APPLICATION OF MACHINE LEARNING TECHNIQUES TO DELAY TOLERANT NETWORK ROUTING

by

RACHEL DUDUKOVICH

Submitted in partial fulfillment of the requirements

For the degree of Doctor of Philosophy

Thesis Adviser: Dr. Christos Papachristou

Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

January, 2019

Application of Machine Learning Techniques to Delay Tolerant Network Routing

Case Western Reserve University
Case School of Graduate Studies

We hereby approve the thesis[1] of

RACHEL DUDUKOVICH

for the degree of

Doctor of Philosophy

December 5, 2018

Dr. Christos Papachristou

Committee Chair, Adviser Date
Department of Electrical, Computer, and Systems Engineering

Dr. Pan Li

Committee Member Date
Department of Electrical, Computer, and Systems Engineering

Dr. Ming-Chun Huang

Committee Member Date
Department of Electrical, Computer, and Systems Engineering

Dr. An Wang

Committee Member Date
Department of Computer Science

[1]We certify that written approval has been obtained for any proprietary material contained therein.

Dr. Michael Rabinovich

Committee Member Date
Department of Computer Science

Dr. Daniel Saab

Committee Member Date
Department of Electrical, Computer, and Systems Engineering

Dr. Francis Merat

Committee Member Date
Department of Electrical, Computer, and Systems Engineering

Dr. Daniel Raible

Committee Member Date
NASA GRC

Alan Hylton

Committee Member Date
NASA GRC

Table of Contents

List of Tables

List of Figures

List of Acronyms

API: Application Program Interface

BP: Bundle Protocol

BR: Binary Relevance

CC: Classifier Chain

CCSDS: Consultative Committee for Space Data Systems

CFDP: CCSDS File Delivery Protocol

CGR: Contact Graph Routing

CL: Convergence Layer

CORE: Common Open Research Emulator

CRAWDAD: Community Resource for Archiving Wireless Data At Dartmouth

DHCP: Dynamic Host Configuration Protocol

DSN: Deep Space Network

DTLSR: Delay Tolerant Link State Routing

DTN: Delay Tolerant Network

ECC: Ensemble of Classifier Chains

EID: Endpoint Identifier

EMANE: Extendable Mobile Ad-hoc Network Emulator

ETO-CGR: Earliest Transmission Opportunity Contact Graph Routing

FTP: File Transfer Protocol

GIL: Global Interpreter Lock

GPS: Global Positioning System

HBSD: History Based Scheduling and Drop

HTTP: HyperText Transfer Protocol

IBR: Instituts für Betriebssysteme und Rechnerverbund (Institute of Operating Systems and Computer Networks)

ID3: Iterative Dichotomiser 3

ION: Interplanetary Overlay Network

IP: Internet Protocol

IPND: IP Neighbor Discovery

JNI: Java Native Interface

LATA: Local Access and Transport Area

LEO: Low Earth Orbit

LoWPAN: Low-Power Wireless Personal Area Networks

LTP: Licklider Transmission Protocol

LXC: Linux Container

MAC: Media Access Control

MANET: Mobile Ad-Hoc Network

MAP: Maximum A Posteriori

MSB: Most Significant Bit

NBF: Neighborhood Bloom Filter

NED: Network Description

NEM: Network Emulation Module

NEN: Near Earth Network

NORM: NACK-Oriented Reliable Multicast

NS-2: Network Simulator 2

OLSR: Optimized Link State Routing

ONE: Opportunistic Network Environment

OSI: Open Systems Interconnection

OSPF: Open Shortest Path First

OTA: Over The Air

Pandas: Python Data Analysis

PCN: Planetary Communication Network

POSIX: Portable Operating System Interface

PRoPHET: Probabilistic Routing Protocol using History of Encounters and Transitivity

RAPID: Resource Allocation Protocol for Intentional DTN

RSA: Rivest-Shamir-Adleman

RTEMS: Real-Time Executive for Multiprocessor Systems

SCaN: Space Communications and Navigation

SDNV: Self-Delimiting Numeric Values

SDT3D: Scripted Display Tools 3D

SN: Space Network

SSH: Secure Socket Shell

SSL: Secure Sockets Layer

STK: Systems Tool Kit

TD: Temporal Difference

TDRS: Tracking Data Relay Satellite

UDP: User Datagram Protocol

URI: Uniform Resource Identifier

UTM: Universal Transverse Mercator

VANET: Vehicular Ad-Hoc Network

XML: eXtensible Markup Language

:

Acknowledgements

I would like to express my gratitude to my advisor, Dr. Christos Papachristou for his guidance and support over several years of having been his student. I would like to thank the members of the HiDRA team at NASA Glenn Research Center for their help and support, especially Dr. Daniel Raible, Alan Hylton, Gilbert Clark, Tom Kacpura and Gary Pease, and Dennis Iannicca for help with DTNbone testing and paper reviews. In addition, I would like to thank my branch and division chiefs, Laura Maynard-Nelson and Derrick Cheston for supporting my continuing education.

