emulator.

Shaun McGarry, my former manager, for agreeing to fund this MSc over many years.

My wife who has shown incredible patience and forbearance over what must have seen a never-ending endeavour.

At last but not least my daughter who tried, as hard as a four year old can, not to disturb ‘Daddy’s project’.
# Table of Contents

Preface................................................................................................................................i
List of Figures................................................................................................................... v
List of Tables ..................................................................................................................vii
Abstract.......................................................................................................................... viii

## Chapter 1  Introduction........................................................................................................ 1

1.1  Background to the research ........................................................................................ 1
1.2  Aims and objectives of the research project ............................................................ 1
1.3  The fundamentals of Delay Tolerant Networking ....................................................... 3
   1.3.1 Store-and-Forward Message switching.......................................................... 4
   1.3.2 Types of Contact............................................................................................ 4
   1.3.3 The Bundle Layer........................................................................................... 4
   1.3.4 Bundles and Bundle Encapsulation ............................................................... 6
   1.3.5 A Non-Conversational Protocol..................................................................... 7
   1.3.6 DTN Nodes, routers and gateways ................................................................. 7
   1.3.7 Delay Isolation via Transport-Layer Termination ......................................... 7
   1.3.8 Custody Transfer............................................................................................ 8
   1.3.9 DTN Routing ................................................................................................. 9
   1.3.10 Naming and Addressing................................................................................. 9
   1.3.11 Security ........................................................................................................ 10
   1.3.12 DTN Regions ............................................................................................... 10
   1.3.13 Convergence Layer ...................................................................................... 10
1.4  Licklider Transmission Protocol............................................................................. 10
1.5  Available DTN Software .......................................................................................... 13

## Chapter 2  Literature Review .......................................................................................... 15

2.1  Background................................................................................................................. 15
2.2  Modifying TCP to improve performance over satellite links .................................. 16
2.3  TCP’s delay-bandwidth problem .............................................................................. 18
2.4  TCP replacements ..................................................................................................... 19
2.5  DTN takes shape....................................................................................................... 20
2.6  Expansion and the ‘D’ of DTN................................................................................. 21
2.7  Current status – space uses .................................................................................... 22
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.8</td>
<td>Current status – terrestrial</td>
<td>22</td>
</tr>
<tr>
<td>2.9</td>
<td>Evolution of addressing and naming in DTN</td>
<td>24</td>
</tr>
<tr>
<td>2.10</td>
<td>Routing - the biggest unsolved issue</td>
<td>25</td>
</tr>
<tr>
<td>2.11</td>
<td>Security</td>
<td>26</td>
</tr>
<tr>
<td>2.12</td>
<td>Time Synchronisation within DTNs</td>
<td>26</td>
</tr>
<tr>
<td>2.13</td>
<td>Criticisms of the Bundle Protocol</td>
<td>27</td>
</tr>
<tr>
<td>2.14</td>
<td>Latest RFCs</td>
<td>28</td>
</tr>
<tr>
<td>2.15</td>
<td>Simulation versus emulation</td>
<td>28</td>
</tr>
<tr>
<td>2.15.1</td>
<td>A convergence of simulation and emulation?</td>
<td>28</td>
</tr>
<tr>
<td>Chapter 3</td>
<td>Research Methods</td>
<td>29</td>
</tr>
<tr>
<td>3.1</td>
<td>Emulation versus simulation</td>
<td>29</td>
</tr>
<tr>
<td>3.2</td>
<td>Issues with maximum delay using netem</td>
<td>31</td>
</tr>
<tr>
<td>3.3</td>
<td>Creation of the test environment</td>
<td>31</td>
</tr>
<tr>
<td>3.3.1</td>
<td>NASA Emulator</td>
<td>31</td>
</tr>
<tr>
<td>3.4</td>
<td>Physical and virtual machines</td>
<td>35</td>
</tr>
<tr>
<td>Chapter 4</td>
<td>Data Collection</td>
<td>37</td>
</tr>
<tr>
<td>4.1</td>
<td>Process and Data Sources</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Analysis of Data</td>
<td>37</td>
</tr>
<tr>
<td>Chapter 5</td>
<td>Results</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>Theoretical TCP performance</td>
<td>38</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Retransmission timers at connection set-up</td>
<td>38</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Retransmission timers after set-up</td>
<td>39</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Slow start</td>
<td>39</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Round trip time and congestion avoidance</td>
<td>40</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Round trip time and TCP performance</td>
<td>41</td>
</tr>
<tr>
<td>5.2</td>
<td>Actual TCP performance</td>
<td>42</td>
</tr>
<tr>
<td>5.2.1</td>
<td>TCP Slow Start and Congestion Avoidance</td>
<td>46</td>
</tr>
<tr>
<td>5.3</td>
<td>DTN performance</td>
<td>50</td>
</tr>
<tr>
<td>5.3.1</td>
<td>ION using TCP as a convergence layer</td>
<td>50</td>
</tr>
<tr>
<td>5.3.2</td>
<td>ION using LTP as the convergence layer over UDP</td>
<td>52</td>
</tr>
<tr>
<td>5.3.3</td>
<td>ION over UDP</td>
<td>55</td>
</tr>
<tr>
<td>5.3.4</td>
<td>DTN2 using TCP as a convergence layer</td>
<td>56</td>
</tr>
</tbody>
</table>
List of Figures

Figure 1. The Bundle Protocol and convergence layers ................................................... 5
Figure 2. Data flow through a delay tolerant network using bundling ......................... 5
Figure 3. The Bundle Protocol primary header .............................................................. 6
Figure 4. Bundle Payload Block ..................................................................................... 6
Figure 5. Bundle status report payload header structure ............................................. 6
Figure 6. Delay isolation via transport layer termination ............................................. 8
Figure 7. LTP flow diagram ......................................................................................... 12
Figure 8. A 2-node virtual test environment ............................................................... 33
Figure 9. A 4-node virtual test environment ............................................................... 34
Figure 10. TCP connection set-up (a) normal. (b) retransmissions ............................. 38
Figure 11. Upper bound on TCP performance with increasing delay ....................... 41
Figure 12. Upper bound on TCP performance for various interplanetary RTTs .......... 42
Figure 13. Archetypal TCP connection set-up and close down ................................... 43
Figure 14. SYN retries with a 434 second delay between hosts ..................................... 44
Figure 15. Log-log graph of TCP performance ............................................................ 46
Figure 16. TCP Reno with no delay ............................................................................. 47
Figure 17. TCP Hybla with no delay .......................................................................... 47
Figure 18. TCP Reno, 600ms delay and no errors ....................................................... 48
Figure 19. TCP Hybla, 600ms delay and no errors ....................................................... 49
Figure 20. TCP Reno, 2.556s delay and no errors ....................................................... 49
Figure 21. TCP Hybla, 2.556s delay and no errors ....................................................... 50
Figure 22. Two-node ring (from ION tutorial) .............................................................. 51
Figure 23. Throughput for ION over TCP Reno ........................................................... 52
Figure 24. Two-node ring using LTP as the convergence layer over UDP

Figure 25. ION throughput over UDP at various BERs

Figure 26. ION against TCP Hybla (no added bit errors)

Figure 27. Flow diagram. One 1024 byte bundle - no custody transfer

Figure 28. Flow diagram, one bundle of 1024 bytes - custody transfer

Figure 29. Comparison of TCP and DTN2 (using TCP Reno as the CL)

Figure 30. Bundle size performance against delay

Figure 31. Three-node LTP/TCP Network

Figure 32. Three-node LTP/BP flow diagram (no custody transfer)

Figure 33. Three-node LTP/BP flow diagram (with custody transfer)

Figure 34. Locations of the inner planets in February 2010

Figure 35. One-way light time between Mars and Earth

Figure 36. Mars (1) – Moon (2) – Earth (3) emulation

Figure 37. LTP Flow Diagram for Mars-Moon-Earth transmission

Figure 38. A comparison of LTP report segments
List of Tables

Table 1. Internet and DTN assumptions ................................................................. 3
Table 2. Types of DTN contact ............................................................................ 4
Table 3. Basic LTP segment structure ................................................................. 11
Table 4. Comparison of transport protocols for space use (after Wang 2008) .... 20
Table 5. Details of the physical machines ........................................................... 35
Table 6. Details of the virtual machines ............................................................... 35
Table 7. Details of the virtualised NASA Channel Emulator ............................ 35
Table 8. The TCP variants possible in the Linux 2.6 Kernel ............................... 36
Table 9. DTN2 (over TCP) commands against TCP commands ..................... 56
Table 10. DTN interoperability results ............................................................... 60
Abstract

Delay tolerant networking (DTN) offers a novel way to successfully transfer information over severely delayed, disrupted or periodically disconnected networks. Such delays or disruption would either cause the transmission control protocol (TCP) to fail or perform extremely poorly.

The research presented here shows the creation of a virtualised test environment, including a channel emulator, which was used to show the effects of delays and errors on both TCP and two DTN implementations. The breakdown of two variants of TCP under increasing delay was shown with TCP Hybla performing much better than the default TCP Reno. The theoretical and actual performance of TCP’s retransmission timeouts was also investigated. In tests retransmission timings neither corresponded to those used by simulation software nor matched the assumptions made in some other research.

The test environment was next used to perform experiments on the DTN2 Reference implementation and Interplanetary Overlay Network. These were run over both TCP and UDP and as expected show resilience and better performance as delays are increased. Some issues were found with DTN2 running over UDP. However, ION running the Licklider Transmission Protocol showed the ability to successfully transfer an image file over delays representing the distance, in terms of light-seconds, between Mars and the Earth using the Moon as a router. When the same transfer was attempted using FTP over TCP it was a total failure despite making adjustments to timing and retry settings in both FTP and TCP.

A comprehensive literature review provides an up to date insight into DTN's short history, the state of current research and those areas that still need addressing or where debate exists.
Chapter 1 Introduction

1.1 Background to the research

The network protocols that underpin the Internet generally assume that a reliable low latency connection exists between any two points. Whilst this approach works extremely well for good quality wired and wireless links there are quite a few situations where communicating devices have neither a fast nor reliable link. Such situations include communication between the Earth and remote spacecraft, or to remote locations in the developing world, or between low power nodes participating in some ad hoc or sensor networks.

Delay or disruption tolerant networks (DTN) are a relatively new area of networking designed to overcome these problems. Although used experimentally in real life applications, both on the Earth (Lindgren et al., 2008, Isaacman and Martonosi, 2008) and in space (Wyatt et al., 2009, W. Ivancic, 2009)\(^1\), DTN is not yet being used for commercial purposes.

Most of the research is either academic or non-commercial. However, some of the software thanks to the authors, their research establishments and its open source nature, is available to the general public. This publically\(^2\) available DTN software includes the Delay Tolerant Networking reference implementation\(^3\) (DTN2), the Interplanetary Overlay Network\(^4\) (ION) and the Licklider Transfer Protocol (LTP) Reference Implementation\(^5\).

1.2 Aims and objectives of the research project

This dissertation concentrated on using network emulation in a virtual environment, as a way of showing the breakdown of TCP under network delay and investigating two DTN implementations in an inexpensive, repeatable way.

---

\(^2\) ION was subject to US Government export rules see (http://www.ocp.ohiou.edu/export.html) as it was downloaded to a non-US location. The author had to apply for a download account and create a PGP key to decrypt the download. In August 2009, it was announced that this restriction was to be lifted https://ion.ocp.ohiou.edu/node/20.
\(^4\) https://ion.ocp.ohiou.edu/index.php (Accessed March 2009)
The research has reviewed the short history and existing knowledge surrounding the rapidly developing area of DTN (Fall and Farrell, 2008). This comprised a literature search, analysis of the various requests for comments (RFCs) that have been sent to the Internet Engineering Task Force and publicly available experiments involving DTN. Criticisms of aspects of DTN such as the Bundle Protocol (Wood, 2009) have been considered.

The research created a virtualised test environment in which virtual machines connect through a virtual network emulator. The emulator was used to increase delays and introduce errors and packet loss similar to those encountered in challenged environments.

To show the usefulness of DTN, the drop in performance and eventual breakdown of TCP was examined, within this virtual environment, as delays or errors increased. The standard TCP variant was compared to one designed for situations involving longer than normal delays. The research compares and contrasts two publicly available implementations of DTN software both against themselves and against TCP.

Little similar work appears to have been attempted in these areas although, since this dissertation was begun, some research comparing TCP and DTN in the presence of channel disruptors in a part-virtualised environment has been published (Caini et al., 2009a).

Previous laboratory examination of DTNs was done using either using simulation software or network emulation on individual physical hosts. One of the criticisms of network simulation software is their sheer complexity and, for proprietary software, their costs (Farrell and Cahill, 2007). There has been some success in simulating DTN in Virtual Network User Mode Linux, although the authors did not provide specific results (Amondaray and Pascual, 2008). Some other DTN specific simulation tools have been developed but are at an early stage or have stagnated6. Other simulation tools are geared towards investigating DTN routing within a network of mobile nodes (Keränen et al., 2009). Testing using physical hosts is limited to the availability of hardware and space requirements. By using a virtual environment it is possible to create a number of hosts and for the networking parameters (for example, packet delay or corruption) to be readily varied.

6 Deep-space Network Emulation http://irg.cs.ohiou.edu/ocp/emulate.html
These aims and goals were formulated into the following research question:

‘Can a virtualised environment be used to successfully emulate the network behaviour of TCP and DTN under disrupted or delayed conditions?’

Any limitations, constraints or problems found with the research question were evaluated.

1.3 The fundamentals of Delay Tolerant Networking

Various sources were used to provide an overview of DTN (Wartham, 2003, Burleigh et al., 2003, Farrell and Cahill, 2006). The Internet Request for Comments (RFCs) are also invaluable - particularly the Delay-Tolerant Networking Architecture (Cerf et al., 2007) and Bundle Protocol Specification (Scott and Burleigh, 2007). Unless noted, these sources provide the basis for section 1.3.

The usability of today’s Internet is based around various assumptions about adequate data rate, low error rate, short consistent trip times and low jitter. DTN’s design requirements are based on the opposite assumptions. A comparison is shown in Table 1.

<table>
<thead>
<tr>
<th>Internet Usability Assumptions</th>
<th>DTN Usability Assumptions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Continuous, two-way end-to-end path during the communication session</td>
<td>No continuous end-to-end path assumed</td>
</tr>
<tr>
<td>High data rates (159Gbps for OC-3072/STM-1024)</td>
<td>Often low data rates (typically 8-256kbps)</td>
</tr>
<tr>
<td>Relatively short and consistent round trip times (in the order of milliseconds)</td>
<td>Long and/or variable delay (minutes to hours)</td>
</tr>
<tr>
<td>Supports moderately asymmetric data rates (eg ADSL)</td>
<td>Supports very asymmetric data rates</td>
</tr>
<tr>
<td>Little loss or corruption of data on each link (in a wired network)</td>
<td>High bit error rates (BER of 10^{-4} to 10^{-9})</td>
</tr>
<tr>
<td>Internet communication abstraction is limited size packets (network layer) or byte streams (transport layer)</td>
<td>DTN’s communication abstraction is variable length possibly long messages</td>
</tr>
<tr>
<td>Network congestion possible, storage congestion unlikely</td>
<td>Network congestion unlikely, storage congestion possible</td>
</tr>
<tr>
<td>Most security issues are met using end-point security mechanisms</td>
<td>Security should prevent unauthorised applications transmitting data over DTN</td>
</tr>
<tr>
<td>Common naming or address syntax</td>
<td>Naming and syntax may vary by region and be ‘late bound’ (not at bundle source)</td>
</tr>
</tbody>
</table>

Table 1. Internet and DTN assumptions
1.3.1 Store-and-Forward Message switching

DTNs use the age-old method of store-and-forward messaging. This is sometimes called a postal model as whole blocks of data (either the whole message, large fragments of it, or bundles of small messages) are moved from one storage location to another until they are eventually delivered to the destination. These blocks may be sent asynchronously with no need for a response to be received before the next one is sent. Unlike Internet routers the data may be held at the storage locations for an extended time. The blocks of data are termed bundles and the protocol itself is named bundling.

