Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Modern Delay-Tolerant Email

Deng Pan

Stephen Farrell
August 15, 2025

Submitted in partial fulfilment of the requirements for the degree of
M.Sc. Computer Science - Future Networked Systems

Abstract

Reliable communication is a critical challenge in deep-space networks, which are characterized by high latency and frequent disruptions. Adapting fundamental tools like email for these Delay-Tolerant Networking (DTN) environments is therefore essential.

This dissertation evaluates a modern DTN-based email proposal (draft-johnson-dtn-interplanetary-smtp) to assess its feasibility and limitations in a simulated deep-space context. To facilitate this analysis, a Docker-based interplanetary network simulation platform was developed. The platform integrates Postfix, Thunderbird, ION-DTN, and the BPMail gateway to form a complete email stack capable of simulating realistic network delays.

Experimental results show that the proposal performs well when the user and mail server are in the same network region. However, testing a user mobility scenario—where the user accesses their home mail server from a remote network—revealed a architectural flaw. The system's real-time protocols (IMAP/SMTP) bypass the DTN gateway, attempting a direct connection that fails as latency increases, rendering the system inoperable.

To address this, a conceptual extension is proposed. This extension introduces a local agent to decouple direct user interactions from background server synchronization, offering a path forward to resolve the issues.

Acknowledgements

I would like to thank my supervisor, Professor Stephen Farrell. Under his supervision, I was able to successfully complete this dissertation. His inspiration and suggestions regarding the project's direction were essential for the progress of this research.

I am also grateful to my family and friends for their support and encouragement throughout this process.

Contents

Abstract **i**

1 Introduction **1**

1.1 Background 1

1.2 Motivation 2

1.3 Research Question and Objectives 3

1.4 Dissertation Overview 4

2 Literature Review **5**

2.1 Characteristics of Interplanetary Networks 6

2.2 DTN Architecture 7

2.3 Email System Architecture 7

2.3.1 Core Components 7

2.3.2 Key Protocols 9

2.4 Solutions for DTN Email 12

2.4.1 Service Interaction 12

2.4.2 Data Format 13

2.4.3 Dependencies Resolution 13

2.5 The BPMail Gateway 14

2.6 Summary 14

3 System Design **16**

3.1 System Architecture 17

3.2 Simulation Environment and Technology Selection 19
3.2.1 Docker 19
3.2.2 Comparison with Other Schemes 19
3.3 Email System Design 21
3.4 Delay-Tolerant Networking Design 22
3.4.1 Interplanetary Overlay Network (ION) 22
3.4.2 Comparison with Other DTN Protocol Stack 22
3.4.3 BPMail 24
3.5 Network Delay Simulation Design 25
3.6 Summary 25
4 Implementation **26**
4.1 Environment and Dependency Preparation 26
4.2 Containerized Implementation 27
4.2.1 Mail Server Container 27
4.2.2 Network Delay Simulator Container 30
4.2.3 User Client Containers 31
4.3 System Orchestration and Startup 33
4.4 Summary 36
5 Results and Discussion **37**
5.1 Baseline Scenario Review 37
5.2 User Interplanetary Mobility Scenario 38
5.2.1 Scenario Description and Problem Analysis 38
5.2.2 Experimental Results 39
5.3 Discussion and Limitation Analysis 39
5.3.1 Architectural Limitation 39
5.3.2 Implementation Limitation 40
5.4 Discussion of Extension 40
5.5 Summary 42

6 Conclusions and Future Work 43

6.1 Conclusion 43

6.2 Limitations 44

6.3 Future Work 44

List of Figures

1.1 DTN stack 2

2.1 Typicall DTN implementation 7

2.2 Email architecture 8

3.1 System Architecture Diagram 17

4.1 Thunderbird security warning for self-signed certificate 33

5.1 Illustration of the mobility scenario 38

5.2 Conceptual Extension of the On-demand Sync Agent 41

List of Tables

3.1 DTN Stacks and Features 23

3.2 Google Scholar Search Results for Selected DTN Implementations 24

5.1 Email retrieval time under different delays 39

1 | Introduction

1.1 Background

As human spaceflight technology advances, we are entering a new era of exploration, moving from Earth into the cosmos. Ambitious initiatives such as NASA's Artemis program (1) and SpaceX Mars colonization program (2) signal a future where humanity may establish permanent bases on other celestial bodies and even achieve interplanetary colonization. However, realizing this grand vision hinges on the establishment of a stable and reliable interplanetary communication infrastructure.

The traditional Internet Protocol suite (TCP/IP), while exceptionally effective on Earth, is ill-suited for deep-space communication. Interplanetary networks face extreme challenges, including immense communication delays, extremely low bandwidth, and frequent link disruptions. These characteristics render the conventional end-to-end, real-time connection model virtually inoperable. To address these issues, academia and industry have proposed the concept of Delay-Tolerant Networking (DTN) (3)(4). The core principle of DTN is the "store-carry-forward" mechanism, which does not rely on persistent, real-time connections. Instead, it leverages the physical movement of network nodes—such as spacecraft, orbital satellites, or planetary bases—to buffer and transport data, thereby enabling reliable data transmission in extreme environments.

1.2 Motivation

Although DTN is widely recognized as the theoretical foundation for interplanetary communication, its practical application and deployment have lagged. The significant differences between the DTN protocol stack and existing internet infrastructure present a "reinventing the wheel" challenge for developers, requiring them to not only design the application layer but also to reconstruct the underlying network architecture. To overcome this barrier, researchers have made several key efforts:

- Modular Protocol Design: The DTN stack, featuring the Bundle Protocol (BP) at its core, is designed with a modular architecture. This allows it to be flexibly integrated with various underlying transport protocols (e.g., TCP, UDP, or specific radio frequency protocols), thereby reducing development complexity. DTN architecture is given in Figure 1.1.

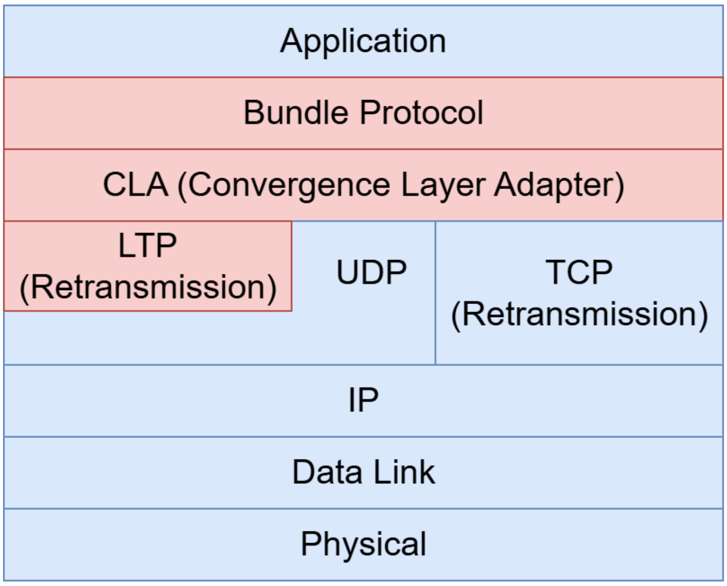


Figure 1.1: DTN stack

- Integration with Existing Networks: Research efforts have focused on enabling the coexistence of DTN and traditional TCP/IP networks. For instance, DTN gateways can facilitate seamless data forwarding between the two network types, allowing DTN-enabled devices to access some existing network services and reducing the cost of rebuilding infrastructure.

Within the DTN application layer, email stands out as one of the most fundamental and critical communication tools. Its implementation in DTN environments has thus received considerable attention. Researchers have aimed to design a solution that leverages the advantages of DTN while maintaining compatibility with the existing email ecosystem. The IETF draft draft-johnson-dtn-interplanetary-smtp (5) represents a significant attempt to create such a solution. This draft proposes a method for operating an SMTP service in a DTN environment, providing a valuable blueprint for translating theoretical concepts into practical, research-oriented applications.

1.3 Research Question and Objectives

This dissertation aims to conduct a systematic simulation and analysis of the Johnson Draft to answer the following core questions: Is the proposed scheme feasible in a simulated deep-space environment? What are its potential design limitations and shortcomings?

To achieve this goal, this dissertation will undertake the following specific objectives:

- Construct a DTN Simulation Environment

  To build a software-based simulation platform capable of modeling the characteristics of deep-space communication, including high latency, low bandwidth, and intermittent link disruptions, thereby providing a foundation for subsequent experiments.

- Implement a DTN Email Service

  To develop a prototype of a delay-tolerant email service based on the Johnson Draft, enabling the functions of message sending, storage, forwarding, and reception within the simulation environment.

- Design and Execute a Validation Scenario

  To design a specific use case, such as an astronaut relocating from a Moon base to a Earth base and attempting to continue using the network for email communication. This scenario will be used to conduct a functional validation and performance assessment of the Johnson Draft.

- Analyze and Evaluate: To collect and analyze key performance metrics, such as average message delivery delay and success rate. Based on these results, this dissertation will conduct an in-depth analysis of the scheme's strengths and weaknesses concerning message ordering, data integrity, security, and resource management, identifying limitations that may arise in practical deployments.

Ultimately, this dissertation will not only validate the feasibility of the Johnson Draft but will also provide valuable insights for the design and optimization of future DTN applications, contributing to the effort of building a true interplanetary communication network.