Abstract

Application of Machine Learning Techniques to Delay Tolerant Network Routing

Abstract

by

RACHEL DUDUKOVICH

This dissertation discusses several machine learning techniques to improve routing in delay tolerant networks (DTNs). These are networks in which there may be long one-way trip times, asymmetric links, high error rates, and deterministic as well as non-deterministic loss of contact between network nodes, such as interplanetary satellite networks, mobile ad hoc networks and wireless sensor networks. This work uses historical network statistics to train a multi-label classifier to predict reliable paths through the network. In addition, a clustering technique is used to predict future mobile node locations. Both of these techniques are used to reduce the consumption of resources such as network bandwidth, memory and data storage that is required by replication routing methods often used in opportunistic DTN environments. Thesis contributions include: an emulation tool chain developed to create a DTN test bed for machine learning, the network and software architecture for a machine learning based routing method, the development and implementation of classification and clustering techniques and performance evaluation in terms of machine learning and routing metrics.

1 Introduction

1.1 Objective and Description

The purpose of this dissertation is to apply machine learning techniques such as classification, clustering and reinforcement learning to address the various challenges throughout the space and deep space networks to improve overall networking performance for future mission applications. As the basis of this work, the delay tolerant networking (DTN) architecture and protocols are considered in order to build upon an established technology which addresses some of the issues which complicate interplanetary networks. The pairing of delay tolerant networking with learning capabilities may be considered a subset of the cognitive networking field of research, with which this dissertation shares many of the same goals. Software and network architecture considerations relative to a particular environment and algorithm are discussed including a survey of popular DTN routing methods. Challenges addressed are the heterogeneous nature of space networks, long round trip times, limited bandwidth, processing and memory resources, potential congestion within local networks and the need to operate within human scheduled constraints while simplifying the necessity of operator supplied input. The problem formulation for the application of machine learning techniques for DTN routing, simulation and emulation, and performance metrics are discussed.

This work focuses on the aspects of scheduling and routing in DTNs. The term scheduling can be taken two ways within this context, on one hand meaning the scheduling of communication between assets such as science mission communication systems, relay satellites and ground stations or the scheduling of messages (packets, bundles, etc.) for transmission and deletion within a single system. Routing focuses on the selection of nodes to form a path from a message source to the destination in the network. These three considerations (scheduling of communication assets, scheduling of message queues within the communication system, and route selection) are all interrelated and impact one another. The scheduling of assets imposes constraints on what nodes are available for communication at a given time. The scheduling and processing of messages (queuing, re-queuing, expiration, deletion, transmission) impact the traffic in the network, the waiting time for new messages to be sent and received, and the buffer availability on communication assets. Routing impacts the utilization of specific nodes, may cause duplicate messages to exist within the network (based on protocol used) and will influence the overall end-to-end delay of message delivery based on paths selected. This work focused primarily on scheduling of message transmission and routing.

1.2 Motivation

In particular, this dissertation focuses on DTNs in the space networking environment, although techniques are discussed which are widely applicable to other types of DTNs. At this time, the NASA system of communication satellites consists of three separate networks, the Near Earth Network (NEN), the Space Network (SN) and the Deep Space Network (DSN), all of which are managed by NASA's SCaN (Space Communications and Navigation) program [3]. While these networks currently function largely as separate entities, it is a future vision for NASA that there will be greater interoperability, flexibility and autonomy among network nodes. Efforts such as cognitive networking and delay tolerant networking are two areas which show great promise to advance these goals. This dissertation aims to apply both of these techniques to the areas of communication scheduling and routing.

1.2.1 Goals of Cognitive Networking for the NASA SCaN Network

This section briefly discusses the high level goals for cognitive networking, as outlined by Ivancic, et al. [3] and how they serve as the inspiration for the research focus of this dissertation.

- Reducing operations cost: Operations costs are related to the amount of human labor required to support systems operations. The more autonomously the network performs as a whole will reduce labor costs. Automated scheduling of network assets, automated detection of failed nodes and replacement by rerouting or redundancy, and nodes which self organize through discovery processes are ways in which cognitive networks could accomplish this goal.

- Provide flexible user services: Intelligent scheduling also impacts this goal. Scheduling methods that can efficiently resolve conflicts improve the flexibility of the system. A scheduling system that can adapt and re-prioritize users and jobs when unexpected changes occur also impacts flexibility.
- Improve system performance: In this case, system performance pertains to throughput of error-free data ("goodput"). Intelligent routing to determine and utilize the network path with shortest overall end-to-end delay based on link reliability, transmission rates, propagation delays, node storage capacity can improve overall network throughput. Protocols which reduce the amount of overhead and duplicate messages sent allow for a greater amount of actual data to be sent within a given transmission period.
- Improve system reliability: Scheduling and routing methods which look for patterns within current and past network performance can sense failures and take a counter measure, allowing neighboring nodes to "fill in" for failed nodes or making predictions and taking proactive measures for when a system failure may occur.
- Increase system asset utilization: Machine learning algorithms can be used to augment or as an alternative to human-created or rule based schedules. This may allow for quicker reactions to unanticipated schedule changes or to produce more efficient schedules.