1.3.2 Types of Contact

Table 2 shows the major types of DTN contact. These contacts are defined mainly on the predictability of performance and the action required to bring them about.

<table>
<thead>
<tr>
<th>Type</th>
<th>Features</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Persistent Contacts</td>
<td>Always available, no action required starting contact.</td>
<td>Always-on Internet connection such as ADSL or Cable Modem</td>
</tr>
<tr>
<td>On-Demand Contacts</td>
<td>Needs some action to initiate and terminate.</td>
<td>Dial-up modem (from point of view of the user)</td>
</tr>
<tr>
<td>Intermittent - Scheduled Contacts</td>
<td>Agreement to establish contact at certain times for particular duration</td>
<td>Link to low-Earth orbiting satellite. Propagation delay can influence timings (eg Earth to Mars).</td>
</tr>
<tr>
<td>Intermittent - Opportunistic Contacts</td>
<td>Contact opportunities presented unexpectedly lasting undetermined amount of time.</td>
<td>Mobile Data MULEs (Mobile Ubiquitous LAN Extensions) which intermittently pick up data from sparsely located sensors</td>
</tr>
<tr>
<td>Intermittent - Predicted Contacts</td>
<td>Contacts have no fixed schedule but are predictions based on previous contacts.</td>
<td>PRoPHET and other routing protocols under development</td>
</tr>
</tbody>
</table>

Table 2. Types of DTN contact

1.3.3 The Bundle Layer

Store-and-forward message switching is implemented by overlaying a new protocol called the Bundle Protocol (BP) on the lower level network layers. The relationship between BP, LTP, other protocols designed for use over delay or disrupted networks such as Saratoga (Wood et al., 2007a) or the ‘traditional’ Internetworking protocols, is shown in Figure 1 (from ‘A Bundle of Problems’ (Wood, 2009)).
It is important to recognize that what sits below the BP is region specific so that during the course of a transmission across a DTN region the bundle layer may be above a traditional TCP/IP/Ethernet/wired network for the terrestrial part of the trip and above an LTP/CCSDS/radio network for the interplanetary parts. This is shown, from (Burleigh et al., 2003), in Figure 2.

Figure 1. The Bundle Protocol and convergence layers

Figure 2. Data flow through a delay tolerant network using bundling.
### 1.3.4 Bundles and Bundle Encapsulation

Bundles consist of a primary header, followed by a number of extension headers, the last of which is usually a payload header. The bundle primary header structure, which is mandatory, is shown in Figure 3 and contains the basic information needed to route bundles to their destinations. Figure 4 shows the bundle payload block and Figure 5 the status report payload header structure. In these figures, the fields that are greyed out are optional. Bundles can cope with being fragmented as they may be too large for transport by the lower layer protocols.

![Figure 3. The Bundle Protocol primary header](image)

<table>
<thead>
<tr>
<th>Version</th>
<th>Processing Flags</th>
<th>Class of Service Flags</th>
<th>Status Report Request Flags</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Header (or block) Length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Destination scheme offset</td>
<td>Destination Scheme Specific Part (SSP) offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Source scheme offset</td>
<td>Source SSP offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report-to scheme offset</td>
<td>Report-to SSP offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Custodian scheme offset</td>
<td>Custodian SSP offset</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation Timestamp time</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Creation Timestamp sequence number</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Lifetime</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dictionary length</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dictionary byte array</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fragment offset</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total application data unit length</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 3. The Bundle Protocol primary header**

![Figure 4. Bundle Payload Block](image)

<table>
<thead>
<tr>
<th>Header (or block) Type</th>
<th>Processing Flags</th>
<th>Header (or block) Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bundle Payload (variable)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4. Bundle Payload Block**

![Figure 5. Bundle status report payload header structure](image)

<table>
<thead>
<tr>
<th>Header Type</th>
<th>Processing Flags</th>
<th>Header (or block) Length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Status Flags</td>
<td>Reason Code</td>
<td></td>
</tr>
<tr>
<td>Fragment Length</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| Timestamps of ‘Action’ |
| Copy of Original Bundle’s Creation Time Timestamp |
| Length of Original Bundle Endpoint ID |
| Original Bundle Source Endpoint ID |

**Figure 5. Bundle status report payload header structure**

Many of the fields use the self-delimiting numeric value (SDNV) type that offers a flexible way of encoding positive numeric values.
Versions later than 1.3-29010 of the popular network packet-sniffer Wireshark now contain a DTN dissector for the Bundle Protocol in the source distribution (Chirieleison, 2009) and revisions after 31120 contain an LTP dissector (Kruse, 2009). These will decode the TCP and UDP convergence layers and a complete example is shown in Appendix C.

1.3.5 A Non-Conversational Protocol

On networks with long delays or intermittent connections conversational protocols may often be impractical. DTN bundle layers try to minimise or eradicate the number of roundtrips required. Acknowledgements are optional and depend on the class of service required. The lower network layers may be conversational like TCP but for delayed or disrupted networks it is important not to choose protocols that would impede the bundle layer.

1.3.6 DTN Nodes, routers and gateways

A DTN node is an entity running the bundle protocol and may act as a host, router or gateway (or combinations of these). These functions are detailed below:

- Hosts send and/or receives bundles but do not forward them. They need persistent storage as substantial numbers of bundles may be generated before the opportunity to transmit them takes place.
- Routers forward bundles within a single DTN region. There is nothing to prevent a router also being a host and the router must also possess persistent storage (particularly important if custody transfer is supported).
- Gateways forward bundles between DTN regions. As well as persistent storage they must support custody transfers. Gateways perform conversions between the lower-layer protocols in the regions they span.

1.3.7 Delay Isolation via Transport-Layer Termination

As Table 1 shows, Delay Tolerant Networks cannot depend on a reliable end-to-end link but may sometimes run over lower layer transport protocols, such as TCP, which require segments to be acknowledged. DTN routers and gateways terminate transport protocols at the
bundle layer. The bundle layer in a gateway acts as a proxy source and destination and isolates ‘conversational’ protocols from long delays in other regions. This is shown in Figure 6 from (Wartham, 2003).

![Figure 6. Delay isolation via transport layer termination.](image)

### 1.3.8 Custody Transfer

At its most basic, the bundle layer just provides unacknowledged, prioritised unicast messaging with no guarantee of delivery. Although lower network layers try to ensure end-to-end reliability of data transmission in a DTN, the bundle layer does provide an optional course-grained retransmission capability called ‘custody transfer’. The node submitting a bundle may request a custody transfer and starts a timer. If the next-hop node accepts the bundle it will send back an acknowledgment, take custody of the bundle, and send it to the next hop. If the sender does not receive an acknowledgement by timer expiration it will
resend the bundle. A bundle custodian must store the bundle until it has acknowledgement that another host has custody or until the bundle time-to-live expires.

Custody transfer notifications do not provide guaranteed end-to-end reliability - for this the additional ‘return receipt’ is also needed, which provides end-to-end acknowledgement (if possible).

Using transport-layer protocols and custody transfers the bundles are moved progressively along the intermediate hops towards the destination. If bundle retransmission is necessary it can be done from an intermediate retransmission point, which is much better than end-to-end retransmission over delayed, or lossy links. Return receipts, custody transfers and notifications are all classes of bundle service.

1.3.9 DTN Routing

The RFCs provide a framework for routing but the details remain one of the biggest unresolved issues in DTN. It is considered more fully in Chapter 2.

1.3.10 Naming and Addressing

BP endpoints are identified by Universal Record Identifiers (URIs), which are ASCII text strings of the general form:

\[ \text{scheme_name:scheme_specific_part} \]

Such as

dtn://topquark.caltech.dtn/mail

However, to reduce transmission overheads for space communication the Interplanetary Overlay Network uses a narrower format:

\[ \text{ipn:element_number.service_number} \]

Element numbers are analogous to IP addresses and service numbers are roughly analogous to TCP or UDP port numbers (Burleigh, 2008c). However, aspects of naming are still under discussion with various proposals being debated (see section 2.9).
1.3.11 Security

Bundle security is concerned with the authenticity, integrity and confidentiality of the bundles transmitted between nodes. Security is achieved by three bundle security blocks which may be used independently or together. Aspects of security is still under development (Farrell et al., 2009b, Symington et al., 2009).

1.3.12 DTN Regions

DTN regions have homogenous characteristics. Regions could range from the whole Earth’s Internet, to a sensor network, military tactical network, an intelligent highway or the surface of Mars. Regions have a unique ‘region ID’ which is part of each node’s name and knowable to other regions. DTN gateways have membership of two or more regions and are the only way of moving messages between regions.

1.3.13 Convergence Layer

As shown in Figure 1, the bundle protocol has to run over lower layer protocol(s). The layer immediately below the bundle protocol is called the convergence layer (CL). The CL manages the specifics of interfacing with the bundle layer. As convergence layers implement protocols (some perhaps not yet invented) outside of the bundle protocol, RFC5050 neither states all the specific services that the CL must perform nor the protocols that can act as a CL. So far draft convergence layer protocols have been written for TCP, UDP and the FLUTE multicast transport protocol (Demmer and Ott, 2008, Kruse and Osterman, 2008, Kutscher et al., 2007).

1.4 Licklider Transmission Protocol


Some of the heritage of the Consultative Committee for Space Data Systems (CCSDS) File Delivery Protocol can be found in LTP (Farrell and Cahill, 2006) but the code was developed from scratch. LTP provides a transmission protocol for delay tolerant yet reliable communication between two points. It acts as a convergence layer protocol (running above the link layer) on those legs of a journey that encounter delays or high error rates. Like much of DTN its origins are in deep space communication but it does have terrestrial applications too (Burleigh et al., 2007, Ramadas et al., 2008, Farrell et al., 2008).

LTP segments have a simple structure as shown in Table 3. The segment type shows which of the four types it is and the session ID is a random SDNV-encoded number used to identify the session. Header and trailer extension counts show the number of extensions - of which 16 are possible. Data is contained in the segment content. Extensions currently include authentication and cookies both aimed at improving security and preventing denial of service attacks.

<table>
<thead>
<tr>
<th>Version Number (4 bits)</th>
<th>Segment Type Flags (4 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Session ID</td>
</tr>
<tr>
<td></td>
<td>Header Extension Count (4 bits)</td>
</tr>
<tr>
<td></td>
<td>Trailer Extension Count (4 bits)</td>
</tr>
<tr>
<td></td>
<td>Header Extension</td>
</tr>
<tr>
<td></td>
<td>Segment Content</td>
</tr>
<tr>
<td></td>
<td>Trailer Extension</td>
</tr>
</tbody>
</table>

Table 3. Basic LTP segment structure

The basic operation of LTP is to break down blocks of data passed to it from an application or higher network layer into segments. Each segment needs to fit into the message transfer unit (MTU) of the underlying layer such as 1528 bytes for UDP. Segments are then queued and if a recipient node is – or should be – available they are transmitted.

To provide a reliable service, report segments can be sent reporting on successful delivery. The last data segment block has an end-of-block marker (EOB). Successful receipt of this causes a report segment, specifying bytes of the block received, to be sent to the source which then acknowledges this as shown in Figure 7. With the report segment various timers are reset and retransmissions can be sent if required.
LTP may send cancellation segments if too many errors or timeouts have occurred. The entire set of data and control segments sent during the transmission of a block is called an LTP session.

LTP dispenses with any TCP-like requirements for sliding windows and slow start mechanisms. For issues like data integrity, origin-authentication and repeat requests, it seeks to avoid multiple (or preferably any) round-trip messages prior to sending data. A significant difference to TCP is LTP’s ability to take ‘cues’ from lower layers which know the times and lengths of possible transmissions.

Partial reliability is provided by the concept of red and green segments. Applications can specify how many bytes of a data block should be red segments. These are typically the early parts of a transmission containing vital scientific or operational data which would render the whole block useless if not received. The ‘end of red block’ marker is treated the same way as a regular EOB marker and, if necessary, prompts retransmission. Green segments, for which the loss of a few does not impact the usefulness of the overall block, are sent without requiring reporting or retransmission until the EOB. The EOB is used to reset timers etc as green segments are never retransmitted.
Recently, LTP’s functions have been greatly modified by the Jet Propulsion Laboratory (JPL) in ION. It is now possible to optimise many different parameters (Burleigh, 2009). The price of these extra features is greater complexity, with the suggestion that parameters be calculated in a spreadsheet before LTP set-up. The modifications are not completely documented (Young, 2009).

An extension to LTP has been proposed. It is called Licklider Transmission Protocol-Transport (LTP-T) and is designed for multi-hop environments and easier connection to the terrestrial internet (Muhammad et al., 2007). In a multi-hop environment reliability and related issues are more complicated. When segment loss or corruption occurs the successfully received segments are forwarded to the next node, while error recovery is initiated for the others. This results in the initial segment sequence being desynchronised.

1.5 Available DTN Software

ION is a software implementation of the DTN architecture described in Internet RFC 4838 (Cerf et al., 2007). It is written in C and designed for flight software aboard spacecraft. As such it is optimised for real time operating systems and utilises shared memory, zero copy procedures, highly distributed processing and is portable between operating systems. Recent tests using the EPOXI (formerly Deep Impact flyby) spacecraft as a DTN router in an 11 node network, were successful with 292 image files transferred without loss (Jones, 2009). The spacecraft was 14-24 million kilometres from earth, with a signal propagation delay of 49-89 seconds and shows that ION can work properly on the sort of radiation hardened relatively low powered processors that spacecraft use (Burleigh, 2008b). As well as implementing the Bundle Protocol (Scott and Burleigh, 2007), later versions of ION starting with 1.0r203 also include LTP described in IRTF RFCs 5325, 5326, and 5327 (Burleigh et al., 2008, Farrell et al., 2008, Ramadas et al., 2008).

DTN2 (release dtn-2.6.0) is a reference implementation of the DTN architecture that aims to provide a robust framework for experimentation and real-life deployment. It is also written in C with the main focus on implementing the Bundle Protocol. It has been ported to both the Symbian and Android platforms.
LTP-RI is a Java 1.5 reference implementation of the Licklider Transmission Protocol as specified in the now superseded internet-draft: draft-irtf-dtnrg-ltp-04.txt. LTP-RI can run over UDP, TCP connections, serial port and Bluetooth datalinks. It appears to be no longer maintained and was not examined further.

Chapter 2 Literature Review

2.1 Background

The Transmission Control Protocol (TCP) and Internet Protocol (IP) have been highly successful as the core protocols of the Internet. The former deals with routing while the latter provides a reliable end-to-end byte stream adapted to the principles of congestion avoidance and fair sharing (Tanenbaum, 2002).

Although delays in setting up a link at lower network layers may impact on the performance of IP, the protocol itself does not rely on timers. It does have a time-to-live (TTL) counter that is decreased by one at each hop the route takes but this value, which can be as high as 255 or more usually 32, is rarely a problem. RFC 2549, issued on an April Fool’s Day, describes how to send IP datagrams by carrier pigeons (Waitzman, 1999). Although motivated by a joke it has been successfully demonstrated (with ping times varying from 3211 to 6389 seconds) and shows IP’s delay tolerance.