1.4 Dissertation Overview

This dissertation is organized into six chapters. Chapter 1 provides an introduction to the research background, motivation, and objectives. Chapter 2 presents a literature review of related work in Delay-Tolerant Networking and its applications. Chapter 3 details the system design of the simulation platform. Chapter 4 describes the specific implementation and deployment of the system. Chapter 5 presents and discusses the experimental results. Finally, Chapter 6 concludes the dissertation and suggests directions for future work.

2 Literature Review

This chapter systematically reviews the existing research on DTN and its application to email services. The chapter begins by outlining the fundamental characteristics of interplanetary network environments and analyzing the inherent limitations of the traditional TCP/IP protocol suite in this context, thereby establishing the necessity of the DTN paradigm. It then delves into the core mechanisms and layered architecture of DTN.

Building on this foundation, the focus shifts to a specific application scenario: email. An analysis is presented on how the conversational nature of the Simple Mail Transfer Protocol (SMTP) conflicts with the operational principles of DTN environments. Subsequently, this chapter examines the prevailing gateway-based solutions that integrate DTN with conventional TCP/IP networks. This includes a detailed analysis of key IETF drafts, such as Johnson Draft (5) which defines the service, draft-blanchet-dtn-email-over-bpas (6) which specifies the data encapsulation format, and a corresponding draft for Domain Name System (DNS) (7) that addresses critical service dependencies. Finally, the chapter introduces BPMail (8), a key open-source implementation of this concept, and summarizes the current state of research to identify existing limitations. This provides the theoretical basis for the objectives and contributions of this dissertation.

2.1 Characteristics of Interplanetary Networks

Interplanetary Network environments differ fundamentally from the terrestrial internet, primarily in the following aspects:

1. Long Delays

   Due to the vast distances involved, signal propagation between celestial bodies can introduce noticeable delays. For instance, light-speed communication between Earth and Moon takes about 1.3 seconds each way.

2. Frequent Disruptions

   Communication links are often unavailable for hours or even days due to factors such as celestial occlusion (e.g., planets rotating to block the line of sight) and the continuous orbital motion of satellites.

3. Asymmetric Channel Bandwidth

   The uplink bandwidth from Earth to deep space is typically much higher than the downlink bandwidth from deep space back to Earth.

4. High Bit Error Rates (BER)

   Space radiation and long-distance signal degradation result in significantly higher data transmission error rates compared to terrestrial channels like fiber optics.

The traditional TCP/IP protocol suite was designed for the low-latency, high-stability terrestrial Internet. Its core assumption of a persistent, end-to-end connection is violated in interplanetary environments, leading to its failure. Key limitations include connection establishment failures, the breakdown of “chatty” protocols, and the misinterpretation of disruptions as network congestion (9). Consequently, a fundamentally new network architecture is required.

2.2 DTN Architecture

To address these challenges, Vint Cerf and colleagues proposed Delay-Tolerant Networking (DTN). DTN eliminates the dependency on real-time, end-to-end paths by employing the Bundle Protocol (BP) (10), the "store-carry-forward" mechanism, and the optional Custody Transfer feature, enabling reliable communication in space network environments. Typically implemented shown in Figure 2.1 as an overlay network positioned between the application and transport layers, DTN operates in conjunction with diverse underlying network technologies through Convergence Layer Adapters.

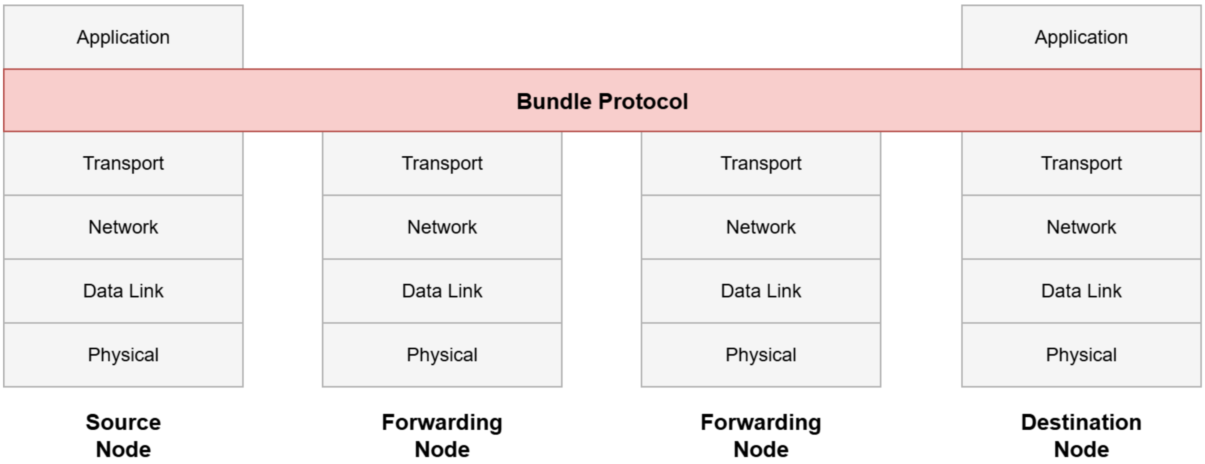


Figure 2.1: Typicall DTN implementation

2.3 Email System Architecture

The traditional email system is a classic example of a distributed, store-and-forward system. It relies on a set of logical components and standardized protocols working in concert to enable global message delivery.

2.3.1 Core Components

As shown in Figure 2.2, a typical email transaction involves the following core components (11):

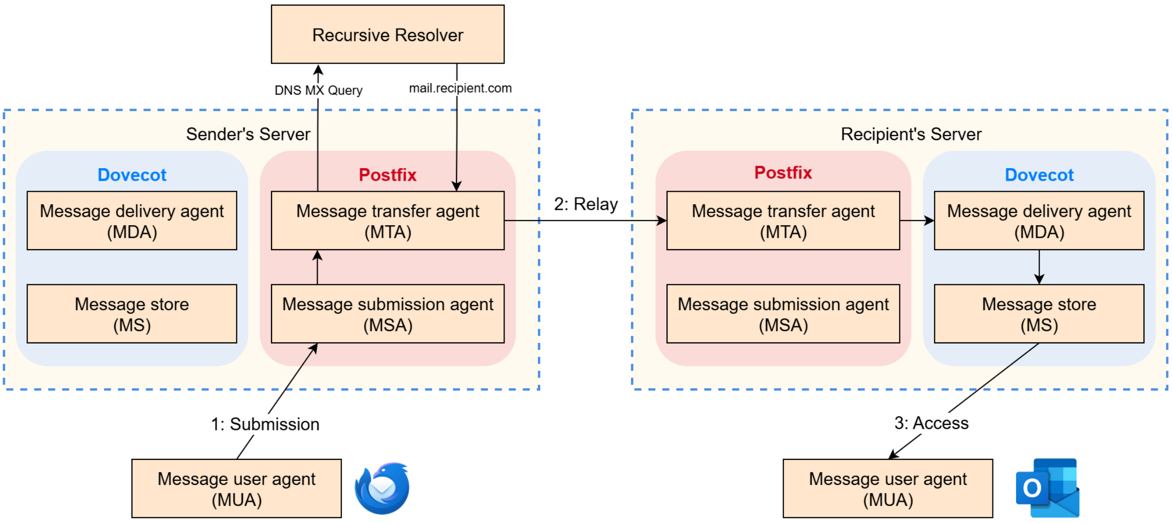


Figure 2.2: Email architecture

1. Mail User Agent (MUA): This is the client software with which the user directly interacts, such as Mozilla Thunderbird or Microsoft Outlook. The MUA is responsible for composing, displaying, and managing emails, and it communicates with mail servers using standard protocols.

2. Mail Submission Agent (MSA): The MUA submits an outgoing email to an MSA. The MSA authenticates the user, performs a preliminary check of the message format, and then passes the email to a Mail Transfer Agent (MTA). Typically, MSA functionality is provided by MTA software (e.g., Postfix) on a dedicated port (port 587).

3. Mail Transfer Agent (MTA): The MTA is the central hub of the email system, acting as a "post office." Examples include Postfix and Sendmail. Its primary responsibility is to relay emails from one server to another. It does this by looking up the Mail Exchange (MX) record in the DNS for the recipient's domain to find the next-hop MTA and then uses SMTP to forward the message.

4. Mail Delivery Agent (MDA): When an email arrives at its final destination MTA, an MDA takes over. The MDA is responsible for placing the email into the appropriate user's mailbox. Software like Dovecot provides robust MDA functionality, including the ability to filter and sort incoming messages.

5. Message Store: This is the physical or logical location on the server where emails are stored, representing the final, static destination in an email's lifecycle. It is a passive component, written to by the MDA and read from by the IMAP/POP3 server in response to MUA requests. The two most common storage formats are (12):

   (a) mbox: Stores all of a user's emails concatenated into a single file. While simple to implement, this format suffers from performance issues with concurrent access and can pose challenges for data recovery.

   (b) Maildir: Creates a separate file for each email, organized within a specific directory structure (e.g., cur, new, tmp). This format avoids file-locking issues, is more robust, and is the preferred choice in modern mail systems.