Within this set of goals it can easily be seen that scheduling and routing have a significant impact on many aspects of improving the overall network operations. Cognitive techniques such as machine learning can provide valuable improvements to many of these areas.

1.3 Contributions

The section summarizes the main contributions of this dissertation.

- Study and comparison of various conventional DTN routing algorithms. In order to first survey the current state of the art of routing techniques in delay tolerant networks, as well as gain familiarity with several popular DTN implementations, the performance of several routing techniques were characterized. These include DTLSR, PRoPHET, flooding which are included as part of the DTN2 reference implementation, as well as the RAPID algorithm implemented as an external routing module for DTN2. The PRoPHET, flooding and epidemic routing implementations as provided within IBR-DTN are also characterized.
- Study and comparison of simulation and emulation environments for DTN testing with 10's of mobile network nodes. Several frequently used simulation and emulation tools including OMNeT++, ONE simulator and CORE/EMANE emulation.
- Develop an architecture, problem formulation and considerations for machine learning techniques relative to the environmental challenges

in various sectors of the space and deep space network, specifically in-situ landed networks, cislunar and similar networks and deep space networks.

- Simulation of the reinforcement learning Q-Routing algorithm and analysis of its applicability to the DTN environment.
- Develop, implement and emulate offline multi-label learning approach to DTN scheduling and routing.
- Develop and emulate a cluster based approach to epidemic routing for DTNs.
- Analyze the results of the classification and clustering routing methods using machine learning metrics.
- Analyze the performance of the routing methods to show a reduction of replicated bundles while maintaining a satisfactory bundle delivery ratio.

1.4 Outline

Chapter 2 covers background information relevant to this work. The current and future interplanetary network as proposed by NASA's SCaN program is discussed, as well as how characteristics of these environments influence networking considerations. Example future mission scenarios and use cases for machine learning are discussed. Next, an overview of delay tolerant networking is given, as well as a brief introduction to machine learning concepts.

Chapter 3, Related Work, covers the main concepts of several well known state-of-the-art DTN routing algorithms. In addition, several closely related works utilizing machine learning techniques for DTN routing are described.

Chapter 4, Approach, discusses the approach to applying machine learning techniques to delay tolerant network routing. A software and network architecture for machine learning based techniques are developed. Several popular machine learning techniques such as Q-Routing, classification, multi-label classification and clustering are implemented as potential improvements for DTN routing.

Chapter 5, Implementation, discusses the details of what software tools were used and how they were extended by this research. DTN implementations that were considered are discussed, as well as how IBR-DTN was modified for machine learning based routing.

Chapter 6, Simulation and Emulation, covers several network simulators and emulators that where explored to create a DTN environment to test the various routing approaches. Node mobility models and scripting as well as libraries used are also described.

Chapter 7, Performance Measurements, discusses the results of the network emulation scenarios. Metrics relative to machine learning as well as performance metrics for routing are considered.

Chapter 8, Conclusion, outlines the work completed in this dissertation as well as suggestions and ideas for follow on work.

2 Background

This chapter covers an introduction to the main topics that comprise the background context for this dissertation. The current and future network concepts of the NASA SCaN (Space Communications and Navigation) program are discussed, as well as the challenges that are unique to each network subsection. Next, an overview of the DTN architecture and relevant protocols are given, as well as how they address the different environmental challenges within the SCaN network. Finally, an overview of the basic concepts of machine learning that were used in this research are given.

2.1 NASA SCaN Networks

The NASA SCaN networks currently consist of the Near Earth Network (NEN), the Space Network (SN) and the Deep Space Network (DSN)[4]. These are currently independent networks, however it is the future vision of SCaN that they will be combined into a single integrated network. The Space Network consists of the set of geosynchronous Tracking Data Relay Satellite (TDRS) and their associated ground stations. The Space Network ground segment consists of three ground stations at the White Sands Complex, the Guam Remote Terminal, and

Network Control Center at Goddard Space Flight Center in Maryland. The Near Earth Network consists of NASA and partner organization ground stations and integration systems that support space communications and tracking for lunar, orbital and suborbital missions. The Deep Space Network consists of communication assets from Geosynchronous Earth Orbit (GEO) to the edge of the solar system as well as the large aperture ground stations which support them. DSN ground stations exist at the Goldstone Deep Space Communication Complex in California, Madrid Deep Space Communication Complex in Madrid, Spain, and the Canberra Deep Space Communication Complex near Canberra, Australia. They are nearly 120 degrees apart which allows for constant communication with space craft as the earth rotates. The SCaN networks provide services between user experiments in space and the user mission ground stations which may consist of NASA, commercial organizations, academia and other space and military organizations. These services include forward data delivery service, return data delivery service, radiometric services from which the position and velocity of the user mission platform are determined, science and calibration services.

The SCaN integrated network will aim to provide customers with the ability to seamlessly use any of the available SCaN assets. The development of a common set of protocols across the various regions of the integrated network will simplify the compatibility requirements between NASA, customers and commercial entities. The need for a high-layer routed and store-and-forward service is among the protocols and techniques to be developed and evaluated to achieve this goal.