When it comes to TCP’s use over long distances, particularly to communications satellites, its performance and usefulness have been questioned. Some claim TCP performs poorly across geostationary satellite links where the round trip time (RTT) is about half a second (Partridge and Shepard, 1997b, Chotikapong et al., 2001) for a single hop. In some cases two ground hops are required. Poor performance is partly blamed on TCP’s ‘slow start’ mechanism whose exponential increase is hampered by delays. To TCP lost segments indicate congestion, requiring a further slow down, rather than segments lost by other causes, such as noise, weak signal, or missing the intended target. However, others point out that TCP has been used successfully from geostationary earth orbits (GEO) at over 400MBits/s and, if used with care, should be adequate to about Lunar distance (Hogie et al., 2005). What no one disputes, is that there comes a point when TCP is not usable. The distance at which TCP fails has been termed the ‘protocol radius’ (Wood et al., 2007b). Before TCP can establish a connection it must first successfully complete a triple handshake between sender and receiver.

10 35,786 km above the equator. From locations on the surface of the Earth, geostationary objects appear motionless in the sky.
receiver. If this is unsuccessful a connection can never be established. In theory, TCP is supposed to time-out after five minutes (RFC 793) although in practice, depending how it is implemented, the time will be shorter. One simulation found that no connection could be established with a RTT of 45s or greater. The Microsoft Windows Server 2003 implementation of TCP only allowed RTTs of nine seconds. Even this is enough for Moon/Earth communication but not, for instance, Earth/Mars where RTTs are between 8 and 40 minutes. However, although it is possible to establish a connection, the efficiency that data can be transferred falls off rapidly after a RTT of 3 seconds. Thus, even using TCP for Earth/Moon communication would be wasteful of available link capacity (Wood et al., 2007b). This poor performance is due to TCP’s Slow Start and Congestion Avoidance algorithms. These issues were examined further in this dissertation both from theoretical and experimental standpoints.

Of course, space communication both in Earth orbit and beyond predates TCP/IP’s invention and the explosion in Internet use. The view held by some in the early 1990’s was that there would be continued development of specific services and protocols for unique space based solutions. These would probably be specified and developed by the CCSDS. Others believed, even at this early stage of the Internet, that space was just one special case in a broader range of computer networks and that the ISO OSI Model (Day and Zimmermann, 1983) which was presumed to be the basis of the future ‘internet’ would suffice (Hooke, 1990). Many papers at research gatherings or in journals were geared towards signal processing, error correction, multibeam and other lower level problems (Campanella et al., 1990). Where issues higher up the network stack were examined such as problems with the end-to-nature of the higher protocols (Chitre and Lee, 1990) or later with TCP (Hogie et al., 2001), it was assumed that IP was usable for space based networks and TCP would be acceptable up to Lunar RTTs.

Recently, CCSDS has been criticised for its outdated OSI-style approach to protocols, lack of security and generally closed nature. However some of the heritage of the CCSDS File Delivery Protocol can be found in LTP (Farrell and Cahill, 2006).

2.2 Modifying TCP to improve performance over satellite links

TCP has been through various changes to improve its efficiency and robustness for normal terrestrial use. Most important were the initial development of four algorithms; slow start,
congestion avoidance, fast retransmit and fast recovery (Jacobson, 1988). Starting with TCP-Tahoe, there has been a move to TCP Reno, TCP NewReno, TCP SACK, TCP Vegas and TCP Westwood etc. Most of these changes to TCP have revolved around modifying the recovery process rather than slow start and congestion avoidance (Brakmo and Peterson, 1995, Politis et al., 2006).

Not surprisingly researchers have also looked at ways to modify TCP for less reliable wireless terrestrial networks, or for earth orbiting satellite links, or for heterogeneous networks, where end-to-end connections span both wired and wireless links. Alterations to standard TCP, such as Selective Acknowledgements (which allows for multiple losses in a transmission window to be recovered in a single RTT) and Path MTU discovery (sender probing for the largest allowable Message Transfer Unit) all help (Qureshi et al., 2009, Henderson and Katz, 1999). Methods such as proxying (Border et al., 2001), splitting the end-to-end connection so as to isolate the wireless legs, or modifying the lower network layers using spoofing techniques, also make TCP more satellite friendly (Caini and Firrincieli, 2004, Hu and Li, 2001, Ishac and Allman, 2001, Luglio et al., 2004). Various types of performance enhancing proxy have been suggested, and indeed used, in GEO satellite communication. These essentially work by intercepting connections that involve a satellite link and "stealing" the TCP SYN packet in TCP’s three-way handshake set-up. They then pretend to be the other side of the connection, initiate a new connection to the real endpoint, and using a userspace application, directly copy data between the two sockets (Caini et al., 2009c, Caini et al., 2009b, Shen et al., 2008). New variants of TCP that differentiate between segment loss caused by congestion and loss caused by attenuation, such as employed TCP Hybla (Caini and Firrincieli, 2004) and TCP Westwood, have been proposed. So too have more radical alterations with new algorithms such as TCP-Peach, TCP-Peach+ and TP-Satellite (Jiong et al., 2009, Akyildiz et al., 2001, Akyildiz et al., 2002, Shen et al., 2008).

Much of the blame for TCP’s perceived poor satellite performance is placed upon its slow start mechanism. Firstly, the probing mechanism can take a long time to get up to speed as it is influenced by RTT. Secondly, it is often the case that the entire transfer will have taken place before the slow start algorithm has finished, so the whole transfer has taken place at a far lower speed than what could have been realised (Partridge and Shepard, 1997a).
TCP-Peach proposes to get over these problems with a ‘Sudden Start’ algorithm that uses NIL segments (which can be regarded as dummy segments), to probe for network resources and recover from errors. Other algorithms such as modifications to Congestion Avoidance, Fast Retransmit, and Rapid Recovery complete Peach’s satellite friendly enhancements. One issue with Peach is the requirement for the routers along the connection to use a priority mechanism at the IP layer to discard dummy (NIL) segments in the presence of congestion (Caini and Firrincieli, 2004).

TCP-Hybla, mentioned above, is available in most Linux kernels if compiled with the “TCP: advanced congestion control” option. It is designed to show considerable improvements over standard TCP on wireless and satellite connections, without infringing the end-to-end design of TCP. The details are complicated but, in essence, are partly achieved by making transmission rate independent of RTT, using mandatory selective acknowledgement policy and additional use of timers (Caini and Firrincieli, 2004, Caini et al., 2007). TCP Hybla is examined in section 5.2.

At a physical level, developments like the Cisco Space Router Series introduce several new features to satellite networking. One feature is onboard routing between transponders. This removes the need to “double-hop” network traffic from the satellite to a ground gateway hub. Another is the onboard routing between terminals using different waveforms. This reduces latency by shortening the end-to-end path compared to switching at ground stations (Cisco, 2009).

2.3 TCP’s delay-bandwidth problem

When standard TCP sees packet loss as a sign of congestion it has the effect that delayed acknowledgements, which are in transmission, can force a retransmission of data that has actually been received correctly. The transmitting node needs to keep these transmitted packets buffered, until it receives an acknowledgment that the receiving node has accepted the packet (Brooks, 1999). This means that in a network with a bandwidth of 622 Mbps and a RTT of 532 milliseconds the transmitting system requires at least 350 Mb of buffer for storage of transmitted packets waiting for acknowledgments (Welch et al., 1998).
A useful summary considering whether the standard internet suite could be used for interplanetary networking, but which primarily focuses on TCP to the exclusion of other potentially more applicable protocols like UDP, has been produced (Durst et al., 2000).

**2.4 TCP replacements**

Complete replacements to TCP have been proposed. These range from an assessment of whether existing protocols like the Stream Control Transmission Protocol are suited for space networks (Fu et al., 2005) to entirely new protocols such as TP-Planet which is proposed for use on IPN backbone networks (Akan et al., 2004). A short critique of TP-Planet pointed out that it still requires a working end-to-end connection for the entire duration of the transport session (Farrell and Cahill, 2006). A comprehensive review of all current data transport protocols for use in space has been produced comparing solutions involving just TCP modification against those requiring complete TCP replacement against DTN and other protocols (Wang et al., 2008). Table 4 shows a modified and slightly updated list.

<table>
<thead>
<tr>
<th>Protocol Names</th>
<th>Application Environment</th>
<th>Problems Solved</th>
<th>Congestion Control Mechanism</th>
<th>Modification to TCP Transmission Control</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>High BER</td>
<td>Long Delay</td>
<td>Channel Asymmetry</td>
</tr>
<tr>
<td>SCTP</td>
<td>Satellite Networks</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>STP</td>
<td>LEO/GEO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>XSTP</td>
<td>LEO</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TCP Peach</td>
<td>Satellite (IP) Networks</td>
<td>Yes</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>TP-Planet</td>
<td>Deep space interplanetary links</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>TCPW</td>
<td>Wired/wireless networks, especially lossy wireless networks</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>XCP</td>
<td>High BDP Networks</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>P-XCP</td>
<td>Same s XCP</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>REFWA</td>
<td>Multihop satellite environments</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>REFWA Plus</td>
<td>Multihop satellite environments</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Protocol</td>
<td>Purpose</td>
<td>Space Environments</td>
<td>Thermal Environment</td>
<td>Antenna</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------</td>
<td>--------------------</td>
<td>---------------------</td>
<td>---------</td>
</tr>
<tr>
<td>SCPS-TP</td>
<td>Earth-orbit space environments</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>I-PEP</td>
<td>Earth-orbit space environments</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>&quot;Split-TCP&quot; Proxy</td>
<td>Satellite Networks</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>PETRA</td>
<td>Targeted GEO space but including LEO</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>BP</td>
<td>Deep space interplanetary links</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>CFDP</td>
<td>All space environments</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LTP</td>
<td>Deep space interplanetary single-hop links</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>LTP-P</td>
<td>Deep space interplanetary multi-hop links</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>ETEN</td>
<td>Earth-orbit space environments</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table 4. Comparison of transport protocols for space use (after Wang 2008)

2.5 DTN takes shape

When NASA’s Jet Propulsion Laboratory and the Mitre Group started examining IP-based standards to address deep space communications in 1997, they initially looked at modifying TCP to cope with long delays and the sender and recipient not being in real-time contact. In 1998, assisted by Vint Cerf, the co-inventor of TCP/IP (Cerf and Kahn, 1974) and part funded by the Defense Advanced Research Projects Agency (DARPA), the idea of an InterPlaNetary Internet began to form (Wiggins, 1998). The InterPlaNetary Internet Special Interest Group of the Internet Society was founded and several Internet draft RFCs were published (Cerf et al., 2007, Cerf et al., 2002). The suggested architecture followed four principal components:

- Follow standard Internet rules for short delay communications – wherever they take place.
- These local autonomous internets are then interconnected by a specialised deep-space backbone network of long-haul links.
- Significantly, a set of new overlay protocols called Bundling would tie together the heterogeneous internets to support end-to-end communications.
Unlike the original Internet protocols security would be built into both the backbone and exchange of data. (Hooke, 2001).

Whilst roughly analogous to TCP/IP the bundling protocol would operate in an environment characterised by extremely long delays and intermittent – if predictable – interruptions to communication. It would do this by working in a store-and-forward mode holding bundles until a forward path was available.

The phrase “Delay-Tolerant Networking” was introduced in a seminal paper at the 2003 ACM Special Interest Group on Communications (SIGCOMM) meeting (Fall, 2003). This paper drew from design work already done for the Interplanetary Internet. Afterwards, the Delay Tolerant Networking Group of the Internet Research Task Force was founded and began developing the DTN architecture and its Bundle Protocol. The most important RFCs produced so far are:


2.6 Expansion and the ‘D’ of DTN

The scope of DTN has widened since its original deep-space origin. DTN has now expanded into terrestrial applications such as sensor networks and highly disrupted networks. In fact, it is quite possible that the ‘D’ in DTN will end up standing for Disruption rather than Delay (Farrell and Cahill, 2006). This process appears to be happening (as of late 2009) if the titles of various papers serve as a guide (Wyatt et al., 2009, Holliday, 2009). Alternatively, the ‘D’ could stand for Disconnection or DTN could be replaced with the new phrase Eventually Transportable Dynamic Network (Ivancic, 2010).
2.7 Current status – space uses

Several tests have now been performed on DTN technologies from earth orbit (W. Ivancic, 2009) and deep space some 32 million kilometres from Earth (Burleigh, 2008b). This aspect of DTN has started to get wider publicity with the ‘The Orbital Internet’ one of Time Magazine’s Best Inventions of 2008 (Hamilton et al., 2008).

It is briefly worth mentioning Saratoga, which is an alternative to the implementations of DTN mentioned so far. Saratoga is a lightweight transport protocol based on UDP (Wood et al., 2009) and has already been used to send files from low earth orbit (W. Ivancic, 2009). Alterations to Saratoga allow it to serve as a convergence layer for DTN Networking so that bundles can be sent as well as files (Wood et al., 2007a). The Saratoga convergence layer sits between the Bundle Protocol and UDP datagram. As the source code for Saratoga is not publicly available it was not examined further.

Some dispute has been highlighted between DTN researchers and those who believe existing Internet protocols can be modified. This disagreement was described as mainly between intra-NASA researchers (Jackson, 2005). In a reply, Cerf and Israel, stated that DTN was not designed to “compete with or to replace TCP/IP, but to complement it” (Cerf and Israel, 2005).

2.8 Current status – terrestrial

One interesting test of terrestrial DTN has been in northern Sweden where DTN2 was used as the core network to give some of the Sámi population of reindeer herders access to the Internet for such things as email, cached web access and ‘Not So Instant Messaging’ (Lindgren et al., 2008).

Military applications of DTN are being developed such as the US Marine Corp’s Command and Control On-the-move Network, Digital Over-the-horizon Relay (CONDOR) project. One aspect still to be developed for these applications is a way to ‘protect’ DTN data lost when a drone or other device crashes or is captured (Jonson et al., 2008).

The US Defense Advanced Research Projects Agency’s (Darpa) Advanced Technology Office (ATO) has solicited proposals on developing or demonstrating key technologies for
solving some of DTN’s “problem areas” [Darpa’s phrase]. Although some of the military aspects of DTN have been discussed there are some areas not in the public domain or where research has not been published in peer-reviewed journals. For instance, it may be necessary to develop specific, and presumably classified, military DTN extensions\textsuperscript{11}.

Environmental uses of DTN include pilots for lake water quality and noise level monitoring using Sensor Network with Delay Tolerance (SeNDT) nodes. The nodes send data wirelessly using LTP over UDP (McDonald et al., 2007). Within the European Union, legal requirements such as the Water Framework Directive whereby Member States must aim to “reach good chemical and ecological status in inland and coastal waters by 2015” (The European Parliament, 2000), may act as an impetus to provide monitoring results using cheap sensor networks running DTN (Farrell and Cahill, 2006).

Whilst nodes in a sensor network have a variety of ways to communicate, one method particularly favourable to DTN is the concept of Data MULEs (Mobile Ubiquitous LAN Extensions) (Seligman et al., 2008) in which mobile entities pick up data from sparsely located sensors, buffer it and deliver the data to access points from where it can reach the Internet (Shah et al., 2003).

To help with testing and developing DTN applications a ‘DTN Bone’ is being developed by various research groups to provide a worldwide collection of nodes running DTN bundle agents and applications. DTN Bone allows various DTN implementations to connect with it and also allows interconnection between different implementations.

A recent development in DTN is ‘Hermes’ a project for ‘Delay Tolerant Bulk Transfers’ on the Internet being developed by Telefonica Research. The aim of this project is to transfer massive amounts of non-time dependent data using left over capacity from commercial ISPs. One intention is to undercut the cost of the physical transport of data on, for example, tape by commercial courier which currently is still much cheaper (Laoutaris et al., 2009).