2.3.2 Key Protocols

Communication between the components described above relies on the following key protocols:

1. SMTP (Simple Mail Transfer Protocol)

   SMTP (13) is the standard protocol for sending and relaying email. It is a TCP-based, text-driven "push" protocol characterized by its highly conversational or interactive nature. A successful mail transfer involves a sequence of strictly ordered command-and-response interactions between client and server, such as: `EHLO/HELO` (greeting and identification), `AUTH` (authentication), `MAIL FROM` (sender's address), `RCPT TO` (recipient address), `DATA` (start of message content), and `QUIT` (termination). After each command, the client must wait for a three-digit status code (e.g., 250 OK) from the server before proceeding. This tightly coupled model requires low latency and stable network connections.

2. POP3 (Post Office Protocol 3) / IMAP (Internet Message Access Protocol)

These protocols are used by an MUA to retrieve and manage emails from a server. POP3 (14) is a simple "pull" protocol that usually downloads all messages to the local client and deletes them from the server. IMAP (15) is more flexible, allowing server-side folder management and message status synchronization across multiple clients, with messages typically remaining stored on the server.

3. TCP (Transmission Control Protocol)

   All of the above application protocols run over TCP, which strongly influence email system behavior and performance. TCP establishes a connection via a three-way handshake and guarantees in-order, lossless delivery using acknowledgments, retransmissions, and congestion control. These mechanisms introduce additional round trips beyond the application's own command/response exchanges, making end-to-end latency directly visible to SMTP, POP3, and IMAP. Specifically: (a) connection setup/teardown incur RTT-dependent delays; (b) per-command turn-taking in SMTP/POP3/IMAP is serialized atop TCP's flow, so each application step often waits at least one RTT; (c) TCP's slow start and congestion control limit initial throughput and reduce the congestion window on loss, which can stall or stretch message transfers; (d) head-of-line blocking within a single TCP stream means that any lost segment pauses delivery of subsequent bytes, amplifying the impact of even modest packet loss when RTT is large. As a result, email protocols—especially SMTP's highly interactive exchange and IMAP's metadata-rich operations—perform best on low-latency, low-loss paths, while high RTT or intermittent loss can significantly degrade responsiveness, prolong sessions, and increase timeout/retry behavior.

4. Anti-Spam and Authentication Mechanisms (via DNS Queries)

   To combat spam and phishing, a suite of protocols works alongside SMTP to verify sender identity. These protocols do not transfer message data themselves but leverage the DNS to publish and check authentication policies.

- DNSBL (Domain Name System blocklist) check occurs the moment a sender's MTA connects, even before significant SMTP commands are exchanged. The recipient's MTA queries a DNS-based, real-time database of IP addresses known for sending spam. If the connecting IP is listed, the MTA can reject the connection outright.
- SPF (Sender Policy Framework) allows a domain to specify which MTAs are authorized to send email on its behalf. The recipient's MTA checks the IP address of the sender's MTA against this authorized list published in the domain's DNS records.
- DKIM (DomainKeys Identified Mail) provides message integrity by adding a digital signature to the email. The sender's MTA signs the message with a private key, and the recipient's MTA uses a corresponding public key from the DNS to verify that the message is authentic and has not been altered in transit.
- DMARC (Domain-based Message Authentication, Reporting, and Conformance) unifies SPF and DKIM. It allows the sender's domain to set a policy that instructs the recipient's MTA on how to handle messages that fail these checks (e.g., to quarantine or reject them) and provides a reporting mechanism for monitoring.

In summary, the traditional email system is a complex ecosystem of components and interactive protocols. Its operation depends on the assumption of a low-latency and continuously available underlying network—namely, the TCP/IP-based Internet. This assumption makes it unsuitable for DTN.

2.4 Solutions for DTN Email

To adapt traditional email systems for DTN environments, a series of collaborative IETF drafts has been proposed. Together, they define a comprehensive solution by addressing three key aspects: service interaction, data format, and dependency resolution.

2.4.1 Service Interaction

Johnson Draft (5) proposes a solution centered on an SMTP gateway. Its core concept is to place a gateway between the DTN and a traditional TCP/IP network. This gateway acts as a protocol translator, converting TCP-based SMTP sessions into BP transmissions for outgoing mail, and reversing the process for incoming mail. This approach enables seamless integration between legacy mail servers and a DTN. The detailed workflow is as follows:

1. *Terminate SMTP Session*

   The gateway presents itself as a standard SMTP server to the local mail server (e.g., Postfix). When the local server attempts to send an email to a destination within the DTN, it establishes a standard SMTP connection with the gateway.

2. *Package Mail Transaction*

   The gateway engages in and completes the entire SMTP conversation, collecting all necessary information, including the sender, recipient(s), and the full message header and body.

3. *Trigger Bundle Encapsulation*

   The gateway then takes the complete mail transaction and triggers its encapsulation into one or more BP bundles, which are then dispatched into the DTN.

This scheme provides excellent backward compatibility with existing email infrastructure. It effectively answers the question of how to interface with the DTN, but it does not specify the precise internal format of the encapsulated bundle.

2.4.2 Data Format

To complement and complete the gateway solution, the Blanchet Draft (6) was introduced. Rather than being a competing proposal, this draft provides a detailed technical specification for how an email should be encapsulated within a bundle, defining a standard format. Its key contributions include:

- *Data Model*

  Specifies how a standard email message (compliant with RFC 5322) should be mapped into a BP bundle.

- *Block Definition*

  Recommends that the entire email, including headers and body, be placed either in the bundle's primary payload block or in a separate payload block.

- *Metadata Mapping*

  Explores the possibility of mapping key email header fields (such as the Message-ID) to bundle metadata to facilitate more efficient processing.

Thus, these two drafts are complementary: the Johnson draft defines *what* the gateway does (interacts with the SMTP world), while the Blanchet draft defines *how* the gateway does it (generates a standardized email bundle). A complete, standards-compliant gateway implementation should adhere to both.

2.4.3 Dependencies Resolution

Email routing is critically dependent on DNS for resolving MX records. Standard DNS, however, is unworkable in a DTN environment. The Johnson DNS Draft (7) addresses this crucial dependency by proposing a DTN-DNS gateway. This gateway intercepts local DNS queries, packages them into bundles for transmission across the

DTN to an authoritative server, and returns the response, also in a bundle. This mechanism is fundamental to ensuring that the DTN email system can correctly route messages.

2.5 The BPMail Gateway

The value of theoretical drafts lies in their practical feasibility. The ASCENT project (8), initiated by California State University, developed an open-source tool called BPMail, which is a concrete implementation of the gateway concept. BPMail is designed to integrate with the Postfix mail system as a custom mail delivery agent.

BPMail's workflow aligns closely with the behavior described in the Johnson draft. It is invoked by Postfix, receives a complete email message, and then uses the library functions of ION-DTN (a leading DTN protocol stack developed by NASA) to encapsulate the email into a bundle and inject it into the local ION node. BPMail provides practical validation for the theoretical proposals, and its encapsulation process represents the core operation that the Blanchet Draft aims to standardize. Therefore, BPMail serves as an ideal tool for researching and simulating a complete DTN email system based on this series of drafts.

2.6 Summary

In summary, the existing literature has established a clear and collaborative technical framework for enabling email communication in DTN environments. The technology roadmap, from the analysis of space network requirements to the development of the DTN architecture and a series of complementary standardization drafts, is well-defined. The existence of practical tools like BPMail further demonstrates the framework's feasibility.

However, a significant gap remains in the current body of research: most literature focuses on architectural design and qualitative feasibility, lacking a systematic analysis of the scheme's limitations within a simulated software environment.

Specifically, the following questions have not been adequately addressed:

- *Architectural Limitation (User Mobility)*

  Current gateway solutions implicitly assume that a user's access point (i.e., their DTN gateway node) is static. The schemes do not consider how email routing and addressing should be dynamically managed when a user travels between celestial bodies or large habitats.

- *Implementation Limitation (Handling of Large Attachments)*

  A theoretical scheme must be realized through software, which may itself have bottlenecks. For example, the performance, resource consumption, and potential failure points of a core tool like BPMail when handling large attachments need to be evaluated to determine the scheme's practical utility.

Therefore, the value of this dissertation lies in filling this gap. By simulating and replicating a complete DTN email system in a highly controlled, containerized environment, this research aims to conduct an analysis of its potential limitations, thereby providing valuable data and insights for its future optimization and deployment.

3 | System Design

This chapter delineates the overall architecture of the simulation platform, the selection of key technologies, and the rationale behind these design choices. A well-designed and scientifically sound simulation platform is the cornerstone for ensuring the validity and credibility of the research findings.

The primary objective of this system design is to create a testbed that can reliably replicate the workflow of the Johnson Draft and flexibly simulate various space network environments. To this end, the design adheres to the following core principles:

- Modularity: The system is partitioned into functionally independent subsystems (e.g., servers, users) to facilitate the configuration and testing of different use cases.
- Reproducibility: Containerization technology is employed to ensure that the entire experimental environment can be precisely and consistently reconstructed, guaranteeing the scientific validity of the results.
- Authenticity: The platform utilizes industry-standard, real-world software (e.g., Postfix, Thunderbird) rather than simplified scripts to simulate application scenarios as authentically as possible.
- Controllability: The network environment, particularly parameters like delay and disruption, must be quantifiable and precisely controllable to provide a solid basis for performance analysis.

3.1 System Architecture

To achieve the aforementioned objectives, this dissertation employs a five-node simulation model built on Docker containers. The overall architecture, which simulates two logically separate network regions—the "Earth Internet" and the "Moon Internet"—is depicted in Figure 3.1. These two regions are interconnected by a simulated DTN link characterized by long latency.