In its current form, as well as the future vision of an integrated network there exists several challenges that differ throughout the network based on a communication asset's location. Most notably, there is a difference in one way distance and therefore round trip times from Earth, low Earth orbit (LEO), the lunar surface and Mars. Table 2.1 shows a summary of some of these parameters. Based on these times, it becomes apparent that in some cases conventional protocols such as TCP/IP will perform quite well, whereas at farther distances it will not be suitable. In a similar manner, routing protocols and other cognitive mechanisms may use approaches which require frequent communication, acknowledgments, or updates between nodes, whereas in the case of long haul links from Mars to Earth, this will not be a suitable approach. In addition to these constraints placed on newly developed protocols, it can also be expected that within these differing regions of the network, there will be a variety of lower level protocols as necessitated by the environment.

RTT	**Network Asset Location**	**One Way Distance**
< 0.1 s	Communication between rovers, landers, etc.	Local landed network
0.1 s	Earth to Low Earth Orbit	A few hundred kilometers
0.1 s	Lunar surface to low-lunar orbit	A few hundred kilometers
0.1 s	Martian surface to low-Mars orbit	A few hundred kilometers
0.5 s	Earth surface to LEO via geosynchronous orbit	72,000 kilometers
0.1-2.5 s	Earth surface to lunar transfer orbit	200,000-384,000 kilometers
2.5 s	Earth surface to lunar surface direct	384,000 kilometers
6-45 min	Earth surface to Mars	55,000,000-401,000,000 kilometers

Table 2.1. Approximate Round Trip Times of Network Assets [1]

2.2 Delay Tolerant Networking

The DTN architecture [5] and Bundle Protocol [6] address several of the aforementioned issues. Bundle protocol is an overlay protocol that it creates an additional layer between the application layer and underlying transport, datalink and physical layers within the protocol stack. Bundle protocol is a store and forward based protocol, so data will be stored during a network disruption and then transmitted once connectivity is restored. Data can be transmitted to a destination without a known end-to-end path by relaying to nodes which may eventually come in contact with the destination. In addition, this overlay concept abstracts away the differences in lower level details so that nodes may use a variety of different protocols from the transport to physical layers as appropriate for their particular conditions. Nodes in close proximity together on a planet's surface may use a standard TCP based network. Long haul links such as a Mars relay to earth may use protocols designed for extended distances such as Licklider Transmission Protocol (LTP) [2]. The bundle layer will abstract away these differences, which will be handled by the lower level mechanisms. The bundle layer will be concerned with storage of the data until an appropriate transmission time, custody transfer of the data and routing of the data. Figure 2.1 shows an example of the various protocols that could be used in a delay tolerant network consisting of a landed Martian rover, orbiter satellite, and various components of the ground segment needed to reach a science mission's control center.

A DTN node may typically consists of some user application which generates, sends and receives data using bundle protocol. The bundle layer implements bundle protocol as well as DTN routing. Bundles will be stored until they can be transferred to a suitable neighbor. Lower level protocol layers will determine when the link is available and are interfaced to the bundle layer using a convergence layer adapter specific to the protocols being used. The DTN protocol stack roughly follows the OSI network stack model [5]. The exact number of layers may differ upon implementation, however in general it can be expected that a DTN node would consist of the following layers:

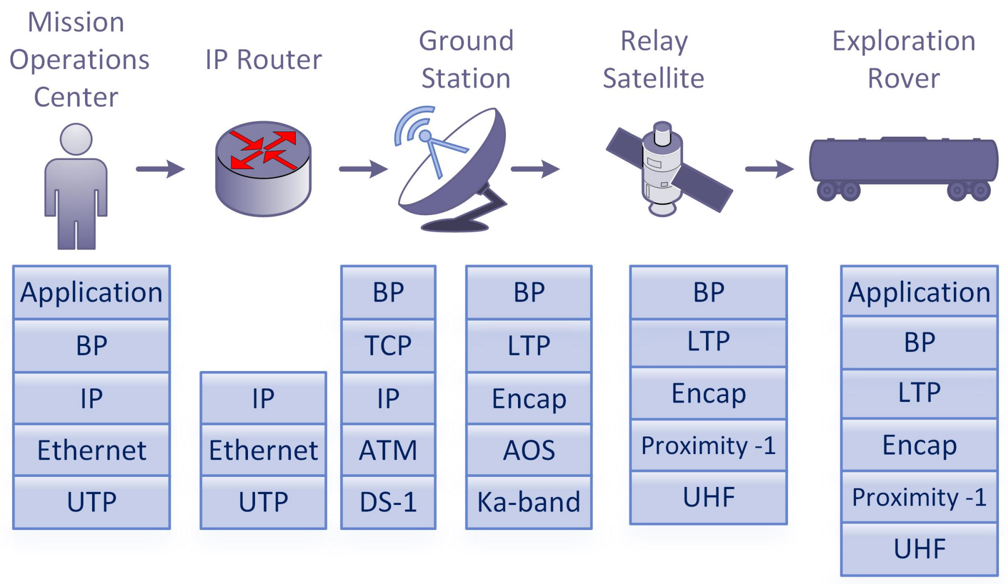


Figure 2.1. Example Mission Operations Center to Deep Space Rover Protocol Stack[2]

- User Applications: The top level applications that may perform any number of tasks, for example reading data from science instruments, accepting user input, or aggregating telemetry to communicate system health to the ground operations.
- Data Processing Protocols: Beneath the user applications, protocols such as CFDP (CCSDS File Delivery Protocol) or BSS (Bundle Streaming Service) may be used. These protocols will take raw user data and break them into bundles and perform other management functionality as outlined in the protocol specification. These protocols are not necessarily required and user applications may interact with the bundle layer using a bundle layer API or a custom developed implementation to send and receive bundles to the bundle protocol agent.