\textsuperscript{11} Disruption Tolerant Networking – Summary Brief, 15 September 2005
It has also been suggested that DTN might be used to provide a service to those whose governments block certain types of Internet access as such censorship might be interpreted as infringing basic human rites (Fall, 2009).

2.9 Evolution of addressing and naming in DTN

As mentioned in section 1.3.10 the overall DTN naming structure proposed in RFC 5050 is generally accepted. However, new variations of naming are under discussion. One idea proposed at the March 2009 DTNRG meeting would allow additional formats. They all start with:

    dtn: which indicates use of bundle protocol.

For example, the display of web pages that, for normal networks, are delivered by http could be:

    dtn: [existing-URI]

    E.g., dtn:http://www.dtnrg.org

    Indicates text from web page carried in BP

Other possibilities include:

    dtn:ether: 00:17:f2:e7:6d:91

    Indicates a bundle carried in a MAC layer frame

Or:

    dtn:ip4: 12.110.110.204

    Indicates a bundle carried within an IP datagram (in this case IP v4)

More exotic examples such as content based addressing are possible:

    dtn:flood:sql:where_name_like_bill

or text to execute:

    dtn:exec:c:while(foo)[bar;]
2.10 Routing - the biggest unsolved issue

Routing in DTN is still an active issue. For space-based applications routes – although complicated – may be simpler to calculate than for terrestrial networks. For instance, the Interplanetary Overlay Network (ION) uses ‘Contact Graph Routing’ (CGR). This is a dynamic routing system that can compute routes through a topology where scheduled contacts between nodes vary over time. CGR relies on the fact that deep space communication opportunities are planned in detail. Therefore communication routes between pairs of nodes that have been informed of each others’ plans can be calculated from those plans, rather than discovered through the usual routing methods such as distance vector or link state routing, which would be impractical over long distance space links. CGR could also have applications for terrestrial routing if a predefined schedule exists for communications (Burleigh, 2008a).

Compared to deep space, terrestrial routing will usually be more complicated. Some experiments, both simulated and real world, have used the PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) routing protocol for DTN. This algorithm calculates a probabilistic metric called ‘delivery predictability’ at each node for each known destination, which is used to decide if the destination is reliable enough to forward a message to. Delivery Predictability is calculated using knowledge obtained from past encounters with other nodes (Lindgren et al., 2003).

Other types of routing for DTN include MaxPROP (Burgess et al., 2006) and Context-Aware Routing (Musolesi et al., 2005). Many other types of routing are possible and some are applicable to DTN, mobile ad hoc networks (MANETs) and sensor networks. For an excellent compendium on different routing protocols see (Zhang, 2006).

Routing remains the most important unresolved issue in DTN and some researchers speculate that DTN routers (which in practice would be most DTN nodes) should be able to route bundles using various possible routing schemes (Farrell et al., 2006). This could be handled by a brokering agent that makes intelligent decisions about which scheme to use (Peoples et al., 2007).
2.11 Security

Security of DTN is another developing area (Symington et al., 2009) which has been actively considered from the start. This did not happen in the more innocent days when TCP/IP was invented. The lengthy round trip times and often-disrupted connections that characterize DTN, preclude security solutions that need continuous connection to online security resources. One useful design possibility is to use concepts from the Transport Layer Security (TLS) specification (Dierks and Allen, 1999, Dierks and Rescorla, 2006, Blake-Wilson et al., 2006), particularly the idea of a “ciphersuite” which, very simply, is a single number describing how all the various cryptographic algorithms are used. Using a TLS-like approach it is possible to design DTN so different parts of the bundle are protected by relevant security services. In many cases this aids integrity rather than confidentiality as it could be helpful where fragments of a bundle have been received. It is important to note, in these considerations of security, that it is not possible to use any of the handshake protocols that TLS and other transport layer security protocols use, as they suffer the same problems as TCP’s triple-handshake (Farrell and Cahill, 2006).

2.12 Time Synchronisation within DTNs

For some time there has been debate within the DTN community about whether time synchronisation is necessary for a DTN to operate and, if it is, how accurate the time should be. Much of the discussion is recorded in the main DTN mailing list (with dtn-interest Digest, Vol 22, Issue 27 being especially useful). The debate shows that despite fully functional implementations that are just starting to be used in real-world, or off-world applications, there are still fundamental issues to be addressed in key areas.

Some researchers, such as Kevin Fall (Intel), believe that where time is unavailable or is just so far out of synchronisation as to be wrong this should be considered as an anomaly in DTN. Others, like Armando Caro (BBN), are of the opinion that the lack of accurate time should not be considered an anomaly in DTNs, especially when the ‘D’ stands for Disruption. Nor, when it comes to generalizing a delay/disruption tolerant store and forward networking protocol, should time synchronisation be a requirement.

Much of the argument stems from RFC 5050’s reliance on time synchronisation using ‘DTN time’ for many timestamps, administrative records and importantly bundle expiry. DTN time
is the number of seconds since the start of the year 2000 followed by the number of nanoseconds since the start of the indicated second. Both values are SDNV encoded. Scott Burleigh (JPL NASA), one of the joint authors of RFC 5050, views the reliance on time synchronisation as a design decision, for which both valid arguments and counter-arguments may be made. Burleigh further states that successful DTN spaceflight tests show that only synchronisation to within ten seconds is needed and this is easy to achieve. While true, the DINET flight validation report does state that strict regulation of clocks is required for ION to manage bundles and contact windows, and significant further work is needed to synchronise clocks throughout a delay tolerant networks (Jones, 2009). The situation is further complicated if exceedingly accurate time is needed for space-based applications as the effects of General Relativity must then be addressed (Pogge, 2009).

Issues with handling ‘real-time’ in the bundle protocol and ways to handle bundle expiry without real time clocks are discussed in an IETF Internet-Draft (Farrell et al., 2009a).

As this dissertation made extensive use of virtualisation it is worth noting that clock synchronisation in virtual machines is a complex topic and severe clock drift could occur for older Linux VMs. Linux kernel versions from 2.6.21 onwards are far better (VMware, 2009) and clock drift issues for this dissertation were not a problem while performing experiments. Pausing and resuming virtual machines did reduce DTN2 data throughput and generate warning messages about clock times so were avoided.

2.13 Criticisms of the Bundle Protocol

A critique of the BP’s perceived problems has been provided. In particular its reliability, due to having no useful error detection built in, is questioned. Error correction provided by the overlying application or underlying layers is not perceived as good enough. At the very least it is believed that BP’s own headers and ‘metadata’ should be verified by checksums. Various proposals are made in which the bundle security specification is used to ‘wrap’ bundle payloads and certain header information using a ‘reliability-only cyphersuite’ (Wood, 2009).

Problems with fragmentation, time synchronisation, a proliferation of possible convergence layers, network management, quality of service and last, but not least, security were identified. Various proposals were made for BP’s improvement in the hope that the “Bundle
Protocol may one day be ready for operational use”. As for alternative DTN approaches, it was proposed to use an adaptation of the HTTP protocol most familiar for providing web pages (Wood and Holliday, 2009).

### 2.14 Latest RFCs

As at February 2010, as well as the five RFCs referenced above, there are now also seven draft-irtf-dtnrg-* drafts in review with eight other non-expired draft-irtf-dtnrg-* drafts and six other non-expired draft-<author>-dtnrg-* drafts.

### 2.15 Simulation versus emulation

Practical simulation tools for DTN are still lacking or unobtainable. The DTNSim2 Delay-Tolerant Network Simulator\(^{12}\) is finally viewable again (May 2009) after months of having a broken hyperlink and being unreachable by other search methods. It is an undocumented dump of source code and is not particularly relevant. Despite frequent visits, the details and source code, to “be posted soon” of a Deep-space Network Emulation tool have never appeared from researchers at Ohio University\(^{13}\).

On a more theoretical level, the design of a general purpose network emulator which is also applicable to interplanetary networks has been described (Zhihao et al., 2006). They concluded that a distributed network emulator was the best testing tool for use in the design of protocols for interplanetary networks. Network emulation rather than simulation was also considered preferable in other research (Endres et al., 2004) and is considered in Chapter 3.

#### 2.15.1 A convergence of simulation and emulation?

Of interest is that the new version of the general network simulation software ns-2, which is called ns-3, will pay more “attention to realism. An ns-3 Node is a husk of a computer to which applications, stacks, and NICs are added\(^{14}\)” This provides a solution that “model(s) nodes more like a real computer”. It is easy to imagine that future network simulators will closely model a real system and the distinction between simulation and emulation will blur.

---

\(^{12}\) [http://watwire.uwaterloo.ca/DTN](http://watwire.uwaterloo.ca/DTN) (accessed May 2009)


\(^{14}\) [http://www.nsnam.org/docs/ns-3-overview.pdf](http://www.nsnam.org/docs/ns-3-overview.pdf)
Chapter 3  Research Methods

As much of the rationale for DTN relies on the unsuitability of TCP for use on links characterized by long delays, and to a lesser extent disruption, it is worth putting this into context with some theoretical work and actual experiments. These also show than an emulated virtual environment provides a valid comparison to a physical one.

3.1 Emulation versus simulation

The rationale for using DTN implementations in an emulated environment rather than through network simulation has been neatly described (Farrell and Cahill, 2007). These authors rejected using network simulators in their experiments with LTP-T for three reasons:

- Firstly, they shared some scepticism as to how useful or faithful simulators could be, especially as DTN is significantly different from typical simulated or real networks. It is not always possible to validate the results presented by research using network simulators due to the “hidden variables” problem. Most of the simulators have many fairly opaque settings. In one paper the authors found 174 separate numeric or string settings. These make the analysis or replication of results very difficult.

- Secondly, there are difficulties where a particular implementation is tied to a network simulator. This is especially important if the aim is to achieve interoperability with other DTN implementations.

- Finally, in testing a number of protocols (such as LTP, LTP-T, BP and the non-DTN sftp) in parallel, it becomes very difficult in a simulation to repeatedly validate results against reference code.

These difficulties with simulation lead them to conclude that reliable and comparable studies were best done using emulation. For this they principally used the netem module\(^\text{15}\) that is part of the Linux 2.6 kernel. Netem is simple to use, for instance the command:

\[
\text{tc \ qdisc \ add \ dev \ eth2 \ root \ netem \ delay \ 750ms}
\]

introduces a delay of 750ms for all packets leaving network interface eth2. Whereas:

\[
\text{ tc \ qdisc \ add \ dev \ eth2 \ root \ netem \ delay \ 750ms}
\]

\[
\text{http://www.linuxfoundation.org/en/Net:Netem} \quad \text{(accessed March 2009)}
\]

\(^{15}\)
causes 0.1% (i.e. 1 out of 1000) packets to be randomly dropped.

More complex delays, such as delay distribution, packet reordering, packet duplication and corruption are possible (Hemminger, 2005). The maximum delay on a single network interface with netem is around 36 minutes. The reasons for this are investigated in section 3.2 with a definitive answer provided.

Other work (Endres et al., 2004) compared emulation versus simulation in relation to space based network emulation. The advantages of simulation were that it could be performed on a single workstation in less than real time but at the cost of being based on performance models constructed by the researcher that excluded what they called ‘nuances’. Emulation, on the other hand, only models hardware performance, can include testing of nuances within the software, runs in real time but requires multiple workstations.

The validity of emulation over simulation was further confirmed when work on TCP’s ‘protocol radius’ is considered (Wood et al., 2007b). Both the simulation tools used in this work, OPNET\textsuperscript{16} and ns-2\textsuperscript{17}, fail to give particularly reliable figures. With OPNET, sender and receiver failed to set up a TCP connection when RTT exceeded 45 seconds. The ns-2, on the other hand, just doubled its initial send request until a connection was finally established. Whilst not strictly incorrect (see section 5.1.1) this does not happen in real life. Wood et al stated “it would be helpful to examine real implementations of TCP with a delay emulator that stores and releases packets”. Using such a delay emulator, it is shown in section 5.2 that a standard Ubuntu Linux virtual machine can still establish a connection and transfer some data with an 80 second delay using its default variant of TCP with the default number of SYN retries. Using a variant of TCP designed for, amongst other uses, satellite communication data could still be transmitted with a 140 second delay again without altering the number of SYN retries. Altering TCP parameters allowed the triple handshake to be accomplished at delays of more than 400 seconds.

\textsuperscript{17} The Network Simulator - ns-2 \url{http://www.isi.edu/nsnam/ns/} (accessed May 2009)
3.2 Issues with maximum delay using netem

During this project it was found no delays greater than 36 minutes could be created with netem. This was investigated and a networking researcher provided the definitive explanation. A delay added using netem is stored as a 32-bit unsigned integer that represents nanoseconds (even though the kernel cannot create a delay in increments any smaller than one millisecond). This gives a theoretical limit of $2^{32} - 1$ns (around 72 minutes). However, at one point in the code there is a signed operation that results in a maximum reliable value of $2^{31} - 1$ ns, or roughly 2147s. If a value greater than this is entered, the resulting delay is nearly random since part of the number is lost. If used anyway, it can occasionally cause the kernel to crash. A patch is available to the Linux kernel that increases the delay to a maximum of 585,000 years (Kollar, 2009).

3.3 Creation of the test environment

When running emulations it is important to create an environment that is robust, easily configured and simple to replicate. It would be possible to use netem to create delays on the virtual machines themselves. However, when netem causes packet loss locally, the loss is reported to the upper level protocols. This can cause TCP to resend and behave as if there was no loss. A better solution is to use an emulator that sits between the machines in an IEEE 802.1d bridged fashion.

3.3.1 NASA Emulator

NASA’s Glenn Research Center has produced a Channel Emulator\(^\text{18}\) (CE) which operates as a link layer bridge (not as an IP router) and provides many features useful for modelling space data links. Some of the features are provided by netem.

Although normally unavailable outside the *.nasa.gov domain, Tad Kollar kindly allowed the author to download the CE and also provided source code allowing netem to produce greater delays than the default (Kollar, 2009).

The CE is based on the Linux Knoppix distribution and is downloadable as a bootable ISO image. This image was converted to a virtual machine with initially three virtual network

interfaces. One connected to the physical network allowing the emulator to be managed either via SSH or its web interface. The other two interfaces connect to virtual switches that, in turn, connect to the virtual machines. Up to five pairs of virtual interfaces can be created. Thanks to the CE’s use of the UNIONFS file system it is possible to save the configuration files between reboots onto a USB memory stick something not normally possible with ISO images (Wright and Zadok, 2004).

The CE allows all the usual netem commands to be used on its network interfaces; delay, jitter, packet loss, packet duplication and bit error rate. It also allows one-way communication for layer-3 traffic. To provide a one-way link (using Ethernet) the emulator needs to ‘cheat’ by still allowing ARP packets to pass both ways.

### 3.3.1.1 Virtual Test Environment

The test environment used in this dissertation was created using VMware Workstation on a Windows XP computer and VMware Fusion on an Apple iMac (details of the specifications are shown in Table 5). A simple two-node environment with two hosts communicating via the channel emulator is shown in Figure 8 and a four-node environment is shown in Figure 9. The VMs sit inside a partially isolated virtual network where traffic between virtual machines does not use the host’s network card or any physical switch. While communicating between themselves the virtual machines are unaffected by the host’s own network traffic.