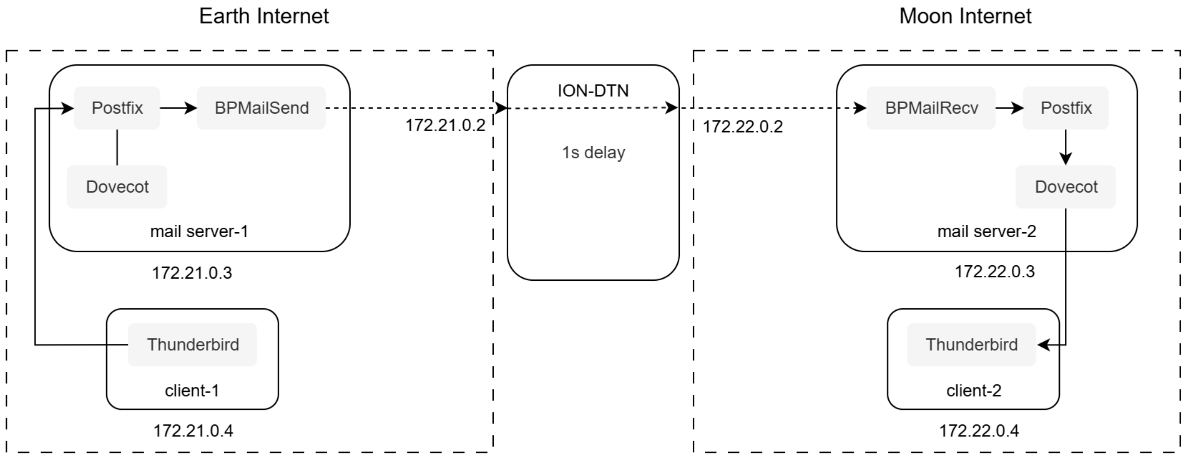


Figure 3.1: System Architecture Diagram

Logical Zones

- Earth Internet

  This zone comprises the `mail-server-1` (IP: 172.21.0.3) and `client-1` (IP: 172.21.0.4) containers, both residing on the 172.21.0.0/24 subnet. They simulate the mail server and user terminal on Earth.

- Moon Internet

  This zone includes the `mail-server-2` (IP: 172.22.0.3) and `client-2` (IP: 172.22.0.4) containers on the 172.22.0.0/24 subnet, simulating the corresponding facilities on the Moon.

Core Component Roles

- Mail Servers (`mail-server-1`, `mail-server-2`): Act as the core of their respective network zones, responsible for email transmission, reception, and storage. They are deployed with a complete mail service suite and the DTN gateway functionality.
- User Terminals (`client-1`, `client-2`): Simulate end-users interacting with their respective local mail servers via standard email client software.
- Delay Node: Acts as a network impairment simulator, bridging the Earth and Moon networks. All inter-regional traffic must traverse this node, enabling application of controlled delays and disruptions.

Data Flow Example (Earth to Moon)

1. The user on `client-1` composes an email in Thunderbird and sends it via SMTP to Postfix on `mail-server-1`.
2. Postfix hands the email to the local BPMailSend component.
3. BPMailSend encapsulates the email into a DTN bundle and sends it to the Delay Node.
4. After the configured delay, the Delay Node forwards the bundle to `mail-server-2`.
5. BPMailRecv on `mail-server-2` decapsulates the bundle and restores the original email.
6. The email is delivered to Postfix, stored by Dovecot, and retrieved by `client-2` via IMAP.

This partitioned architecture clearly simulates a multi-node, cross-network distributed communication scenario, providing a solid foundation for subsequent simulation analysis.

3.2 Simulation Environment and Technology Selection

3.2.1 Docker

Docker was selected as the underlying platform for this simulation experiment. As an open-source application container engine, Docker allows packaging applications and dependencies into a portable container. This choice offers advantages in isolation, reproducibility, and resource efficiency.

Lightweight Virtualization and Resource Efficiency

Unlike traditional virtual machines, Docker containers share the host OS kernel, avoiding the overhead of a full OS per instance. This enables near-instant startup times and supports running multiple nodes concurrently on one physical host (16).

Environmental Isolation and Consistency

Docker provides isolated filesystems, processes, and networking for each container, preventing conflicts such as port clashes or dependency issues—particularly useful when running multiple instances of Postfix or ION.

Guaranteed Scientific Reproducibility

Through Dockerfiles and Docker Compose, the environment is defined as code, ensuring identical experimental setups across different machines (17).

3.2.2 Comparison with Other Schemes

The selection of an appropriate methodology for the simulation platform is a critical step in the system design phase, as it directly influences the reproducibility, scalability, and experimental control of the research. In developing the testbed for this dissertation, three primary approaches were considered: traditional virtualization using virtual machines, a physical hardware deployment, and OS-level virtualization, or containerization. This section provides a comparative analysis of these approaches to justify the selection of Docker as the foundational technology.

Comparison with a Linux Virtual Machine (VM) Scheme

- Resource Overhead

  A significant drawback of a VM-based approach is the substantial resource overhead. Running five independent Linux VMs would require immense memory and disk allocation, as each VM encapsulates a complete guest operating system. In contrast, Docker containers share the host machine's OS kernel, making them exceptionally lightweight (18). This lower resource footprint makes it feasible to conduct complex, multi-node experiments on a standard personal computer or a single server.

- Deployment Efficiency

  The efficiency of deployment and iteration is another key differentiator. Creating and configuring a VM is a time-consuming process, often taking several minutes or longer. Conversely, a Docker container can be launched from a cached image in a matter of seconds. This rapid deployment cycle is particularly advantageous during the iterative development and debugging phases of research, where environments must be frequently rebuilt and adjusted.

Comparison with a Physical Hardware (Raspberry Pi) Scheme

The use of physical hardware, such as multiple Raspberry Pi devices, was employed by researchers in the "SMTP Gatewaying Across Delay Tolerant Networks" project. While this method offers a tangible and intuitive representation of a physical network, a Docker-based approach was deemed more suitable for the objectives of this dissertation.

- Cost and Scalability

  A physical hardware setup involves logistical overhead, including the procurement, deployment, and maintenance of multiple devices and their corresponding network accessories. This approach incurs higher costs and is not easily scalable. The Docker solution, however, allows for the simulation of a virtually unlimited number of nodes on a single host machine at minimal cost, offering superior scalability.

- Control and Consistency

  Physical hardware is susceptible to confounding variables, such as inconsistent network cable quality, power supply fluctuations, and SD card performance. Docker, on the other hand, provides a purely software-defined and highly controlled virtual environment. It allows for the precise management of network topology and parameters, effectively eliminating physical-world variables and making it the superior choice for scientific simulations that require rigorous control over experimental conditions.

In conclusion, while both VMs and physical hardware are viable for certain types of network experimentation, the containerization approach with Docker offered the optimal balance of resource efficiency, deployment speed, scalability, and experimental control. Its ability to provide a consistent, reproducible, and precisely controlled software-defined environment made it the most suitable methodology for this dissertation.

3.3 Email System Design

- Postfix (MTA)

  Chosen for security, stability, and performance. Provides a flexible interface for integrating BPMail via configuration files.

- Dovecot (MDA/IMAP Server)

  High-performance, secure, and integrates seamlessly with Postfix.

- Thunderbird (MUA)

  Real-world client generating realistic emails, improving simulation authenticity.

3.4 Delay-Tolerant Networking Design

3.4.1 Interplanetary Overlay Network (ION)

The ION is an open-source software suite developed by NASA's Jet Propulsion Laboratory (JPL) to implement the DTN architecture. With significant space-flight heritage from numerous deep-space and near-Earth missions, ION is designed for robust operation in challenged communication environments characterized by long delays and frequent link disruptions (19). Architecturally, ION provides a mature implementation of the Bundle Protocol. Key services within the ION suite include a variety of Convergence Layer Adapters (e.g., TCPCL, UDPCL, LTP) to operate over different underlying links, and security extensions via the Bundle Protocol Security (BPSec) specification.

3.4.2 Comparison with Other DTN Protocol Stack

For this dissertation, ION was selected as the DTN implementation after a comparative evaluation against other alternatives. The selection is justified by three primary factors: technical completeness, academic precedent, and practical feasibility.

Technical Completeness

In terms of technical completeness, ION offers the features required by this dissertation. A comparative analysis of leading DTN stacks (Table 3.1), adapted from and extended by Ronny L. Bull et al. (20) based on the Internet Society Interplanetary

Chapter (21) Pilot Projects Working Group's analysis (22), highlights ION's extensive support for the BP core specifications.