- Bundle Protocol: The bundle protocol is intended to be an overlay protocol which connects a variety of networks using a store, carry and forward methodology. It is here that the delays and disruptions experienced in a space network are mitigated since bundles are stored in long term storage until a contact opportunity is available. Bundle protocol uses the concept of custody transfer among nodes, meaning that a neighboring node must be willing to accept responsibility of a received bundle, ensuring that it will attempt to deliver it to its destination prior to the bundle's expiration and must notify a "report-to" node on the status of the bundle . Custody may be refused if the receiving node's bundle storage is too full, among other reasons. In addition to bundle storage and custody, routing and forwarding decisions are made at the bundle layer.
- Convergence Layer Adapters: The convergence layer adapters are designed to provide an interface between the bundle protocol layer and the transport layer protocols.
- Transport Layer: A DTN node may commonly use several transport protocols including TCP, UDP and LTP. In particular, LTP may be considered specific to delay tolerant networks and is often used over long haul links since its functionality is not as dependent on handshaking procedures that would be impractical over long latency paths in the way that TCP does, yet it provisions for more reliability than UDP by dividing data according to the need for reliable transfer , called red data segments, versus unreliable transfer (green data segments). Red data segments may

be retransmitted in the event of corruption or loss and require status reports and acknowledgments.

The DTN architecture [5] specifies that nodes will be identified by Endpoint Identifiers (EID) which follows the syntax of URIs (Uniform Resource Identifier). The URI begins with a scheme name maintained by IANA. Following the scheme name is a scheme specific string of characters. In DTN, the scheme specific portion is the end point identifier which serves as the DTN address of the node. This specification however, is obviously very general and as such different implementations of the DTN architecture use different EID syntax. The ION implementation uses the scheme "ipn:n.m" where n is the node number and m is the service number. The DTN2 and IBR-DTN implementations use the syntax: "dtn://nodename.dtn/endpointname" In both cases the node name or number identifies a specific node and the service number or endpoint name is used to send and receive data to a particular application. A node may have one name and multiple applications will communicate using a given EID, but each application will listen to and receive from a specific service number or endpoint name. An EID may also refer to a group of nodes in a multicast scenario.

2.2.1 Top Level DTN Flow Chart

Figure 2.2 shows a top level flow chart of one DTN node transmitting data to a receiving node [7]. User applications will generate data such as image or text files which can be sent to a file delivery service such as CFDP (CCSDS File Delivery Protocol) that will segment the files and interact with the bundle protocol layer. User audio and video data may be sent over a streaming service which will also interact with the bundle layer. Once the data has been formatted into