As the network configuration for each virtual machine is held in a simple text-based file it is easy to alter the number of VMs or the switch they are connected to. Switch creation was a little more complex especially with Fusion where the ‘Tokamak Networking Scripts for VMware Fusion’ (Parsons, 2008) were used. Each host, and the emulator, have another virtual network card that serves for management purposes. This management network is connected to the physical host by interface en0 and allows Internet access for downloading later versions of the DTN software etc. During tests virtual machine interfaces connecting to the management network were disabled to prevent extraneous traffic interfering with results.
Figure 8. A 2-node virtual test environment
Figure 9. A 4-node virtual test environment

The only issue encountered when creating five VMs was that too much of the host’s memory was used. Some unnecessary services and daemons were disabled on the VMs and, apart from the ones running Wireshark, the X graphical interface was turned off.
3.4 Physical and virtual machines

The two physical machines available are shown in Table 5, with the virtual machines in Table 6. The Channel Emulator details are shown in Table 7.

<table>
<thead>
<tr>
<th>Work Computer</th>
<th>Home Computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Manufacturer &amp; Model</td>
<td>Dell Optiplex</td>
</tr>
<tr>
<td>Processor</td>
<td>2.4 GHz dual-core</td>
</tr>
<tr>
<td>Memory</td>
<td>2 GB</td>
</tr>
<tr>
<td>Network Interface</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td>Operating System</td>
<td>Microsoft Windows XP (SP3)</td>
</tr>
<tr>
<td>Virtualisation software</td>
<td>VMware Workstation 6.5</td>
</tr>
</tbody>
</table>

Table 5. Details of the physical machines

<table>
<thead>
<tr>
<th>Ubuntu Virtual Machine - settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Linux kernel</td>
</tr>
<tr>
<td>Virtual Processors</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>TCP Congestion Control</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Fedora Virtual Machine - settings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Linux kernel</td>
</tr>
<tr>
<td>Virtual Processors</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Networking</td>
</tr>
<tr>
<td>TCP Congestion Control</td>
</tr>
</tbody>
</table>

Table 6. Details of the virtual machines

<table>
<thead>
<tr>
<th>NASA Channel Emulator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
</tr>
<tr>
<td>Linux kernel</td>
</tr>
<tr>
<td>Virtual Processors</td>
</tr>
<tr>
<td>Memory</td>
</tr>
<tr>
<td>Network</td>
</tr>
<tr>
<td>TCP Congestion Control</td>
</tr>
</tbody>
</table>

Table 7. Details of the virtualised NASA Channel Emulator

The kernel build of these virtual machines had the advanced networking options enabled and could use any of the 12 TCP variants shown in Table 8.
To prevent over complexity the default ‘Reno’ algorithm was used along with Hybla which was especially designed for longer RTTs and heterogeneous networks (Caini and Firrincieli, 2004).

<table>
<thead>
<tr>
<th>TCP Advanced Congestion Control in Linux 2.6 kernels</th>
</tr>
</thead>
<tbody>
<tr>
<td>Binary Increase Congestion (BIC) control</td>
</tr>
<tr>
<td>CUBIC TCP</td>
</tr>
<tr>
<td>TCP Westwood+</td>
</tr>
<tr>
<td>H-TCP</td>
</tr>
<tr>
<td>High Speed TCP</td>
</tr>
<tr>
<td>TCP-Hybla congestion control algorithm</td>
</tr>
<tr>
<td>TCP Vegas</td>
</tr>
<tr>
<td>Scalable TCP</td>
</tr>
<tr>
<td>TCP Low Priority</td>
</tr>
<tr>
<td>TCP Veno</td>
</tr>
<tr>
<td>YeAH TCP</td>
</tr>
<tr>
<td>TCP Illinois</td>
</tr>
<tr>
<td>Reno (Default)</td>
</tr>
</tbody>
</table>

Table 8. The TCP variants possible in the Linux 2.6 Kernel
Chapter 4  Data Collection

4.1 Process and Data Sources

For performance measurement the most important tools used were iperf, tcpprobe, dtnperf and bpdriver. Wherever possible large samples were produced to reduce the uncertainty in the measured mean and standard deviation. Procedures were run on both Windows and Apple hosts and no differences were found. Results were transferred to a spreadsheet for further analysis and graphing.

4.2 Analysis of Data

Wireshark proved invaluable for examining network flows and investigating issues such as timer expiry. The latest development versions have dissectors for both BP and LTP but the latter still has some issues to be resolved. All extraneous network traffic other than that being measured was removed as far as possible and other network interfaces were shutdown.

All data was stored unfiltered for further analysis. This proved useful as Wireshark’s DTN dissectors continued to be refined.

The metrics employed for analysing the primary results of the research were measurements of throughput, goodput (uncorrupted throughput), bit error rates, queue lengths and bandwidth delay. They were analysed using standard statistical techniques. Especially useful for graphs were log scales as many TCP performance curves show an exponential decrease with RTT increase.

The performance of the physical host and virtual machines (CPU usage, disk activity, memory and network activity) were monitored both before and during experiments. As mentioned in section 3.3.1.1 memory on the physical hosts was reduced unacceptably when running more than four virtual machines. However, reducing the VM memory footprint solved this problem without altering DTN or TCP performance.

No factors that impact on data collection methods could be identified.
Chapter 5  Results

5.1 Theoretical TCP performance

The behaviour of TCP has been extensively investigated (Mathis et al., 1997) and continues to be a source of enquiry. In this dissertation only those areas of TCP performance dealing with aspects of connection set-up, slow start, and congestion avoidance were considered of interest.

5.1.1 Retransmission timers at connection set-up

Figure 10. TCP connection set-up (a) normal. (b) retransmissions.

Under normal conditions for TCP connection set up, the sender transmits a SYN segment for the receiver to acknowledge with a SYN/ACK before the sender transmits an ACK segment to complete the three-way handshake. This is shown in Figure 10(a). In the absence of a reply the default timeout before resending a SYN is 3 seconds and then TCP implements a doubling exponential backoff with SYNs sent at 9, as in Figure 10(b), 21 and 45 seconds etc. There is no formal upper limit to the number of resends. Microsoft Windows 2003 Server defaults to two retransmissions (in a range of 0-255) giving an RTT of 9 seconds and a radius of 4.5 light-seconds. The Linux kernel used in this work defaults to 5 (in a range of 0-8192) giving an RTT of 93 seconds. Interestingly, the upper limit for Windows would theoretically be $1.74 \times 10^{77}$ seconds giving a radius greater than diameter of the visible universe of about 93
billion light years. In reality, the increments are not calculated this way (see Section 5.2) a finding that has not been reported by others (Farrell et al., 2006, Wood et al., 2007b).

5.1.2 Retransmission timers after set-up

TCP uses a number of timers to function properly and the most important is the retransmission timer (Paxson and Allman, 2000). When a segment is sent the retransmission timer is started. The timer is stopped if a segment is acknowledged before timer expiration. If the timer expires before the segment is acknowledged the segment is retransmitted (and the timer restarted). Setting the right timer value is important and most versions of TCP first use a smoothed deviation variable $D$ with the formula:

$$D = aD + (1 - \alpha) \left| RTT - M \right|$$

Equation 1

where:

$RTT$ is the Round Trip Time

$\alpha$ is a smoothing factor (often 0.125)

$M$ is the latest measurement of RTD

The timeout value is then usually set at:

Timeout = $RTT + 4 \times D$

5.1.3 Slow start

TCP’s Slow Start algorithm is:

$$t_{\text{SlowStart}} = RTT \cdot (1 + \log_2 B \cdot RTT / l)$$

Equation 2

where:

RTT is the round-trip time

$l$ is the average packet length (bits)

$B$ is the bit rate

Although (Partridge and Shepard, 1997a) use a slightly different equation.
The default maximum segment size (MSS) for TCP is 536 bytes but over Ethernet it is often 1500 bytes.

Equation 4 gives the time taken for the slow start algorithm to reach a window size of \( W \) segments presuming no segment loss occurs (Jacobson, 1988):

\[
t_{\text{SlowStart}} = RTT \log_2 W
\]

Equation 4

Assuming a window size of 128 segments (TCP’s default maximum) and 600ms for a single-hop GEO RTT then it takes 4.2 seconds to increase the congestion window to its maximum size. For an Earth/Moon situation, where the RTT is around 2.58 seconds, it would take just under 18 seconds.

### 5.1.4 Round trip time and congestion avoidance

TCP uses the round trip time for congestion control. It sees increasing RTT as symptomatic of congestion and uses a binary exponential back off algorithm whereby if transmission fails – causing the retransmission timer to elapse – then the value of the timer is doubled. This reduces the traffic offered by the sender. For these reasons the calculation of the time-out period is important and the most common method is to base it on an estimate of the average RTT (The Open University, 2004). The measured RTT will fluctuate so, as in Equation 5, it is smoothed by averaging:

\[
RTT_{\text{new}} = (1 - \alpha)RTT_{\text{old}} + \alpha M
\]

Equation 5

Where:

- \( RTT \) is the Round Trip Time
- \( \alpha \) is a smoothing factor (often 0.125)
- \( M \) is the latest measurement of RTD

(RFC 2988, 2000)
5.1.5 Round trip time and TCP performance

The upper bound on TCP’s performance is represented by Equation 6:

\[ BW < \left( \frac{\text{MSS}}{\text{RTT}} \right) \frac{1}{\sqrt{p}} \]  

Equation 6

Where:
- \( BW \) is bandwidth
- \( \text{MSS} \) is maximum segment size in bytes (1460 or 536)
- \( \text{RTT} \) is round trip time (ms)
- And \( p \) is the loss event rate

(Mathis et al., 1997)

Using a 1460 byte MSS and three different packet loss values produces the performance graph shown in Figure 11.

![Upper Bound on TCP Performance](insert graph)

Figure 11. Upper bound on TCP performance with increasing delay

Equation 6 is a bound on performance with the actual data rate below this level. As shown in Figure 11, performance declines rapidly as delay increases up to about 5 seconds mark, and
there afterwards declines gradually. In the case of deep-space communication the delay is not caused by congestion but by the speed of light.

A highly theoretical upper bound on TCP throughput is shown in Figure 12, from (Durst et al., 2000). This assumes that a connection can still be achieved even at Jovian distances.

Figure 12. Upper bound on TCP performance for various interplanetary RTTs.

5.2 Actual TCP performance

It is possible to fine-tune TCP’s variables in multiple ways. Unless specified default values were used below. Details of TCP settings and modifications are shown in Appendix D.

An actual TCP Reno flow diagram between the Ubuntu virtual machines (specifications shown in Table 6) using the iperf performance tool is shown in Figure 13.

---


TCP Flow Diagram (time in seconds) – captured using Wireshark on 10.1.1.1. TCP Congestion Control is the default (Reno).

Command run on 10.1.1.1: iperf -s -B 10.1.1.1. Command run on 10.1.1.2: iperf -t 1 -f b -c 10.1.1.1

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>10.1.1.2</th>
<th>10.1.1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>SYN</td>
<td>10.1.1.1</td>
</tr>
<tr>
<td>0.023</td>
<td>SYN, ACK</td>
<td></td>
</tr>
<tr>
<td>0.023</td>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>0.023</td>
<td>PSH, ACK - Len: 24</td>
<td></td>
</tr>
<tr>
<td>0.046</td>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>0.046</td>
<td>ACK - Len: 2896</td>
<td></td>
</tr>
<tr>
<td>0.080</td>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>0.080</td>
<td>PSH, ACK - Len: 2896</td>
<td></td>
</tr>
<tr>
<td>0.080</td>
<td>ACK</td>
<td></td>
</tr>
<tr>
<td>0.080</td>
<td>ACK - Len: 2896</td>
<td></td>
</tr>
</tbody>
</table>

Intermediate data excluded

| 0.000   | ACK - Len: 2896 |         | Seq = 1 Ack = 1 |
| 0.000   | FIN, PSH, ACK - Len: 200 |         | Seq = 2897 Ack = 1 |
| 0.000   | ACK      |         | Seq = 1 Ack = 4294903585 |
| 0.000   | ACK      |         | Seq = 1 Ack = 4294906481 |
| 0.012   | ACK      |         | Seq = 1 Ack = 4294961505 |
| 0.012   | ACK      |         | Seq = 1 Ack = 4294964401 |
| 0.012   | ACK      |         | Seq = 1 Ack = 1 |
| 0.026   | ACK      |         | Seq = 1 Ack = 2897 |
| 0.034   | FIN, ACK |         | Seq = 1 Ack = 3098 |
| 0.034   | ACK      |         | Seq = 3098 Ack = 2 |

Figure 13. Archetypal TCP connection set-up and close down

It shows an archetypal TCP connection set up, data transfer and connection close down. The sender transmits a SYN segment for the receiver to acknowledge with a SYN/ACK before the
sender transmits an ACK segment to complete the three-way handshake. The ACK also piggybacks a PSH command to start pushing data at the receiver. As is quite common a three-way connection release also takes place with a FIN/ACK, FIN/ACK, ACK sequence.

<table>
<thead>
<tr>
<th>Time (s)</th>
<th>10.1.1.2</th>
<th>10.1.1.1</th>
<th>Seq = 0 Ack = 2355244475</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.000</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>2.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>8.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>20.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>44.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>92.780</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>189.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>308.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>428.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>548.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>668.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>788.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>908.997</td>
<td>SYN (34239)</td>
<td>SYN (5001)</td>
<td>Seq = 0 Ack = 2355244475</td>
</tr>
<tr>
<td>996.644</td>
<td>SYN (34239)</td>
<td>ACK (5001)</td>
<td>Seq = 0 Ack = 1</td>
</tr>
<tr>
<td>996.645</td>
<td>ACK (34239)</td>
<td>SYN, ACK (5001)</td>
<td>Seq = 1 Ack = 1</td>
</tr>
</tbody>
</table>

Figure 14. SYN retries with a 434 second delay between hosts

Figure 14 shows, from host2’s perspective, what happened when a 434 second delay was introduced. Connection establishment involves retransmission timeouts (RTOs) and the sender waits 3 seconds, the default value, for a reply to its initial SYN before sending another. As this is not replied to, the sender doubles the waiting time. This leads to exponential back-off and SYNs at 9, 21, 45, 93 and 189 seconds, as stated in section 5.1.1, until the seventh retry when the time is simply increased by 120 seconds for this and all further retries. The reason for the change is not apparent (Paxson and Allman, 2000) and does not appear in one of the many TCP Linux kernel parameters settings that can be altered. However, it can be surmised that calculating increasingly huge exponential back-off times
would become a computationally intensive operation and many of TCP’s calculations are designed to be efficient. For instance, Equation 1 has the advantage of being calculated using only integer adds, subtracts and shifts (Tanenbaum, 2002).

As multiple tests were conducted the command `sysctl -w net.ipv4.tcp_no_metrics_save=1` was run so that Linux will not remember the last slow start threshold (ssthresh). If this was not done it could modify the result of the subsequent tests.

Actual throughput of two versions of TCP (Reno and Hybla) was examined using iperf with no added bit errors. Host1 ran the iperf server process whilst host2 used iperf to transmit a random TCP byte stream so the time to put data ‘on the wire’ was not an issue. The log-log graph is shown in Figure 15. Four measurements were taken at 50 different time intervals (made using the CE) for each algorithm. Reno shows better performance at very short delays but after 10 ms Hybla starts to outperform Reno for the duration of the test. Both protocols suffer big performance drops at 8 seconds for Reno and 15 seconds for Hybla. Both appear to be timeouts but the exact cause was not found (Caini and Firrincieli, 2004). Additional measurements were made to ensure these drops were not artefacts. The last data that could be transmitted was 253 bps at 80 seconds for Reno and 776 bps at 140 seconds for Hybla.
In reality, many applications higher than TCP in the application stack utilize their own timeouts. For instance, the widely used ‘secure shell’ application has a “LoginGraceTime” of between 10 and 120 seconds by default.