Table 3.1: DTN Stacks and Features

Feature/Stack - Subfeature	ION	IONE	HDTN	uD3TN	DTNME	CFS	Unibo	IBR
BPv6	Y	Y	Y	Y	Y	Y	N	Y
BPv6 - TCPCLv3	Y	Y	Y	Y	Y	-	N	Y
BPv6 - UDPCL	Y	Y	Y	Y	Y	-	N	Y
BPv6 - LTPv1	Y	Y	Y	Y	Y	-	N	N
BPv6 - BPSEC	Y	Y	N	N	N	-	N	Y
BPv6 - Custody BPv6	Y	Y	Y	Y	Y	-	N	N
BPv7	Y	Y	Y	Y	Y	Y	Y	N
BPv7 - TCPCLv3	Y	Y	Y	Y	Y	-	Y	N
BPv7 - TCPCLv4	Y	Y	Y	Y	Y	-	N	N
BPv7 - UDPCL	Y	Y	Y	Y	Y	-	N	N
BPv7 - LTPv1	Y	Y	Y	Y	Y	-	Y	N
BPv7 - BPSEC	Y	Y	Y	N	N	-	N	N
BPv7 - Custody (with BIBE)	Y	Y	N	Y	Y	-	N	N
BPv7 - RTP	N	N	Y	N	N	-	N	N
CGR, SABR	Y	Y	Y	N	N	-	Y	Y
CCSDS SPP	N	N	N	Y	N	-	N	N
BSSP	Y	Y	N	N	N	-	N	N
AMS	Y	Y	N	N	N	-	N	N
IPv6 (for CLAs)	N	Y	Y	N	N	-	Y	Y
IPND	Y	Y	N	N	N	-	N	Y
CFDP	Y	Y	N	Y	Y	-	N	N
Language	C	C	C++	C	C++	C	C++	C++

Academic Precedent

Second, ION's position as the established research baseline solidified the choice. A bibliometric analysis was performed using Google Scholar (queried on August 14, 2025) to quantify the academic footprint of various implementations. The Table 3.2 show that approximately 18,300 publications found using the query `"ION" AND "DTN"`. This represents a lead of nearly two orders of magnitude over the next alternatives combined. Adopting ION ensures that our findings can be directly compared against the broadest body of existing work.

Table 3.2: Google Scholar Search Results for Selected DTN Implementations

DTN Implementation	Results
ION	∼18,300
HDTN	∼142
DTN7	∼96
DTNME	∼39
µD3TN	∼30
TinyDTN	1

Practical Feasibility

Finally, practical feasibility was a decisive factor. The existence of an established ecosystem of tools for ION, specifically the open-source bpmail project (8), provides a ready-made gateway for bridging standard email protocols with ION's Bundle Protocol implementation.

In summary, the selection of ION is justified by its demonstrated technical completeness, its unparalleled position, and the practical availability of an ecosystem that directly facilitates my research objectives. These factors, complemented by its strong institutional heritage from NASA and its comprehensive documentation, confirm ION as the most suitable choice for this dissertation.

3.4.3 BPMail

BPMail is an outcome of the SMTP Gatewaying Across Delay Tolerant Networks project. It is designed to function as a gateway between standard email systems and the DTN. For outgoing mail initiated by a server like Postfix, BPMail intercepts the message, encapsulates its data into a BP Bundle, and then injects the bundle into the DTN. Conversely, when a BP Bundle containing an email arrives, BPMail decapsulates it and delivers the restored email to the local mail server. Users can then retrieve the message using standard protocols such as POP3 or IMAP.

3.5 Network Delay Simulation Design

Linux Traffic Control (tc qdisc netem)

- Kernel-level packet manipulation ensures minimal performance overhead (23).
- `netem` supports precise simulation of delay and packet loss, meeting DTN testing needs.
- Can be executed inside Docker containers to modify network behaviour without extra software.

3.6 Summary

This chapter has detailed the system design of the simulation platform. Through a description of the overall architecture and a justification of the technology choices for each subsystem, a solution has been established that is based on Docker containerization and organically integrates a suite of mature, mainstream technologies including Postfix, Dovecot, Thunderbird, ION-DTN, BPMail, and tc. This design demonstrates significant advantages in modularity, reproducibility, authenticity, and controllability, particularly when compared to alternative schemes. This robust design provides a solid foundation for the implementation work detailed in the next chapter and for the data collection in the subsequent results analysis.

4 Implementation

This chapter details the practical process of transforming the architectural blueprint from Chapter 3, “System Design,” into a functional simulation platform. The content covers the preparation of the host environment, the containerized build process for core components, and the orchestration of the entire distributed system using Docker Compose. The source code for this implementation are available on GitHub[1].

4.1 Environment and Dependency Preparation

Host Environment and Core Software

The simulation platform was developed and tested in the following environment:

- Host Operating System: Ubuntu 24.04.2 LTS
- Container Engine: Docker Engine v28.0.4
- Container Orchestration Tool: Docker Compose v2.34.0

Compilation of Dependencies

A key challenge during implementation was the version management of core dependency libraries. It was determined that the successful compilation and stable operation of BPMail required specific minimum versions of its dependencies,

[1] `https://github.com/pandeng65536/Modern-Delay-Tolerant-Email/tree/main/Code`

particularly `gmime`. However, the versions available in the official Ubuntu 24.04 software repositories were outdated and did not meet these requirements.

As direct installation via `apt-get` was not feasible, this dissertation adopted a strategy of manually compiling these dependencies from source. This was achieved by packaging the required source code versions with the project files and automating the compilation and installation process within the mail-server container's `Dockerfile`. The primary dependencies include:

- mbedtls v2.28.8: A lightweight, open-source TLS/SSL stack providing cryptographic support for ION-DTN.
- gmime v3.2.15: A C library for parsing and creating MIME messages, fundamental to BPMail's email processing capabilities.
- c-ares v1.34.4: An asynchronous DNS request library used by network applications such as BPMail.

Although this approach increased the complexity of the build process, it offered two key advantages. First, it completely resolved all version compatibility issues, ensuring the stable operation of the BPMail application. Second, it provided complete and precise control over the software environment, significantly enhancing the reproducibility and reliability of the simulation platform, while also demonstrating the depth and thoroughness of this implementation.

4.2 Containerized Implementation

This section provides a detailed account of the build and configuration process for each independent component of the simulation platform.

4.2.1 Mail Server Container

The mail server container is the most complex component in this simulation, integrating three primary functions: email services (sending and receiving), a DTN node, and the email-DTN gateway.

Dockerfile Analysis

The complete build process for this container is defined in `mail-server/Dockerfile`, which executes the following key stages:

1. *Base Environment*: The official `ubuntu:24.04` image is used as the base.
2. *Service Installation*: Using `apt-get` to install Postfix, Dovecot, and essential compilation and network debugging tools.
3. *Compilation of Dependencies*: As outlined previously, specific versions of `mbedtls`, `gmime`, and `c-ares` are required. The `COPY` directive is used to add their source code to the image, after which they are compiled and installed using their respective build systems (`make`, `./configure`, `cmake`).
4. *Compilation of the ION-DTN*: With `mbedtls` already in place, the ION-DTN stack is compiled. The process uses `autoreconf` to generate configuration scripts, followed by `./configure` with mbedtls support enabled (`-enable-crypto-mbedtls`), and finally `make && make install`. Runtime configuration files for ION-DTN (e.g., the `.rc` files defining node EIDs and contact plans) were pre-configured and included with the source. This allows the startup script to simply run `ionstart` to load the correct configuration. The `ldconfig` command is used to ensure that newly installed shared libraries are discoverable.
5. *Compilation of the BPMail Gateway*: Once dependencies such as `gmime` and `c-ares` are ready, BPMail is compiled using the Meson build system.

```
meson setup build

meson compile -C build

meson install -C build
```

Listing 4.1: Meson Build & Install Stage

This installs the gateway's core programs such as `bpmailsend`.

6. *Installation of Thunderbird Client*: For end-to-end debugging, Thunderbird is installed. Since Ubuntu 24.04 installs Thunderbird via Snap by default (problematic inside containers), a workaround is applied by adding the Mozilla PPA and forcing use of the traditional `.deb` package:

```
RUN add-apt-repository ppa:mozillateam/ppa
RUN cat <<EOF > /etc/apt/preferences.d/mozilla-ppa
Package: *
Pin: release o=LP-PPA-mozillateam
Pin-Priority: 1001
EOF
RUN apt-get update && apt-get install -y thunderbird
```

Listing 4.2: Install Thunderbird from Mozilla PPA

Entrypoint Execution

After installation of all components, the container's `CMD` is set to run the entrypoint script `/mail-server.sh`.

Entrypoint Script (`mail-server.sh`)

This script is central to automated container configuration. On startup, it performs:

1. *Network Configuration*: Reads the environment variable `$GATEWAY` from `docker-compose.yml` and sets it as the container's default gateway. This enables communication with the "outside world" (i.e., the delay node).

2. *Postfix and BPMail Integration*: Uses `$HOSTNAME` to distinguish between `mail-1` and `mail-2`, applying different configurations accordingly.

   - Appends a `pipe` transport service named `bp` to `master.cf`, which invokes bpmailsend for outgoing mail.

- Creates /etc/postfix/transport to map a destination domain to the bp service and specify the destination node's ION EID.

```
MASTER_CF="/etc/postfix/master.cf"
POSTFIX_TRANSPORT_FILE="/etc/postfix/transport"
if [[ "$HOSTNAME" == "mail-1" ]]; then
  echo "bp unix - n n - - pipe" >> "$MASTER_CF"
  echo " flags=ODR user=user argv=/usr/local/bin/bpmailsend -t 1 21 ${
      nexthop}" >> "$MASTER_CF"
  echo "mail-2.a.com bp:ipn:3.129" >> "${POSTFIX_TRANSPORT_FILE}"
  postmap "${POSTFIX_TRANSPORT_FILE}"
fi
```

Listing 4.3: Example snippet for mail-1

3. *Dovecot Configuration*: Uses sed to adjust configuration files for the Maildir format and Postfix LMTP delivery.