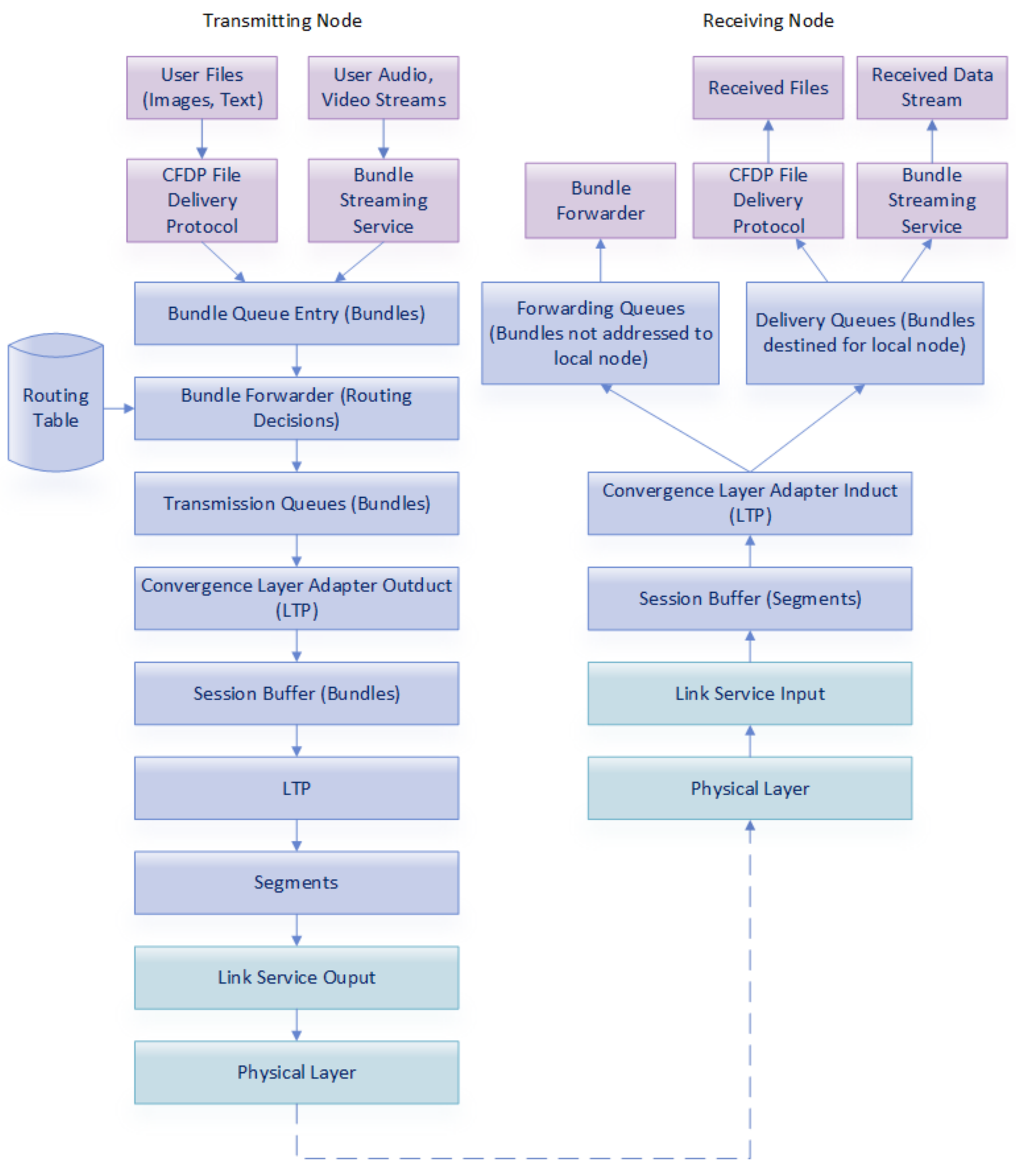

Transmitting Node
Receiving Node
User Files (Images, Text)
User Audio, Video Streams
Received Files
Received Data Stream
CFDP File Delivery Protocol
Bundle Streaming Service
Bundle Forwarder
CFDP File Delivery Protocol
Bundle Streaming Service
Bundle Queue Entry (Bundles)
Forwarding Queues (Bundles not addressed to local node)
Delivery Queues (Bundles destined for local node)
Routing Table
Bundle Forwarder (Routing Decisions)
Transmission Queues (Bundles)
Convergence Layer Adapter Induct (LTP)
Convergence Layer Adapter Outduct (LTP)
Session Buffer (Segments)
Session Buffer (Bundles)
Link Service Input
LTP
Physical Layer
Segments
Link Service Ouput
Physical Layer

Figure 2.2. DTN Top Level Flow Chart

bundles via bundle layer library calls (bundle protocol send and receive functions), they enter the bundle forwarding queue. The bundle forwarder will examine the bundle header destination, custody transfer requests and time-to-live to determine a suitable path to send the data to its destination. The bundle forwarder/router will consult a routing table with information regarding which of its outgoing queues corresponds to a suitable neighboring node.

Depending on the transport layer that corresponds to the selected outgoing queue, a convergence layer adapter must be selected to encapsulate the bundle into the appropriate protocol data unit. In the case of LTP, a session buffer will be used to store the LTP segments while waiting for acknowledgments from the destination node regarding the reception of reliably sent data. Similarly, when data arrives at the receiver, segments enter the LTP session buffer and the appropriate convergence layer adapter interacts with the bundle layer. The new bundles are either sent to a forwarding queue if they are destined for another node and sent to a delivery queue if they are addressed to the local node. They will then be reassembled into application data units and received by the appropriate application at the final destination node.

2.2.2 Bundle Protocol

The format of each bundle consists of at least two blocks, a primary block and a payload block. The first block in a sequence is the primary block which contains information about the bundle source, destination, creation time and other processing information. The payload block contains the actual data that is being sent in the bundle. Figures 2.3 and 2.4 show examples of the bundle primary and payload blocks. Bundle protocol makes use of Self-Delimiting Numeric Values (SDNVs) for much of its processing information. SDNVs consist of a variable number octets. The last octet has its most significant bit set to zero. The most significant bit of all other octets is set to one. The value encoded in the SDNV is the unsigned binary number obtained by concatenating the seven least significant bits of each octet [6]. Each field other than the version can be of variable length since they are using the SDNV format.