5.2.1 TCP Slow Start and Congestion Avoidance

The next series of experiments show the effects of increased RTTs on the time taken to reach the slow start threshold and the size of the congestion window.

The scripts used to produce these results are shown in Appendix B. Both hosts always used the same version of TCP. As well as forcing the sender not to retain the last slow start threshold, changes were also made to increase the receiver’s TCP receive buffers, as it was important to only concentrate on the impact of delays to the sender.

The delays added using the channel emulator were 0 ms, 600 ms and 2556 ms, these were chosen to represent; a normal situation, the RTT for a typical single-hop GEO satellite and a Earth-Moon RTT. The TCP window size was set to 16.0 KB (the default value). In the
following graphs the sender Congestion Window (snd_cwd) and sender Slow Start Threshold (snd_ssthresh) are plotted as segments against time.

Figure 16. TCP Reno with no delay

Using Reno in Figure 16 the sender can be seen probing for bandwidth by increasing the congestion window until a segment is lost. This gives a characteristic saw-toothed graph.

Figure 17. TCP Hybla with no delay
Hybla’s rapid recovery from congestion or lost segments is shown in Figure 17 but on connections with minimal delay it offers no greater throughput than Reno.

Figure 18. TCP Reno, 600ms delay and no errors

Figure 18 shows Reno’s congestion window which grew slowly over the first 13 seconds being badly impacted by timeouts/losses and having to grow again from a very low base.
As delays increased, Hybla quickly ramps up the SST irrespective of RTT, halving it when a loss is encountered. Compare y-axis to Reno’s y-axis in Figure 18.

Figure 20. TCP Reno, 2.556s delay and no errors
Figure 21. TCP Hybla, 2.556s delay and no errors

Figure 20 again shows Reno taking even longer to ramp-up throughput. Hybla’s performance graphs at this delay period are inconsistent and Figure 21 is representative of a sample of ten.

5.3 DTN performance

Tests were performed on ION and DTN2 over both TCP and UDP. Details of the machines used are shown in Table 5 and Table 6.

5.3.1 ION using TCP as a convergence layer

A simple throughput exercise was performed using ION installed on two Ubuntu Linux virtual machines. The Channel Emulator was used to increase the delay from 0 to 10 seconds. Repeating the experiments on Windows and Apple hosts showed no differences at all. The two-node ring setup for the test is show in Figure 22.
Figure 22. Two-node ring (from ION tutorial)

Throughput was measured using ION’s testing and benchmarking programs:

- **bpdriver** (on host1) uses the Bundle Protocol (BP) to send a specified number of application data units to a BP endpoint. For the experiment below, bpdriver operated in streaming mode with custody transfer requested for all bundles. 100 cycles were performed of 100000 bytes.
- **bcounter** (on host2) uses Bundle Protocol to receive application data units from the host running the bpdriver application task when in streaming mode.
- **bpecho** (on host2) uses the BP to receive application data units from the host running the bpdriver application task when in echo mode.

Four experiments were performed. bpdriver was run in both ‘streaming’ and ‘echo’ modes. For each mode both custody and non-custody transfer was invoked. The results are shown in Figure 23.
Streaming mode outperforms echo for obvious reasons but the differences between custody and non-custody transfers are minimal.

5.3.2 ION using LTP as the convergence layer over UDP

The Two-node ring configuration file was edited so that it used LTP instead of TCP as the convergence layer. This then ran over UDP, see Figure 24 (adapted from the ION Tutorial).
Various experiments were done and ION options tested:

Sending 10 cycles of 1000 bytes in streaming mode. This does not wait for acknowledgement of bundles:

```
bdriver 10 ipn:2.1 ipn:1.1 -1000
```

Streaming mode was not considered further as success was just dependent on suitable storage being available (bandwidth delay product) along the route and errors being below a rate correctable by the various network layers.

Sending 10 cycles of 1000 bytes in echo mode. This does wait for acknowledgement of bundles:

```
bdriver 10 ipn:2.1 ipn:1.1 1000
```

In the following tests an actual (200KB) file was sent from host 2 to host 1 using the commands;

```
host1 bprcvfile ipn:1.1
host2 bpendfile ipn:2.1 ipn:1.1 PIAo9097-800x600.jpg
```

and a progressively longer delay was added using the Channel Emulator with the results plotted. To simulate both a delayed and disrupted network three different bit error ratios (BERs) were also added. To give an indication of potential BERs suggested figures are $10^{-8}$ to $10^{-7}$ for almost clear sky, $10^{-6}$ to $10^{-4}$ for hard link intermittance and $10^{-3}$ to $10^{-2}$ for deep-fading periods (Cola et al., 2007).

The results are shown in Figure 25. With no added bit errors it was possible to send data with a 4000s delay which is 66 minutes (or 33 minutes on both network cards in the CE). At higher BERs there was a sudden failure in data transmission (for example after 0.02s at 10% BER and after 60s at 0.1% BER) with throughput dropping to zero. UDP has no concept of acknowledgement or retransmission but LTP will attempt to retransmit corrupt segments until it is impacted by time-outs. It is also worth noting ION’s performance at extremely small delays was often worse than at slightly higher values. This appears to be caused by the receiving buffer in the LTP engine being overwhelmed with data. The lost data then needs retransmitting. As ION’s author stated, “at high data rates in terrestrial deployment environments its performance hasn’t been all I’d hoped” (Burleigh, 2009).
In Figure 26, ION, using LTP over UDP, was compared against TCP Hybla and the results vividly show that whilst TCP initially outperforms ION it becomes incapable of sending data at a 140 second delay whereas ION still works with a 4000 second delay. In terms of RTT, at the speed of light, this is just under 600 million kms. (The distance from Earth to Mars is between 55-400 million kilometers.)
5.3.3 ION over UDP

As the current Wireshark disector (version 31120) misinterprets control flags as errors (Roy, 2009) the false messages were removed from the flow diagrams in Figure 27 and Figure 28.

Figure 26. ION against TCP Hybla (no added bit errors)

Figure 27. Flow diagram. One 1024 byte bundle - no custody transfer

```
Figure 27. Flow diagram. One 1024 byte bundle - no custody transfer
```
5.3.4 DTN2 using TCP as a convergence layer

Data sent using DTN2 - with TCP Reno as the convergence layer (Demmer and Ott, 2008) - was compared against data sent straight over two variants of TCP. The commands are shown in Table 9. No errors were introduced and delays up to five seconds were examined.

| Time (s) | | 10.1.1.1 | 10.1.1.2 |
|----------|-------------------------------|---------------------------------|
| 0.000    | Data Segment                  | LTP Segment: Red, CP, EORB, EOB |
| 0.000    | Report segment                | LTP Segment: Report segment     |
| 0.001    | Data Segment                  | LTP Segment: Red, CP, EORB, EOB |
| 0.002    | Data Segment                  | LTP Segment: Red, CP, EORB, EOB |
| 2.003    | Report segment                | LTP Segment: Report ack segment |
| 2.004    | Report segment                | LTP Segment: Report segment     |
| 2.005    | Report segment                | LTP Segment: Report ack segment |
| 2.006    | Report segment                | LTP Segment: Report ack segment |

key: Red = Red Data, CP = Checkpoint, EORB = End of Red Block, EOB = End of Block

Figure 28. Flow diagram, one bundle of 1024 bytes - custody transfer

<table>
<thead>
<tr>
<th></th>
<th>host1</th>
<th>host2</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP Test</td>
<td>iperf -s</td>
<td>iperf -c host3 -t 30</td>
</tr>
<tr>
<td>DTN2 Test</td>
<td>./dtnperf-server</td>
<td>./dtnperf-client -d dtn://host3.dtn -t 30 -p 10M</td>
</tr>
</tbody>
</table>

Table 9. DTN2 (over TCP) commands against TCP commands.

The results are shown in Figure 29 (no bit errors were added).
Figure 29. Comparison of TCP and DTN2 (using TCP Reno as the CL).

Figure 29 shows both TCP variants initially outperforming DTN2 with, as expected, Hybla performing best as delays increase to around 0.5 seconds. Thereafter performance of both types of TCP falls off rapidly with DTN2 tailing off at a far slower rate. At a five second delay, DTN2’s throughput outperforms Hybla by ten times and Reno by 45 times. DTN’s performance was in fact indistinguishable whether it was run over Reno or Hybla.

5.3.5 DTN2 using UDP as the convergence layer

When DTN2 was tested using UDP as the convergence layer extremely low throughput was observed and failures observed when trying to send all but the smallest files.

The reason is down to fragmentation or rather the need to avoid fragmentation. In theory, the maximum UDP datagram size is 65 KB and “when using UDP directly as a CL, the software SHOULD NOT directly encapsulate large bundles into large UDP datagrams that would need to be fragmented” (Kruse and Osterman, 2008). The UDP convergence layer adds a header, possibly but unlikely, up to 1KB in size. As bundles must be sent inside a single UDP datagram the bundle should not exceed about 64KB.
In reality the situation is worse as the underlying datalink layer, such as Ethernet, rarely supports the maximum UDP size and fragments the datagrams for reassembly at the far end. This process, like UDP, does not handle retransmission. Over a disrupted network UDP datagrams big enough to trigger IP fragmentation can end up never being received. The “practical upper limit for the size of bundles sent via the UDP Convergence Layer is nearer to 2 to 3 times the Maximum Transmission Unit (MTU) of the underlying medium. For Ethernet, the MTU is 1500 bytes”\(^{21}\).

Practical experiments show that after a delay of 40 seconds 10KB bundles could no longer be sent and even 1KB bundles could not be sent after 50 seconds. The poor performance of DTN2 over UDP compared to TCP is shown in Figure 30.

![DTN2 bundle size performance against delay](http://www.dtnrg.org/docs/code/DTN2/doc/manual/cl-udp.html)

Figure 30. Bundle size performance against delay

5.3.6 DTN2 using Ethernet or Bluetooth as the convergence layer

DTN2 was designed to use Ethernet or Bluetooth as additional convergence layers. Inspection of the source code showed both ‘temporarily’ disabled\(^{22}\). Uncommenting the code and recompiling proved fruitless and more importantly no feasible Ethernet or Bluetooth delay products were found. As a consequence no further work was done on these techniques.

5.3.7 DTN Interoperability

A number of tests assessed the interoperability of DTN 2.6 and ION. No delays or errors were added, as the intention was purely to test if files could be sent between the two implementations. All tests were done using DTN 2.6 and ION 2.0_r245_ou708-disconnectathon. Minor configuration changes were needed compared to the examples used in DTNRG\(^{23}\).

After starting the DTN implementation on each host, a 20KB image file was sent from ION (host4) to DTN2 (host1) with the command:

```
 bpsendfile  dtn://host4.dtn/sink  dtn://host1.dtn/testing  pia00111-1024-768.jpg
```

On the receiving node running DTN2 the dtnrecv command was waiting to receive bundles:

```
./dtnrecv -o p p##.jpg  dtn://host1.dtn/testing
```

In a successful test the image was received as ‘p00.jpg’ and subsequent images as ‘p01.jpg’, ‘p02.jpg’ etc. All successfully received images were identical to the sent image.

The first image file was then sent back from DTN2 to ION with the following command run on host1:

```
./dtnsend -D -s “dtn://host1.dtn/a” -d “DTN://host4.dtn/sink” -t f -p ..dtnrecv/p00.jpg
```

While host4, running ION the bprecvfile command, was waiting for the bundles;

\(^{22}\) [http://dtn.sourceforge.net/cgi-bin/hgwebdir.cgi/DTN2/rev/722a72961285](http://dtn.sourceforge.net/cgi-bin/hgwebdir.cgi/DTN2/rev/722a72961285)

In a successful test the image was received as ‘testfile’, with subsequent images as ‘testfile1’, ‘testfile2’ etc.

Successful file transfers between the implementations of DTN are shown in Table 10. It was not possible to send bundles over UDP from ION to DTN2. Again this was linked to MTU size. If the bundles were larger than the link MTU the DTN2 daemon crashes. This bug was confirmed right at the end of the dissertation (Roy, 2010).

<table>
<thead>
<tr>
<th></th>
<th>To DTN2</th>
<th>To ION</th>
</tr>
</thead>
<tbody>
<tr>
<td>From DTN2</td>
<td>TCP, UDP</td>
<td>TCP</td>
</tr>
<tr>
<td>From ION</td>
<td>TCP, UDP</td>
<td>TCP</td>
</tr>
</tbody>
</table>

Table 10. DTN interoperability results

5.4 Three node experiments

The three node networks were again relatively simple. The first was a slightly modified version of the example from the ION Tutorial. It is shown in Figure 31. Host2 acts as a router between host1 and host3. This is achieved in the configuration files by making host2 the gateway for host1 and host3.

![Figure 31. Three-node LTP/TCP Network](image)

A simple test involved sending a short message and date from host1, via host2, to host3:

On host1 (10.1.1.1)

```
Echo "hello host3 at \"date\" | bpsource ipn:3.1
```

On host3 (10.1.1.3)

```
‘hello host3 at Thu Dec 1 21:49:00 GMT 2009’
```

The flow diagram in Figure 32 shows LTP being used for the first leg of the trip to host2 with associated acknowledgements. Host2 then routes the message using BP and TCP CL to host3. Due to the workings of bpsource no custody transfer or status reports are requested.
Figure 32. Three-node LTP/BP flow diagram (no custody transfer)

Figure 33 shows the results with custody transfer invoked, for example, by using the `bpdriver` command with appropriate options;

Figure 33. Three-node LTP/BP flow diagram (with custody transfer)

### 5.4.1 Emulation of Mars – Moon – Earth data transmission

After considering theoretical simple two-node and three-node cases, it is time to consider a more realistic situation and emulate it in a virtual environment. One such, slightly futuristic, situation would be the transmission of data from a satellite around Mars to a ground station on Earth via a relay station on the Moon. The situation will be simplified in that direct communication between all three bodies is assumed which does not occur in reality.
Using JPL’s HORIZONS system the distance between many objects in the solar system can be calculated and converted into light time. As at December 2009 and into late-January 2010 the orbits of Earth and Mars moved closer and their February 2010 positions are shown in Figure 34 (generated using JPL Solar System Simulator\textsuperscript{24}). The one-way light time (OWLT) declines from 435 to 331 seconds before increasing again (Figure 35). The OWLT between the Earth and Moon also varies very slightly but we will assume a mean value of 1.29 seconds.

![Image of the solar system showing Earth, Mars, Venus, and Mercury in February 2010.](http://space.jpl.nasa.gov/)

Figure 34. Locations of the inner planets in February 2010

\textsuperscript{24} http://space.jpl.nasa.gov/
5.4.1.1 Attempt to transfer an image file using FTP over TCP

A number of changes were made to the default TCP Hybla settings in an attempt to use the file transfer protocol to send an actual 80.28 KB photograph\(^{25}\). In particular, all timeouts and retries were increased to allow the 345s OWLT to be accommodated. Port forwarding was setup on host2 (Moon router) to transfer data on to host3 (the Earth). Finally, an FTP server (vsftp) was installed on host3 and anonymous, passwordless authentication set to avoid the need for extra roundtrips or user interaction. The FTP server also had its default timeouts increased to avoid the connection being dropped due to delays.

The transfer of the 80KB image file was then attempted and proved a total failure. It was just possible (at BERs <0.01%) to set up an FTP connection but successful authentication was never achieved preventing file transfer. As expected, the three-way handshake was achieved at around 1430 seconds (from the sender’s perspective). Soon after an enormous number of duplicate ACKs were received and a TCP reset was initiated from the sender that closed the connection.