4. *Service Startup*: Starts postfix and dovecot, then—based on $HOSTNAME—launches the correct ION node and the BPMail receiving daemon (bpmail2postfix.sh), ensuring service dependencies are respected.

4.2.2 Network Delay Simulator Container

This container, while simple in structure, serves a critical function. Its Dockerfile is based on wbitt/network-multitool, a public image rich with networking utilities, and is configured by copying and executing a startup script.

Network Delay Script (delay-server.sh)

This script executes upon container startup and is responsible for injecting network latency into the simulation platform:

```
#!/bin/bash
```

```
tc qdisc add dev eth0 root netem delay 1000ms
tc qdisc add dev eth1 root netem delay 1000ms
sysctl -w net.ipv4.ip_forward=1
tail -f /dev/null
```

Listing 4.4: delay-server.sh startup script

The script uses the `tc qdisc` command to add a fixed 1000 ms delay to the virtual network interfaces connecting to the "Earth" network (`eth0`) and the "Moon" network (`eth1`). It also enables IP forwarding via `sysctl`, allowing the container to function as a router between the two subnets. To simulate different network delays or disruptions in later experiments, only this script needs to be modified.

4.2.3 User Client Containers

In the `docker-compose.yml` file, the services named `client-1` and `client-2` function as the user terminals. Although they use the same base image as the mail servers, their configuration and purpose are different.

GUI Forwarding

As shown in the following `docker-compose.yml` snippet, the container is configured to forward its GUI display to the host machine's desktop. This is achieved by setting Wayland-related environment variables and mounting the corresponding runtime directory as a volume, facilitating direct graphical interaction with the Thunderbird client.

```
client-1:
  build: ./mail-server
  hostname: client-1
  environment:
    - MOZ_ENABLE_WAYLAND=1
    - WAYLAND_DISPLAY=${WAYLAND_DISPLAY}
    - XDG_RUNTIME_DIR=${XDG_RUNTIME_DIR}
  volumes:
    - ${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY}:${XDG_RUNTIME_DIR}/${WAYLAND_DISPLAY
        }:ro
  # ... other volumes ...
```

Listing 4.5: docker-compose.yml: GUI forwarding for client-1

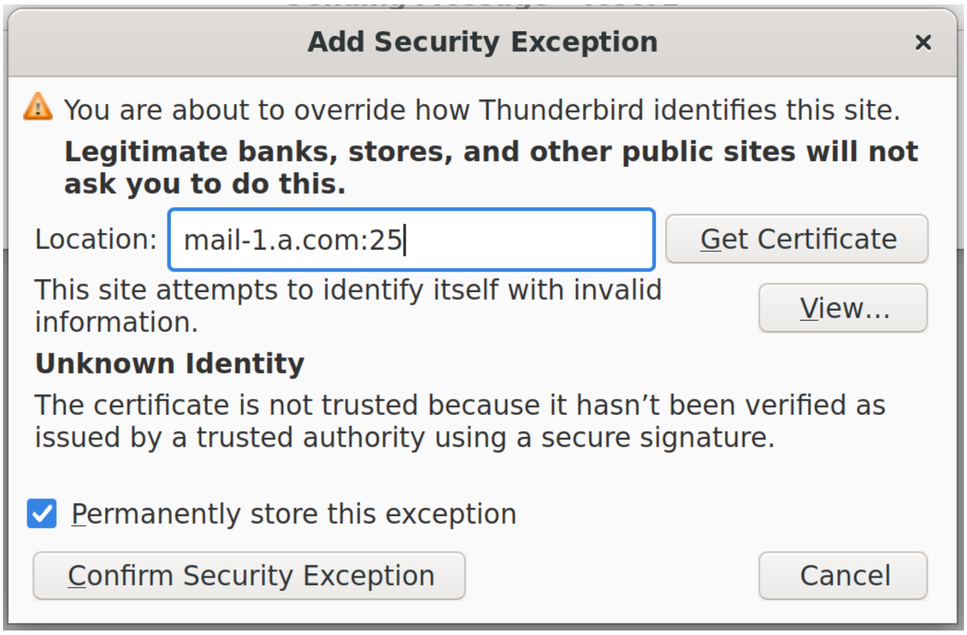
Thunderbird Configuration

The Thunderbird profile was pre-configured on the host machine, and then mounted directly into the container's /root/.thunderbird directory upon startup. This approach obviated the need for tedious manual configuration each time the containers were launched.

The pre-configured profile contains all necessary settings (e.g., display name, email address and password) for the two user accounts, `user@mail-1.a.com` and `user@mail-2.a.com`, including saved security exceptions for the server certificates.

In this simulation environment, I use the default self-signed certificates generated by Postfix and Dovecot during installation. Consequently, when Thunderbird first connected to the server, it was unable to verify their identity, resulting in a security warning as shown in Figure 4.1. After manually accepting and saving the exception, a standard TLS/SSL encrypted channel is established. All subsequent communication between the client and servers is encrypted.

A similar certificate trust issue also exists in the server-to-server communication. Because the two Postfix instances each use their own self-signed certificate, they do not trust each other's identity during SMTP exchanges. Nevertheless, the SMTP connection between the two servers is still encrypted using TLS.



You are about to override how Thunderbird identifies this site.

Legitimate banks, stores, and other public sites will not ask you to do this.

Location: mail-1.a.com:25

This site attempts to identify itself with invalid information.

Unknown Identity

The certificate is not trusted because it hasn't been verified as issued by a trusted authority using a secure signature.

☑ Permanently store this exception

Figure 4.1: Thunderbird security warning for self-signed certificate

4.3 System Orchestration and Startup

The docker-compose.yml file acts as the master controller for the entire simulation platform, defining all components and their interconnections.

Service Definitions

It defines the five core services: delay-server, mail-1, mail-2, client-1, and client-2.

Each service definition specifies its build context, hostname, static IP address, and necessary permissions (cap_add: [NET_ADMIN]). A key design element is the use of environment variables and the startup script to enforce cross-subnet traffic routing.

The GATEWAY variable is injected into the mail-1 and mail-2 services, and the mail-server.sh script within each container reads this variable to dynamically set its default gateway to the IP address of the delay-server. Because the delay-server is the only node connected to both subnets, this design forces all inter-regional IP traffic to be routed through it, ensuring that the network impairments configured with tc are precisely applied to the critical communication link.

Network Definitions

This section defines two independent bridge networks, `mail-1_net` (Earth) and `mail-2_net` (Moon), each with a distinct IP subnet. The service configurations in `docker-compose.yml` connect `mail-1` & `client-1` to `mail-1_net`, and `mail-2` & `client-2` to `mail-2_net`. The `delay-server` is uniquely connected to both networks, establishing it as the mandatory communication relay between Earth and Moon. This ensures that all inter-regional traffic traverses the `delay-server`, allowing the network impairments configured via `tc` to be precisely applied to the critical communication link.

```
networks:
  mail-1_net:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.21.0.0/24
  mail-2_net:
    driver: bridge
    ipam:
      driver: default
      config:
        - subnet: 172.22.0.0/24
```

Listing 4.6: docker-compose.yml: network definitions

System Startup

The startup process is divided into two phases: deploying the background services and launching the foreground client.

System StartupBuild and Start Background Services

First, all background services (`mail-1`, `mail-2`, `delay-server`) and the client container "shell" are built and launched in detached mode using a single command. The `-build` flag ensures that the images are rebuilt from the latest `Dockerfile` prior to launch:

```
docker compose up -d --build
```

Listing 4.7: Building and launching background services

Launch the Graphical Mail Client After the backend services are running, the graphical Thunderbird client is launched manually. First, the host machine's X server must be authorized to accept connections from the container. Then, a shell is opened inside the running client container (`client-1`):

```
# Allow clients to connect to the local X server
xhost +
# Execute a shell inside the running client container
docker compose exec client-1 /bin/bash
```

Listing 4.8: Authorizing X server and opening client container shell

From within the container's shell, the Thunderbird application is started:

```
# Launch the Thunderbird program inside the container
thunderbird
```

Listing 4.9: Launching Thunderbird inside the container

Thanks to the pre-configured GUI forwarding and the mounted profile volume, the graphical Thunderbird interface appears on the host's desktop with all email accounts ready for immediate use.

4.4 Summary

This chapter has provided a detailed account of the practical process of transforming the design blueprint into a working reality. By combining Dockerfiles for component encapsulation, shell scripts for automated configuration, and Docker Compose for system orchestration, a complex, multi-component distributed system was successfully converted into an automated, one-click simulation platform. Key implementation details, from the compilation of third-party dependencies from source to the deep integration of Postfix with BPMail and the precise control of the network environment, have been clearly articulated. At this point, a fully functional and highly controllable experimental environment has been deployed, setting a solid foundation for the system testing and limitation analysis in the next chapter.

5 | Results and Discussion

This chapter presents and analyzes the results from experiments conducted on the simulation platform described previously. The chapter begins by reviewing the system's normal workflow in a standard scenario, which serves as a baseline for analysis. Subsequently, it introduces a more challenging "user interplanetary mobility" scenario. Through quantitative test data, this section reveals the performance limitations inherent in the current Johnson Draft when applied to this scenario. Finally, based on an discussion of these limitations, the chapter offers a conceptual proposal to address the challenges.