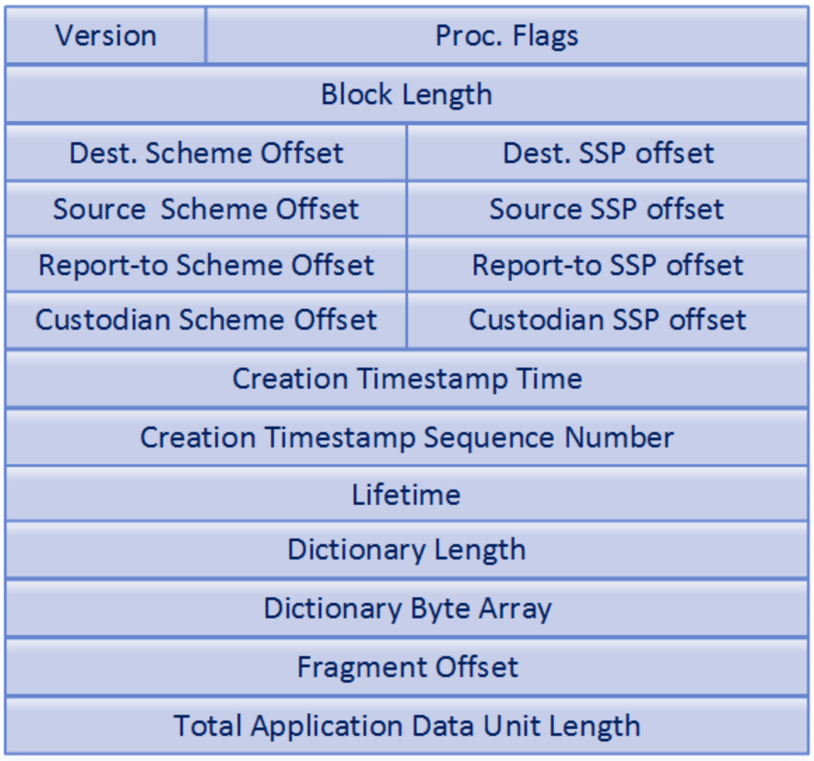


Figure 2.3. Bundle Protocol Primary Block[6]

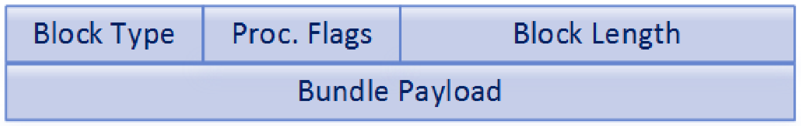


Figure 2.4. Bundle Protocol Payload Block[6]

The steps for bundle transmission outlined in the bundle protocol specification [6] are shown in Figure 2.5. A brief summary of the procedure is as follows:

- The forwarder examines the bundle header to see if custody transfer is requested. This means that the local bundle agent must agree to accepting custody of the bundle
- Transmission of the bundle begins by creating an outbound bundle with retention constraint “dispatch pending” and the current custodian endpoint is set to none.