\(^{25}\) http://photojournal.jpl.nasa.gov/jpeg/PIA12393.jpg
connection. Initially it was assumed that FTP was responsible but the resets were also initiated when iperf was used. This appeared to be a TCP issue not configurable using sysctcl.

Even with a 10 second, rather than a 345 second delay, FTP performance is very poor with successful logon only achieved at 58 seconds and the first data transmitted at 248 seconds. Whilst data transmission was successful any BER resulting in corrupt TCP segments was enough to prevent successful transfer.

### 5.4.1.2 Successful use of LTP over UDP

After the failure of FTP over TCP, and with DTN2’s problems sending large files over UDP, ION was tested next. Careful modification of the ION configuration files allowed two types of three-node network to be created with settings altered for very long delays. The first uses LTP over UDP for the longest part of the trip, host1 to host2, and BP over TCP for shorter part, host2 to host3. This is shown in Figure 36(a). The second, commented on below, used LTP for both legs of the trip and is shown in Figure 36(b). The Moon (host2) acts as a router.

![Figure 36(a)](image1)

![Figure 36(b)](image2)

**Figure 36. Mars (1) – Moon (2) – Earth (3) emulation**

Various situations were emulated but the most sensible was the transmission of the same photograph as used in the FTP experiment using both custody and non-custody transfers. For LTP the ‘red part’ comprised the whole file (as it has for real-life experiments (Burleigh, 2008b)).

The data was successfully transferred taking on average 694 seconds for the sender to receive a success confirmation. This equates to a throughput of 115 bps.

One issue initially encountered, after examining Wireshark dumps, was the large number of cancellation segments that LTP generated. This was tracked down to the command used to
send files called bpsendfile that has an in-built time-to-live setting of 300 seconds.
Recompiling bpsendfile.c with a higher time-out value prevented cancellation segments.

A stylised LTP flow diagram (using the scenario in Figure 36b) from when the Earth/Mars one-way light time was 345 seconds is shown in Figure 37. It shows the transfer from Mars (host1) of an LTP block (in actual fact around 50 data segments). This arrives at the router on the ‘Moon’ (host2) which sends a single acknowledgment segment back to Mars before forwarding the bundle on to the ‘Earth’ (host3). Upon successful receipt of the bundle on Earth the Moon router is informed and sends a series of receipt segments covering all the data segments back to Mars. These report segments where then acknowledged.
The transmission of data proved resilient to errors. Even at a BER of 0.1% the uncorrupted transmission of the file was successful on five out of five tests and was achieved on four out of five tests at 1%. These BERs are higher than the working assumption in Table 1 and 1% corresponds to the ‘deep-fade’ figure from (Cola et al., 2007). At a BER of 10% no successful data transfer was possible.
As expected, when data was corrupted LTP reports on the data received. This is shown in Figure 38 where a report segment showing successful reception of a valid data segment is compared to one showing only partial receipt of valid data. This is reported using offsets, so data between the start of the segment and 5557 is valid but there is corruption somewhere between 5558 and 47197. This part of the data segment, being flagged as ‘red’, is eligible for retransmission.

<table>
<thead>
<tr>
<th>Report Segment – data received successfully</th>
<th>Report Segment – some data corrupted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Internet Protocol, Src: 10.1.1.2 (10.1.1.2), Dst: 10.1.1.1 (10.1.1.1)</td>
<td>Internet Protocol, Src: 10.1.1.2 (10.1.1.2), Dst: 10.1.1.1 (10.1.1.1)</td>
</tr>
<tr>
<td>Licklider Transmission Protocol</td>
<td>Licklider Transmission Protocol</td>
</tr>
<tr>
<td>LTP Header</td>
<td>LTP Header</td>
</tr>
<tr>
<td>LTP Version: 0</td>
<td>LTP Version: 0</td>
</tr>
<tr>
<td>LTP Type: 8 (Report segment)</td>
<td>LTP Type: 8 (Report segment)</td>
</tr>
<tr>
<td>Session ID</td>
<td>Session ID</td>
</tr>
<tr>
<td>Session originator: 1</td>
<td>Session originator: 1</td>
</tr>
<tr>
<td>Session number: 975</td>
<td>Session number: 612</td>
</tr>
<tr>
<td>Header Extension Count: 0</td>
<td>Header Extension Count: 0</td>
</tr>
<tr>
<td>Trailer Extension Count: 0</td>
<td>Trailer Extension Count: 0</td>
</tr>
<tr>
<td>Report Segment</td>
<td>Report Segment</td>
</tr>
<tr>
<td>Report serial number: 288558281</td>
<td>Report serial number: 1680768159</td>
</tr>
<tr>
<td>Checkpoint serial number: 261051083</td>
<td>Checkpoint serial number: 1882256822</td>
</tr>
<tr>
<td>Upper bound: 80301</td>
<td>Upper bound: 80301</td>
</tr>
<tr>
<td>Lower bound: 0</td>
<td>Lower bound: 0</td>
</tr>
<tr>
<td>Reception claim count: 1</td>
<td>Reception claim count: 3</td>
</tr>
<tr>
<td>Reception claims</td>
<td>Reception claims</td>
</tr>
<tr>
<td>Offset[0] : 0</td>
<td>Offset[0] : 0</td>
</tr>
<tr>
<td>Length[0] : 80301</td>
<td>Length[0] : 5557</td>
</tr>
<tr>
<td></td>
<td>Offset[1] : 8335</td>
</tr>
<tr>
<td></td>
<td>Length[1] : 47198</td>
</tr>
</tbody>
</table>

Figure 38. A comparison of LTP report segments

5.5 Validation

Experiments were performed in a regulated environment in which it was assumed transmission opportunities were always available. In reality this would not be the case and other software would alert the nodes as to connection opportunities and timings.
The research focussed on two particular virtualisation products from VMware. There was certainly no difference between results on these two products running on different hardware platforms but generalisation to other virtualisation products should be approached with caution.

The key difference between a physical network and a virtual network is that in a virtual network the hardware devices are represented in software within the virtualisation environment. However, one of the essential characteristics of a virtual machine monitor (which includes VMware Workstation and Fusion) is that software on the VMM executes identically to its execution on hardware, barring timing effects (Popek and Goldberg, 1974). There should be, and research seems to support the idea, no appreciable differences between networking in a virtual and physical environment (Adams and Agesen, 2006). As far as the virtual machines are concerned the virtual network adaptors appear as physical hardware devices specifically an Intel Pro/1000 MT in this dissertation.

5.6 Analysis

Relatively few problems were identified that could have affected the research but a number of issues were found which initially caused problematical results.

When ION was operated in streaming-mode, that is, without the sender waiting for acknowledgments, two problems were identified. The first when operating with either TCP or UDP as the convergence layer - and with very short delays - was that ION’s receiving buffers could be overwhelmed. A second problem when sending large amounts of data over UDP with very long delays was the bandwidth delay product.

The problems with receiving buffers cannot be readily fixed and it has to be acknowledged that DTN software currently does not work well on high-speed non-delayed networks that are not DTN’s intended habitat. Increasing the CE’s queue length did fix the bandwidth delay product issues at the expense of using more virtual machine memory.

Choosing an appropriate bit error rate to approximate deep space communications is somewhat complicated. The error rate on deep space transmissions is very high with the uncoded bit error rate often around $10^{-1}$. This has lead to considerable effort being placed on error correction methods. Using forward error correction techniques, such as a concatenated
code composed of inner convolutional code combined with Reed-Solomon outer code, can bring error rates down to better than $10^{-9}$ (Durst et al., 2000) or $10^{-7}$ (Hogie et al., 2005). Techniques such as these are outside the scope of this dissertation and it assumed that underlying protocols would handle the majority of error correction. Thus leaving LTP and the BP to handle a BER of between 0.001% and 0.1% seems a reasonable approach and agrees with the parameters suggested in the Channel Emulator (Kollar, 2008).
Chapter 6  Conclusions

The research clearly shows that a virtualisation environment can be used to successfully emulate the network behaviour of TCP and DTN under delayed or disrupted conditions. The creation of an across platform virtualised environment is relatively straightforward with the only real limitation being host computer memory. The advantages include; the ability to clone more virtual nodes as necessarily, reversion to previous snapshots in the event of problems and easy network reconfiguration. The use of a channel emulator, in this case also virtualised, is essential for accurate results as delays or errors introduced locally on (virtual) machines are often reported to the upper level protocols resulting in immediate retransmissions.

TCP’s problems with increasing delay are vividly shown and compare well to other research using physical emulation. The improvement of TCP Hybla compared to Reno over delayed links was investigated and congestion windows and slow start thresholds were graphed. These show Hybla’s distinctive way of dealing with the problem of slow start and its better performance over delayed networks compared to Reno. Interestingly, DTN2 over TCP Reno shows far better performance than Hybla for delays equivalent to single-hop ground to GEO transmissions. With a five second delay, DTN2 over Reno outperforms Hybla by an order of magnitude. Running DTN2 over Hybla showed no significant improvement than running it over Reno.

The superiority of emulation to simulation is shown as the upper limits to achieving a TCP connection in emulation are often very different to those reported by simulation software. Especially noteworthy are the TCP SYN re-transmission timings for long delays that are neither reflected in the simulation software used nor the assumptions made by some researchers. Current developments with simulation software where ‘husk’ machines will model real devices in a far more accurate way might bridge the gap between simulation and emulation. That is not to discredit current simulation software that is not designed to deal with the excessively long delays encountered in DTN.

Linux, and to far lesser extent Microsoft Windows, allow changes to be made to TCP’s retransmissions and timings. These alterations do make a three-way handshake possible over
extremely long delays. Nothing, however, can be done to resolve the problem that this process still takes 1.5 RTTs that could instead be used for transmitting data.

Experiments with the two DTN implementations showed them to be robust and readily usable by anyone with an interest in networking and with the ability to compile source code and experiment with configuration files.

It was a little surprising how poorly DTN2 performed when run over UDP, unlike ION with which it was quite possible to emulate reliable interplanetary data transmission. This was primarily down to bundles bigger than the underlying message transfer unit failing when the UDP convergence layer was used for long delays. The same issue prevented ION sending to DTN2 where the DTN2 daemon crashed if sent bundles bigger than the link MTU. ION had no such problems running over UDP although its LTP receiving buffer could be overwhelmed with data when high data rates at low latencies were tested.

TCP Hybla throughput initially outperformed ION over UDP. As delays were increased ION continued to perform far past the point that Hybla stopped working and was still sending data with a 4000s delay.

Using ION, running LTP over UDP, in a three-node network with the middle node acting as a DTN router was highly successful. The Channel Emulator was used to create one-way-light-times matching a Mars, via the Moon, to Earth transmission. Bit error rates mimicked those found for deep-space transmissions and LTP’s report segments distinguished good data from corrupted data. When FTP over TCP was tried over the same delays it failed totally even after TCP and FTP were configured as much as possible to deal with delay and timeouts.

In summary, using a virtualised setting provides a very useful test environment for both TCP and DTN software. It shows advantages over simulation without the costs, complications and difficulties with repeatability of results that are found using a number of physical hosts.

6.1 Project review

The overall aims of the project were successfully achieved. It was originally intended to use a third DTN implementation (a Java based version of LTP) but this was not possible.
Fortunately, LTP became available in later versions of ION and has allowed careful investigation of this protocol.

The nuances of TCP behaviour are extremely complicated when examined in detail and it is necessary to restrict changes to only those areas that relate to delay. Otherwise, it is all too possible to fall into the “hidden variables” problems that was noted with simulation software in Section 3.1.

**6.2 Future research**

DTN is an evolving area and both new and updated DTN implementations are on their way. The much-awaited DTN3, future versions of ION, and DTN implementations ported to Android and Symbian platforms are being developed. These are ripe for further research.

Emulation spanning both the physical and virtual environments is becoming possible with perhaps tests starting and finishing with virtual machines but with the centre section using DTNBone being done. This would provide ways of investigating delays at network layers below IP that are more difficult to emulate cheaply.

Research into more complicated examples of emulated routing and the effects of poor time synchronisation are also possible. It was noted, but not explored further, that pausing and resuming virtual hosts interfered negatively with DTN2 performance. This situation could be relevant to awakening solar powered sensor nodes that hibernate during the night.
References


Burleigh, S. (2008a) In *SpaceOps 2008 Conference* Heidelberg, Germany


Caini, C., Firrincieli, R., Lacamera, D., de Cola, T., Marchese, M., Marcondes, C.,


Hemminger, S. (2005 ) 'Network Emulation with NetEm', *Linux Conf Australia* 2005


Computer Sciences, Social-Informatics and Telecommunications Engineering), Rome, Italy, pp. 1-10.


Index

Apple iMac, 32, 35
ARP, 32
Bluetooth, 14, 59
bundle primary header, 6
Bundle Protocol, 2, 5, 13, 21, 22, 51
CCSDS, 5, 16
CGR, 25
CL, 10, 52, 57
congestion avoidance, 15, 16, 17, 18, 46, 90
convergence layer, 10, 11, 22, 50, 52, 56, 57, 59
custody transfer, 7, 8, 51
DARPA, 20, 22
DTN, 1, 11, 20, 21, 22, 23, 24, 25, 26, 28
DTN Bone, 23
DTNRG, 24
end-of-block
EOB, 11
EOB, 11, 12
Ethernet, 5, 32, 35, 40, 58, 59, 92
FLUTE, 10
geostationary earth orbits, 15
Hermes, 23
Hybla, 17, 18, 35, 36, 45, 47, 49, 50, 54, 55
Internet, 1, 2, 3, 4, 13, 14, 15, 16, 20, 21, 22, 23, 24, 86, 92
ION, 1, 13, 25, 27, 50, 51, 52, 53, 54, 55, 59, 60
IP, 5, 9, 15, 16, 18, 20, 21, 22, 24, 26, 31, 58, 86, 92
ISO, 16, 31
Licklider Transmission Protocol, 10, 13, 14, 21
Linux, 2, 18, 29, 30, 31, 35, 36, 45, 50
LTP, 1, 4, 5, 7, 10, 11, 12, 13, 14, 16, 20, 23, 29, 37, 52, 53, 54, 55, 56, 60, 61, 64, 65, 66, 67, 69, 71, 88
LTP-T, 13, 29
mobile ad hoc networks, 25
84
MULEs, 4, 23

NASA, 1, 20, 22, 27, 31, 35

netem, 29, 30, 31, 32

one-way light time

   OWLT, 62

OSI, 16

Peach, 17, 18

PRoPHET, 4, 25

return receipt, 9

RFCs, 13, 20, 28

RTT, 15, 17, 18, 30, 39, 40, 41, 46, 54

Saratoga, 4, 22

Security, 26

Sensor Network, 23

slow start, 15, 16, 17, 39, 45, 46, 47, 91

SSH, 32

store-and-forward messaging, 4

Sudden Start algorithm, 18

TCP, 1, 2, 9, 14, 15, 16, 17, 18, 19, 20, 21, 22, 26, 29, 30, 35, 36, 38, 39, 40, 41, 42, 45, 46, 47, 48, 49, 50, 52, 54, 55, 56, 57, 60, 86, 90, 91

time synchronisation, 26

transport-layer, 9

UDP, 9, 14, 19, 23, 52, 54, 57, 58, 60

Universal Record Identifiers, 9

USB, 32

VMware Fusion, 32, 35

VMware Workstation, 32, 35

Water Framework Directive, 23

Windows XP, 32, 35
Appendix A – Extended Abstract

Virtually Delayed: An investigation into delay tolerant networks and their emulation in a virtual environment

Paul Tomlinson

Extended Abstract of Open University MSc Dissertation Submitted 28 February 2010

Introduction

This work was motivated by developments in delay (or disruption) tolerant networking (DTN). The current Internet depends primarily on the Transmission Control Protocol (TCP) and Internet Protocol (IP). IP deals with routing while TCP provides a reliable end-to-end connection. Importantly, TCP only works satisfactorily if there is a reasonably reliable connection, with limited delay, between any two points on a link. However, in many situations such as deep space communications, or low power devices participating in sensor networks, there is neither a reliable nor fast link. DTN is designed to overcome these problems using a variety of techniques, such as chopping the data into bundles and using a store-and-forward procedure to move the bundles towards their destination.