5.1 Baseline Scenario Review

In its standard scenario, email transmission and reception rely entirely on the DTN gateway for relay. For instance, when a user on Earth sends an email to a user on the Moon, the message is submitted to the local `mail-1`. There, the `BPMailSend` gateway component encapsulates it into a bundle, which is then transmitted asynchronously and reliably over the DTN link to `mail-2`. Upon arrival, the `BPMailRecv` component decapsulates the bundle and delivers the email to the end user. In this scenario, interactive experience is seamless, as the high-latency interplanetary link is effectively shielded by the DTN gateway and the Bundle Protocol.

5.2 User Interplanetary Mobility Scenario

5.2.1 Scenario Description and Problem Analysis

A core assumption of existing DTN email gateway proposal is that the user's MUA and their home MTA/MDA are located within the same low-latency "local" network. However, this raises a practical and important question: how should users access their email when they travel between celestial bodies?

The key test scenario designed in this dissertation is as follows: a user whose home mailbox is on the Moon (located on `mail-2`) travels to Earth for a mission. From Earth, the user attempts to connect to and synchronize all the emails from their home server on the Moon using a local device (`client-1`).

In this scenario, a problem immediately becomes apparent. As depicted in Figure 5.1, the user's Thunderbird client is configured to connect directly to their home server `mail-2`. Because `mail-2` is located on the distant Moon, this IMAP connection request must traverse the entire long-latency interplanetary link. The DTN gateway is completely bypassed in this process because IMAP is an end-to-end, TCP-based, real-time synchronization protocol that cannot be handled by an gateway like BPMail.

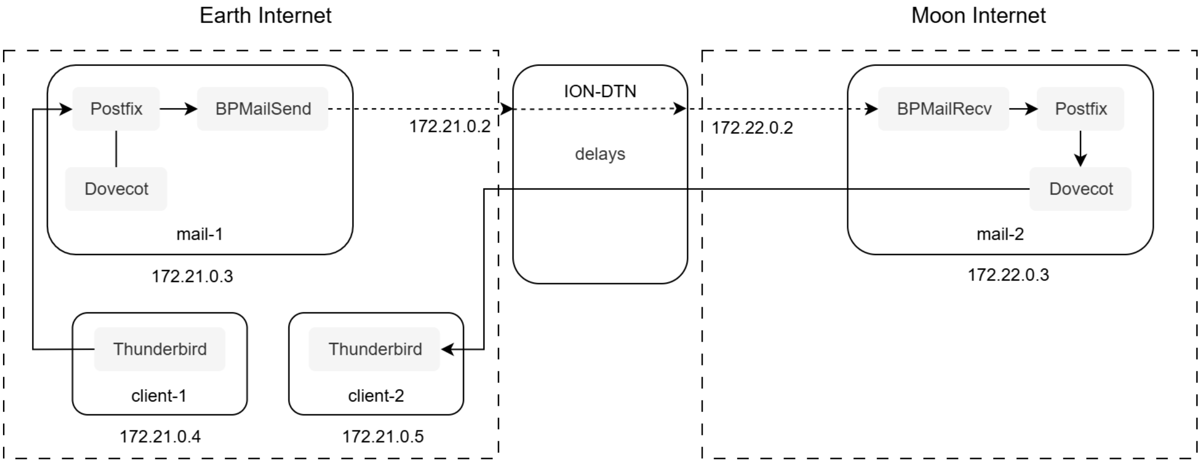


Figure 5.1: Illustration of the mobility scenario

5.2.2 Experimental Results

To quantify the impact of this problem, an experiment was conducted to measure the total time required for a client on Earth to synchronize 1000 simple emails (with the body "Hello") from the server on the Moon under various network delays. The results are presented in Table 5.1.

Table 5.1: Email retrieval time under different delays

Delay (s)	Body	Number of Emails	Retrieval Time (s)
0	"Hello"	1000	19
1	"Hello"	1000	33
2	"Hello"	1000	64
5	"Hello"	1000	153
0	"Hello" with attachment	1	Time out

As is clearly evident from Table 5.1, the email retrieval time deteriorates sharply as the one-way delay increases. In a zero-delay environment (equivalent to a local area network), retrieving the emails takes only 19 seconds. However, when the delay reaches 5 seconds, completing the synchronization requires 153 seconds (over two and a half minutes). At this point, the user experience becomes unacceptable, and the system is rendered effectively unusable.

5.3 Discussion and Limitation Analysis

The experimental results above reveal multiple limitations in the design of the Johnson Draft.

5.3.1 Architectural Limitation

The data provides direct evidence of the proposal's failure to support user interplanetary mobility. The effectiveness of this solution is predicated on decoupling real-time user interaction (Client–Server) from asynchronous server-to-server relay (Server–Server) and applying DTN technology only to the latter. When a user roams, the client–server interaction, which is supposed to occur over a local, low-latency network, is forced to traverse a long-latency interplanetary link. This negates the delay-tolerant advantages of the entire system. The performance collapse is caused by “chatty” protocols like IMAP and SMTP, which require numerous round-trips that are severely penalized by high latency.

5.3.2 Implementation Limitation

During testing, a significant bottleneck was identified in the practical application of the current proposal: the BPMail gateway implementation is incapable of correctly processing emails with large attachments.

When attempting to send an email with a moderately large attachment (e.g., 3 MB) through the gateway, the mail delivery fails. The likely cause of this issue lies within the memory management mechanisms of the BPMail software itself. For instance, the program may attempt to read the entire email content into a fixed-size memory buffer. When the email size exceeds this buffer’s capacity, a buffer overflow or memory allocation failure occurs.

This implementation flaw severely limits the practical utility of the solution. In modern communication, email serves not only as a medium for text but also as a crucial tool for exchanging files and data. An email system that cannot handle a 3 MB attachment is unacceptable for many real-world applications.

5.4 Discussion of Extension

To address the limitations identified above, particularly the challenges related to user mobility, this section proposes a conceptual extension informed by existing research in related fields.

In fact, the problem identified in this research falls under the broader category of “Mobility Management” within the field of DTN. Existing research has generally pursued solutions at two distinct layers:

- **The Network Layer**: This approach aims to provide transparent mobility support through mechanisms such as dynamic routing protocols or directory services that track the changing locations of mobile nodes and route data accordingly.
- **The Application Layer**: This approach focuses on using proxies or caches to confine real-time user–service interactions to the local network, while handling data synchronization asynchronously in the background. Concepts like "Data Mules" are a classic example of this application-layer data ferrying philosophy.

Given the immense complexity of network-layer solutions, an application-layer proxy approach represents a more pragmatic and readily achievable path. To this end, this dissertation proposes the *On-demand Sync Agent* extension, a specific conceptualization of an application-layer solution. Its architecture is depicted in Figure 5.2.

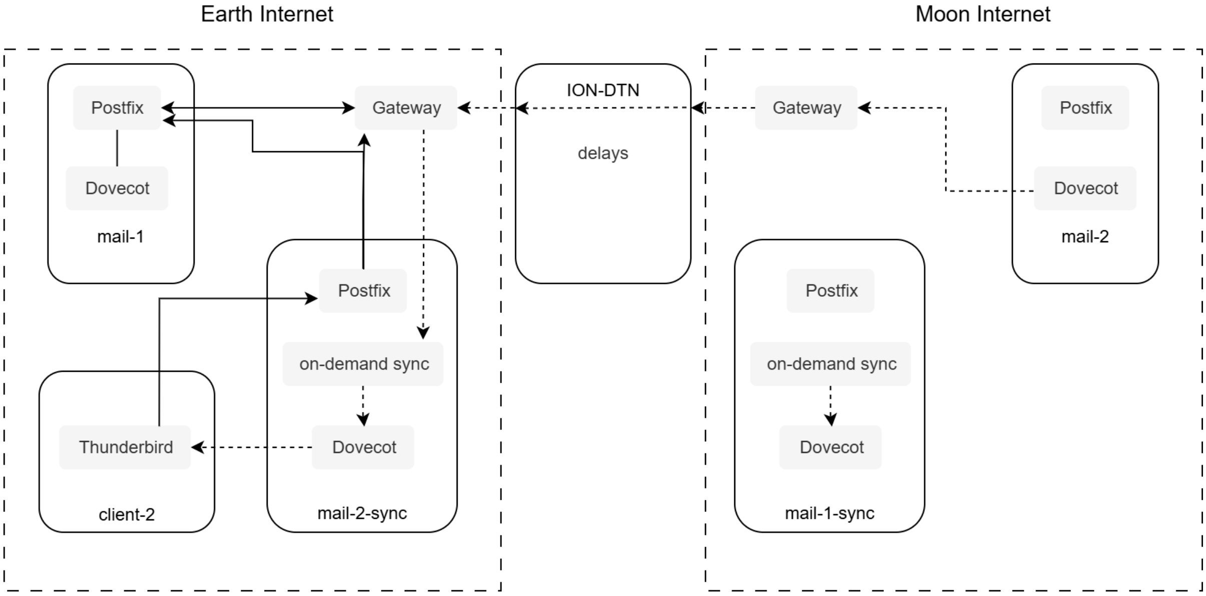


Figure 5.2: Conceptual Extension of the On-demand Sync Agent

The operational logic of this conceptual architecture is as follows:

1. *Introduction of a Local Sync Agent*: A "Sync Agent" service is deployed on each celestial body (e.g., Earth). To the user, this service appears as a fully functional, local mail server.