The relationship between DTN protocols such as the Bundle Protocol and Licklider Transmission Protocol, and the ‘traditional’ Internetworking protocols, is shown below (from ‘A Bundle of Problems’ (Wood, 2009)).

<table>
<thead>
<tr>
<th>Data-link: Ethernet, Frame Relay, etc.</th>
<th>TCP (direct over UDP, not yet agreed)</th>
<th>other e.g. LuriDTN over FLUTE</th>
<th>Saratoga</th>
<th>Licklider (LTP)</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Internet Protocol</td>
<td>UDP User Datagram Protocol</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Direct convergence layer adapter</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

---

86
All the DTN software is effectively open-source making experimentation relatively easy. Some prior research has been done on DTN using simulation software and emulation on physical hosts. This dissertation has focussed on constructing an entirely virtual test environment, including a delay/disruption emulator, which could be easily modified and extended.

The experiments have focussed on showing the breakdown of TCP, as delays and errors increase, and how the various implementations of DTN offer better reliability and performance.

The research also addresses whether virtualisation offers a valid model for this sort of emulation and suggests possibilities for further research.

Results

An easily reconfigurable virtual emulator was designed that could run successfully on either a Windows or Apple physical host. The virtual Linux nodes (which had two DTN implementations installed on them) communicated through a delay/disruption emulator that was obtained courtesy of NASA. An example two-node network is shown below.
To prove the suitability of DTN the breakdown of TCP, both theoretically and through emulation, was investigated as delays were increased. In a series of emulations TCP Reno, the default variant of TCP found in many Linux installations, was compared to TCP Hybla which is designed for satellite and wireless usage. Except at the smallest delay Hybla outperformed Reno consistently. Using standard settings, the last successful transfer of data was 253bps with an 80 second delay for Reno and 776 bps with a 140 second delay for Hybla. The slow start thresholds and congestion windows of these two TCP variants were also shown over no-delay, GEO delay and lunar delays scenarios. By altering TCP settings in the Linux kernel, a three-way handshake could be achieved at delays of 435 seconds but data transfer was impossible.

Two implementations of DTN (DTN2 and ION) were then examined both against themselves and TCP. Both performed well over TCP. DTN2 performed poorly over UDP. However, ION over UDP proved quite capable of transmitting data, albeit at a data rate of around 100bps, at a 4000 second delay.

Finally, the virtualised environment was used to set up a three-node network with the nodes representing devices on Mars, the Moon and Earth. Accurate delays and error rates were added. In this emulation, ION using LTP over UDP proved successful at transmitting data from Mars via a router on the Moon to the Earth. The one way light time was 345 seconds for the Mars/Moon leg with a further 1.29 seconds for the Moon/Earth hop. On average it took 694 seconds for the sender to receive acknowledgment of an 80KB file.

**Analysis**

The superiority of emulation to simulation was shown as the upper limits to achieving a TCP connection in emulation are often very different to those reported by simulation software. Especially noteworthy were the TCP SYN re-transmission timings for long delays that are neither reflected in the simulation software used nor the assumptions made by some researchers.

Whilst increasing the number of TCP’s SYN retransmissions makes achieving a three-way handshake possible over extreme delays this does not mean it can still transmit data even when timing and retry values are adjusted.

It was surprising how poorly DTN2 performed when run over UDP, unlike ION with which it was quite possible to emulate reliable interplanetary data transmission. This was found to be primarily down to bundles being bigger than the underlying network layer’s ability to cope with them.

In the Mars-Moon-Earth emulation with ION, using LTP over UDP, flow diagrams were obtained showing the correct receipt and acknowledgement of segments. When LTP encountered corrupted data it identified this correctly asked for retransmission. Even at very high error rates transmission was possible.
Discussion

A virtualised environment has a number of advantages. These include; the ability to clone more virtual hosts as necessarily, reversion to previous snapshots in the event of problems and easy network reconfiguration. Up to five virtual machines (including the Channel Emulator) were run but this did require running four of them without the X graphical interface to reduce the memory footprint. The cost and complexity of running five physical machines would have been far greater.

The virtualised environment provided a more reliable evaluation of TCP and DTN than some simulation software which is both complex and may perform allowable but unrealistic actions such as simply doubling TCP’s retransmission time-out until a three-way handshake eventually becomes possible.

No differences were found between the results of experiments done on Microsoft Windows or Apple iMac hosts and TCP behaviour was found to be comparable to that reported on physical hosts.
Appendix B: Test scripts

This script was used to perform tests on congestion avoidance for various versions of TCP. It is a modified version of a script to be found at http://www.cs.virginia.edu/~sdb7e/os_project/assignment.html

#!/bin/sh

# Execute a series of TCP tests using two different congestion algorithms.
# The first command line argument will form the file name suffix used
# to identify the test run.
# A receiving instance of iperf -s should be running prior to starting
# this script. This script should be run as root.

# We can obtain more detailed information about TCP performance using tcpprobe
sudo modprobe tcp_probes port=5001

# TCP is sensitive to a few other settings:
sudo sysctl -w net.ipv4.tcp_no_metrics_save=1
# Do not remember past metrics
sudo sysctl -w net.ipv4.tcp Moderate_rcvbuf=1
# Don't cache ssthresh from previous connection

sudo sysctl -w net.ipv4.tcp_wmem="4096 16384 10000000"
# TCP send buffer first value is minimum TCP send buffer space available for
# Single TCP socket. Second value is default buffer space allowed for a single TCP
# socket to use. The third value tells the kernel the maximum TCP send buffer
# space.

sudo sysctl -w net.ipv4.tcp_rmem="4096 87380 10000000"
#

if [ $# -ne 1 ]; then
  echo "Usage: caTest testName"
  exit 1
fi

echo "\n*** running test $1 ***\n"

# duration of test in seconds
# pass this to iperf
t=60

# for each congestion algo
for c in reno hybla
  do
    echo "running algo $c for $t seconds"
    # construct name for output files
    o=$c$c1
    # TODO: set congestion control algo
  done
echo "TODO: set congestion control algo"
sudo sysctl -w net.ipv4.tcp_congestion_control=$c

# collect packet level stats using tcpprobe
cat /proc/net/tcpprobe >"$o.prb" &
p=$!

# TODO: run iperf for $t seconds, and redirect the output to disk
echo "TODO: run iperf for $t seconds, and redirect the output to disk"
sudo iperf -t $t -c 10.1.1.1

# capture iperf exit status
s=$?
# Stop recording from tcp
kill $p
wait

# Make sure iperf returned without errors
if [ [ $s -ne 0 ]; then
    rm $o.prb
    rm $o.ipf
    exit 1
fi

# retain sending side packets only. This assumes use of the default port 5001.
grep -v "10.1.1.2:5001 10.1.1.2" $o.prb > $o.tmp
mv $o.tmp $o.prb
done

This script, a slight modification of the one found at

#!/bin/bash
if [[ [ $# -ne 1 ]]; then
    echo "Usage tcpview input-data"
    exit 1
fi

#
# Format:
# Time src:port dst:pnt size sndnxt snduna cwnd ssthresh window
# 1 2 3 4 5 6 7 8 9
(cat <<EOF
set data style linespoints
show timestamp
set title "$1"
set xlabel "time (seconds)"
set ylabel "Segments (cwnd, ssthresh)"
plot "$1" using 1:7 title "snd_cwnd", "$1"
    using 1:(\$8<2147483647 ? 0 : \$8) title "snd_ssthresh"
EOF
) | gnuplot -persist
## Appendix C: Wireshark Capture Using the DTN Dissector

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>35</td>
<td>6.581095</td>
<td>10.1.1.4</td>
<td>10.1.1.1</td>
<td>Bundle</td>
</tr>
</tbody>
</table>

(dt://host4.dtn/ping > dt://host1.dtn/testing)

Frame 35 (880 bytes on wire, 880 bytes captured)

- **Arrival Time:** Aug 31, 2009 22:25:35.309088000
- **Time delta from previous captured frame:** -0.000151000 seconds
- **Time since reference or first frame:** 6.581095000 seconds

Frame Number: 35

- **Frame Length:** 880 bytes
- **Capture Length:** 880 bytes
- **Frame is marked:** False
- **Protocols in frame:** eth:ip:tcp:bundle

**Ethernet**

- **Type:** IP (0x0800)

**Internet Protocol**

- **Version:** 4
- **Header length:** 20 bytes
- **Differentiated Services Field:** 0x00 (DSCP 0x00: Default; ECN: 0x00)
- **Total Length:** 866
- **Identification:** 0xf1da (61914)
- **Flags:** 0x04 (Don’t Fragment)
- **Time to live:** 64
- **Protocol:** TCP (0x06)

**Transmission Control Protocol**

- **Sequence number:** 18826 (relative sequence number)
- **Acknowledgement number:** 2 (relative ack number)
- **Header length:** 32 bytes
- **Flags:** 0x18 (PSH, ACK)

0... .... = Congestion Window Reduced (CWR): Not set
TCP segment data (814 bytes)

Reassembled TCP Segments (19638 bytes): #9(1448), #11(1448), #13(1448), #15(1448), #17(1448), #19(1448), #21(1448), #23(1448), #25(1448), #27(1448), #29(1448), #31(1448), #33(1448), #35(814))

Frame: 9, payload: 0-1447 (1448 bytes)
Frame: 11, payload: 1448-2895 (1448 bytes)
Frame: 13, payload: 2896-4343 (1448 bytes)
Frame: 15, payload: 4344-5791 (1448 bytes)
Frame: 17, payload: 5792-7239 (1448 bytes)
Frame: 19, payload: 7240-8687 (1448 bytes)
Frame: 21, payload: 8688-10135 (1448 bytes)
Frame: 23, payload: 10136-11583 (1448 bytes)
Frame: 25, payload: 11584-13031 (1448 bytes)
Frame: 27, payload: 13032-14479 (1448 bytes)
Frame: 29, payload: 14480-15927 (1448 bytes)
Frame: 31, payload: 15928-17375 (1448 bytes)
Frame: 33, payload: 17376-18823 (1448 bytes)
Frame: 35, payload: 18824-19637 (814 bytes)

DTN TCP Convergence Layer Protocol

TCP Convergence Header
Pkt Type: Data
...11 = TCP Convergence Data Flags: 0x03
...1. = Segment contains start of bundle: True
...1.1 = Segment contains end of Bundle: True
Segment Length: 19634

Bundle Protocol
Primary Bundle Header
Bundle Version: 6
Bundle Processing Control Flags: 0x8110
General Flags
...0 = Bundle is a Fragment: False
...0 = Administrative Record: False
...0... = Do Not Fragment Bundle: False
...0... = Request Custody Transfer: False
...1.... = Destination is Singleton: True
...0 .... = Request Acknowledgement by Application: False
Class of Service Flags
01 -- Priority = Normal
Status Report Request Flags
...0 = Request Reception Report: False
...0... = Request Report of Custody Acceptance: False
...0... = Request Report of Bundle Forwarding: False
...0... = Request Report of Bundle Delivery: False
...0 .... = Request Report of Bundle Deletion: False

Bundle Header Length: 63
Destination Scheme Offset: 0
Destination SSP Offset: 9
Source Scheme Offset: 0
Source SSP Offset: 29
Report Scheme Offset: 0
Report SSP Offset: 29
Custodian Scheme Offset: 0
Custodian SSP Offset: 4
Timestamp: 0x122efc4f [Mon Aug 31 22:25:35 2009]
Timestamp Sequence Number: 1
Lifetime: 300
Dictionary Length: 46
Dictionary
   Destination Scheme: dtn
   Destination: //host1.dtn/testing
   Source Scheme: dtn
   Source: //host4.dtn/ping
   Report Scheme: dtn
   Report: //host4.dtn/ping
   Custodian Scheme: dtn
   Custodian: none
Metadata Block
   Block Type: 19
   Block Flags: 0x1
   Block Length: 2
Payload Header
   Header Type: 1
   Block Processing Control Flags: 0x09
       .......1 = Replicate Block in Every Fragment: True
       ......0. = Transmit Status if Block Can’t be Processed: False
       ....0.. = Delete Bundle if Block Can’t be Processed: False
       .... 1... = Last Block: True
       ...0 .... = Discard Block If Can’t Process: False
       ..0. .... = Block Was Forwarded Without Processing: False
       .0.. .... = Block Contains an EID-reference Field: False
Payload Length: 19557
Appendix D: Available Linux kernel 2.6.24-21 TCP Settings

Values in bold indicate settings changed from the default shown in brackets. More details of
the settings are available at http://fasterdata.es.net/TCP-tuning/ip-sysctl-2.6.txt

net.ipv4.tcp_timestamps = 1
net.ipv4.tcp_window_scaling = 1
net.ipv4.tcp_sack = 1
net.ipv4.tcp_retransCollapse = 1
net.ipv4.tcp_syn_retries = 5
net.ipv4.tcp_synack_retries = 5
net.ipv4.tcp_max_orphans = 4096
net.ipv4.tcp_max_tw_buckets = 180000
net.ipv4.tcp_delayed_ack_time = 7200
net.ipv4.tcp_keepalive_probes = 9
net.ipv4.tcp_keepalive_intvl = 75
net.ipv4.tcp_retries1 = 3
net.ipv4.tcp_retries2 = 15
net.ipv4.tcp_fin_timeout = 60
net.ipv4.tcp_sck_cookies = 0
net.ipv4.tcp_tw_recycle = 0
net.ipv4.tcp_abort_on_overflow = 0
net.ipv4.tcp_stdurg = 0
net.ipv4.tcp_rfc1337 = 0
net.ipv4.tcp_max_syn_backlog = 1024
net.ipv4.tcp_orphan_retries = 0
net.ipv4.tcp_fack = 1
net.ipv4.tcp_reordering = 3
net.ipv4.tcp_ecn = 0
net.ipv4.tcp_dsack = 1
net.ipv4.tcp_mem = 24192 32258 48384
net.ipv4.tcp_wmem = 4096 16384 10000000 (1032256)
net.ipv4.tcp_rmem = 4096 87380 10000000 (1032256)
net.ipv4.tcp_app_win = 31
net.ipv4.tcp_adv_win_scale = 2
net.ipv4.tcp_tw_reuse = 0
net.ipv4.tcp_frto = 2
net.ipv4.tcp_frto_response = 0
net.ipv4.tcp_low_latency = 0
net.ipv4.tcp_no_metrics_save = 1 (0)
net.ipv4.tcp_moderate_rcvbuf = 1
net.ipv4.tcp_tso_win_divisor = 3
net.ipv4.tcp_congestion_control =reno
net.ipv4.tcp_abr = 0
net.ipv4.tcp_mtu_probing = 0
net.ipv4.tcp_base_mss = 512
net.ipv4.tcp_workaround_signed_windows = 0
net.ipv4.tcp_dma_copypbreak = 4096
net.ipv4.tcp_slow_start_after_idle = 1
net.ipv4.tcp_available_congestion_control = *
net.ipv4.tcp_allowed_congestion_control = *
net.ipv4.tcp_max_ssthresh = 0