2. *Localization of User Interaction*: A roaming user (e.g., a user from the Moon who has traveled to Earth) configures their Thunderbird client to connect to this local Sync Agent on Earth, not to their distant home server on the Moon. This ensures a seamless and responsive user experience.
3. *Background DTN Synchronization*: The Sync Agent is responsible for conducting batch, asynchronous email synchronization with the user's home server (e.g., `mail-2` on the Moon) in the background. All data that must cross the interplanetary link is handled by the agent, which encapsulates it into bundles for transmission via the DTN gateway.

This conceptual extension decouples the user's real-time interactions from the background data synchronization by introducing an intermediary proxy layer. It thus holds potential to resolve the performance issues caused by user roaming.

It must be emphasized, however, that this is a preliminary architectural concept intended to inspire future research. It still faces significant challenges — for example, the proposed "Sync Agent" would require email service providers to pre-deploy servers on other planets, which poses considerable implementation difficulties.

5.5 Summary

By designing and testing a key "user interplanetary mobility" scenario, this chapter has identified and validated limitations of the existing DTN email gateway proposal.

First, experimental data demonstrated that direct access to a remote mail server in a roaming scenario leads to severe performance degradation. Second, functional testing revealed that the core BPMail gateway component is unable to handle large attachments.

Following a discussion of these limitations, this chapter proposed an extension based on an *Sync Agent*, and revealed its shortcomings as well.

6 Conclusions and Future Work

6.1 Conclusion

This dissertation has completed a simulation and analysis of a modern delay-tolerant email proposal. In the implementation phase, a simulation platform was established based on Docker containerization, integrating a mail server (Postfix/Dovecot), a client (Thunderbird), a DTN protocol stack (ION-DTN), an email gateway (BPMail), and a network simulator (`tc`).

During the experimental testing phase, a key "user interplanetary mobility" roaming scenario was designed and validated. The test results revealed a architectural flaw in the proposal: when a user roams to a remote network and attempts to access their home mail server, the real-time synchronization protocols (IMAP/SMTP) are forced to operate over a long-latency link. This leads to a sharp performance degradation that renders the system unusable, demonstrating that the Johnson Draft does not support user mobility.

Furthermore, functional testing identified an implementation deficiency in the BPMail gateway component, which was unable to process large attachments, thereby limiting its practical value.

To address the mobility issue, this research proposed a conceptual extension based on an *Sync Agent*. This extension decouples the user's real-time interaction with the email system from background data synchronization across interplanetary links by establishing a local agent, offering a potential solution to this limitation.

6.2 Limitations

Although this research achieved its primary objectives, it is subject to the following limitations:

- *Simplified Network Environment Simulation*: The simulation primarily focused on long delay, the most prominent characteristic of deep-space networks. Other complex network conditions, such as bit error rates, asymmetric bandwidth, and periodic disruptions, were not extensively tested in combination.
- *Lack of DNS and Security Mechanism Simulation*: To focus on the core issues of mail transport and synchronization, the implementation simplified the DNS server configuration. Consequently, it was not possible to experimentally validate the potential failure of DNS-dependent email trust mechanisms (e.g., SPF, DKIM) in the roaming scenario.
- *Unverified Nature of the Proposed Extension*: The *On-demand Sync Agent* extension presented in this dissertation remains at the conceptual design stage. Its effectiveness, and any new issues it might introduce (such as implementation difficulties) have not been verified through prototype implementation.

6.3 Future Work

Based on the findings and limitations of this research, several avenues for future work can be identified:

- *Improving DTN Gateway Support for Large Files*: The most direct and practically valuable line of future work would be to address the shortcomings of BPMail. This could involve debugging and patching its source code or developing a new, more robust mail gateway to ensure reliable handling of large attachments, thereby removing a key barrier to the proposal's practical deployment.
- *Implementation and Validation of the Proposed Solution*: The next logical step is to

transform the *On-demand Sync Agent* extension from a concept into a reality. This would involve developing a prototype system and deploying it on the simulation platform to evaluate its performance in user roaming scenarios.

- *Testing Under More Adverse Network Conditions*: The existing simulation platform could be extended to more complex network environment. By simulating a combination of delay, jitter, packet loss, and disruptions, a more comprehensive stress test could be conducted to evaluate the ultimate performance and robustness of DTN email proposals.

Bibliography

[1] National Aeronautics and Space Administration. 2024 architecture concept review. White paper, December 2024. URL `https://www.nasa.gov/wp-content/uploads/2024/12/acr24-compiled-white-papers.pdf`.

[2] Elon Musk. Making humans a multi-planetary species. *New Space*, 5(2):46–61, 2017. doi: 10.1089/space.2017.29009.emu. URL `https://doi.org/10.1089/space.2017.29009.emu`.

[3] Kevin Fall. A delay-tolerant network architecture for challenged internets. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 27–34, 2003.

[4] Leigh Torgerson, Scott C. Burleigh, Howard Weiss, Adrian J. Hooke, Kevin Fall, Dr. Vinton G. Cerf, Keith Scott, and Robert C. Durst. Delay-Tolerant Networking Architecture. RFC 4838, April 2007. URL `https://www.rfc-editor.org/info/rfc4838`.

[5] Scott M. Johnson. A method for delivery of SMTP messages over Bundle Protocol networks. Internet-Draft draft-johnson-dtn-interplanetary-smtp-01, Internet Engineering Task Force, March 2025. URL `https://datatracker.ietf.org/doc/draft-johnson-dtn-interplanetary-smtp/01/`. Work in Progress.

[6] Marc Blanchet. Encapsulation of Email over Delay-Tolerant Networks(DTN) using the Bundle Protocol. Internet-Draft draft-blanchet-dtn-email-over-bp-05,

Internet Engineering Task Force, March 2025. URL https://datatracker.ietf.org/doc/draft-blanchet-dtn-email-over-bp/05/. Work in Progress.

[7] Scott M. Johnson. An Interplanetary DNS Model. Internet-Draft draft-johnson-dtn-interplanetary-dns-04, Internet Engineering Task Force, March 2025. URL https://datatracker.ietf.org/doc/draft-johnson-dtn-interplanetary-dns/04/. Work in Progress.

[8] Lance Batac, Brian Contreras, Adrian Flores Aquino, Jiahao Li, Sophia Liwag, Nathan Luu, Nicholas Martin, Antonio Ortega Guererro Jr, Anderson Godo Pena Reyes, Ryan Torrez, and Jilei Zou. Bpmail: Utilities for transferring email over bundle protocol, 2025. URL https://github.com/250MHz/bpmail.

[9] Mohanchur Sarkar, KK Shukla, and KS Dasgupta. A survey of transport protocols for deep space communication networks. *International Journal of Computer Applications*, 31(8):25–32, 2011.

[10] Scott Burleigh, Kevin Fall, and Edward J. Birrane. Bundle Protocol Version 7. RFC 9171, January 2022. URL https://www.rfc-editor.org/info/rfc9171.

[11] Dave Crocker. Internet Mail Architecture. RFC 5598, July 2009. URL https://www.rfc-editor.org/info/rfc5598.

[12] D. J. Bernstein. Using maildir format, 1996. URL https://cr.yp.to/proto/maildir.html. Accessed: 14-Aug-2025.

[13] Dr. John C. Klensin. Simple Mail Transfer Protocol. RFC 5321, October 2008. URL https://www.rfc-editor.org/info/rfc5321.

[14] J. Myers and M. Rose. Post office protocol - version 3. RFC 1939, Internet Engineering Task Force (IETF), May 1996. URL https://www.rfc-editor.org/info/rfc1939.

[15] A. Melnikov, N. Cridland, and D. Black. Internet message access protocol (imap) - version 4rev2. RFC 9051, Internet Engineering Task Force (IETF), August 2021. URL https://www.rfc-editor.org/info/rfc9051.

[16] Dirk Merkel et al. Docker: lightweight linux containers for consistent development and deployment. *Linux j*, 239(2):2, 2014.

[17] Carl Boettiger. An introduction to docker for reproducible research. *ACM SIGOPS Operating Systems Review*, 49(1):71–79, 2015.

[18] Wes Felter, Alexandre Ferreira, Ram Rajamony, and Juan Rubio. An updated performance comparison of virtual machines and linux containers. In *2015 IEEE international symposium on performance analysis of systems and software (ISPASS)*, pages 171–172. IEEE, 2015.

[19] Scott Burleigh. Interplanetary overlay network: An implementation of the dtn bundle protocol. 2007.

[20] Ronny L. Bull, Rachel M. Dudukovich, Juan A. Fraire, Nadia Kortas, Robert Kassouf-Short, Aaron Smith, and Ethan Schweinsberg. Network emulation testbed capabilities for prototyping space dtn software and protocols. In *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, pages 1–8, 2024. doi: 10.1109/INFOCOMWKSHPS61880.2024.10620672.

[21] IPNSIG. Internet society interplanetary chapter. https://ipnsig.org/. Accessed: 2023-12-27.

[22] IPNSIG PWG. Bundle protocol implementations. https://ipnsig-pwg.github.io/, 2023. Accessed: 2023-12-27.

[23] Lucas Nussbaum and Olivier Richard. A comparative study of network link emulators. In *Communications and Networking Simulation Symposium (CNS'09)*, 2009.