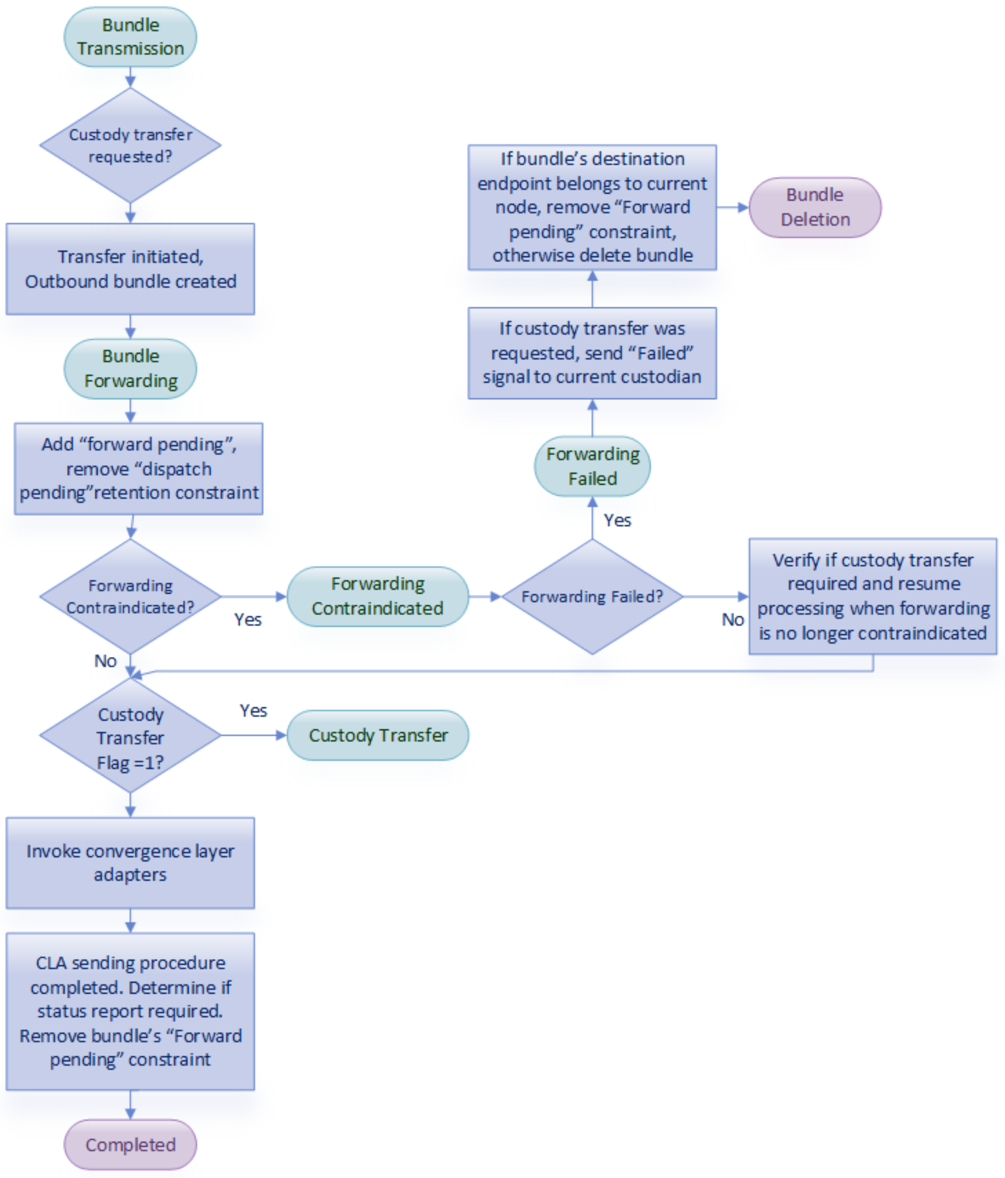

Bundle Transmission
Custody transfer requested?
Transfer initiated, Outbound bundle created
Bundle Forwarding
Add "forward pending", remove "dispatch pending"retention constraint
Forwarding Contraindicated?
Yes
No
Forwarding Contraindicated
Forwarding Failed?
Yes
No
Forwarding Failed
If custody transfer was requested, send "Failed" signal to current custodian
If bundle's destination endpoint belongs to current node, remove "Forward pending" constraint, otherwise delete bundle
Bundle Deletion
Verify if custody transfer required and resume processing when forwarding is no longer contraindicated
Custody Transfer Flag =1?
Yes
Custody Transfer
Invoke convergence layer adapters
CLA sending procedure completed. Determine if status report required. Remove bundle's "Forward pending" constraint
Completed

Figure 2.5. Bundle Protocol Transmission Procedure[6]

- If the bundle is destined for an endpoint to which the node is a member, the bundle is delivered locally. Otherwise, the bundle forwarding procedure is followed.
- To begin bundle forwarding, the "forward pending" retention constraint is added and the "dispatch pending" constraint is removed. A "retention

constraint" is term used in the protocol for essentially flags added to the bundle's state designating a reason for which the bundle must continue to be stored and must not be deleted until all retention constraints have been removed.

- It must be determine if forwarding contraindications exist. If the bundle forwarding agent cannot find any endpoint to send the bundle to which moves the bundle towards its destination, forwarding is not possible. In addition, if an endpoint is found to forward the bundle to, an appropriate convergence layer adapter must exist that will allow the bundle to be sent to the selected endpoint. If either of these conditions exist, the forwarding contraindicated procedure is followed.
- If forwarding was contraindicated, the bundle protocol agent must determine if forwarding has failed, depending on the reason that forwarding was contraindicated. If it has been determined that forwarding is not possible, the forwarding failed procedure is followed. Otherwise, the bundle agent will attempt to forward the bundle again at a later time.
- If forwarding has failed and custody transfer was requested, a "failed" custody signal is sent to the bundle's current custodian, with a reason code corresponding to the forwarding failure reason. If local node is a member of the bundle's destination endpoint, the "forward pending" retention constraint must be removed. The bundle is deleted.
- If custody transfer was requested, the custody transfer procedure is followed.
- The bundle is sent to each convergence layer adapter for each endpoint selected.

- When each convergence layer adapter has completed, a bundle forwarding report is sent to the report-to endpoint id requested in the bundle header and the "forward pending" retention constraint is removed.

Figure 2.6 shows the steps for the bundle reception procedure as outlined in the bundle protocol specification [6]. The steps are summarized as follows:

- The "dispatch pending" retention constraint is added to the bundle.
- If "request reporting of bundle reception" is specified, a bundle reception status report is sent to the bundle's report-to endpoint id.
- If there are any blocks that are intelligible and cannot be processed, a "block unintelligible" status report is sent if required. The bundle is deleted if the block processing flags indicate to do so.
- If bundle custody transfer was requested, and if it is found that this bundle has the same source, timestamp, fragment offset and payload length as an existing bundle, this bundle is redundant. In this case custody transfer redundancy must be handled.
- If the bundle's destination is the local node, the bundle delivery procedure is followed. Otherwise the bundle forwarding procedure is followed as discussed in the bundle transmission procedure.
- When the bundle has reached its destination endpoint if the bundle is a fragment, the application data unit reassembly procedure is followed. If this results in the completion of an application data unit, the bundle can be delivered. Otherwise, the "reassembly pending" retention constraint is added to the bundle and it must wait for the remaining fragments.

- If the registration corresponding to the destination endpoint id is in the active state, the bundle is delivered. If the registration is in the passive state, then delivery failure results.
- If “request reporting of bundle delivery” was indicated, then a bundle delivery report is generated and sent to the bundles report-to endpoint. If custody transfer was indicated, then a “Succeeded” custody signal is sent to the bundle’s current custodian.

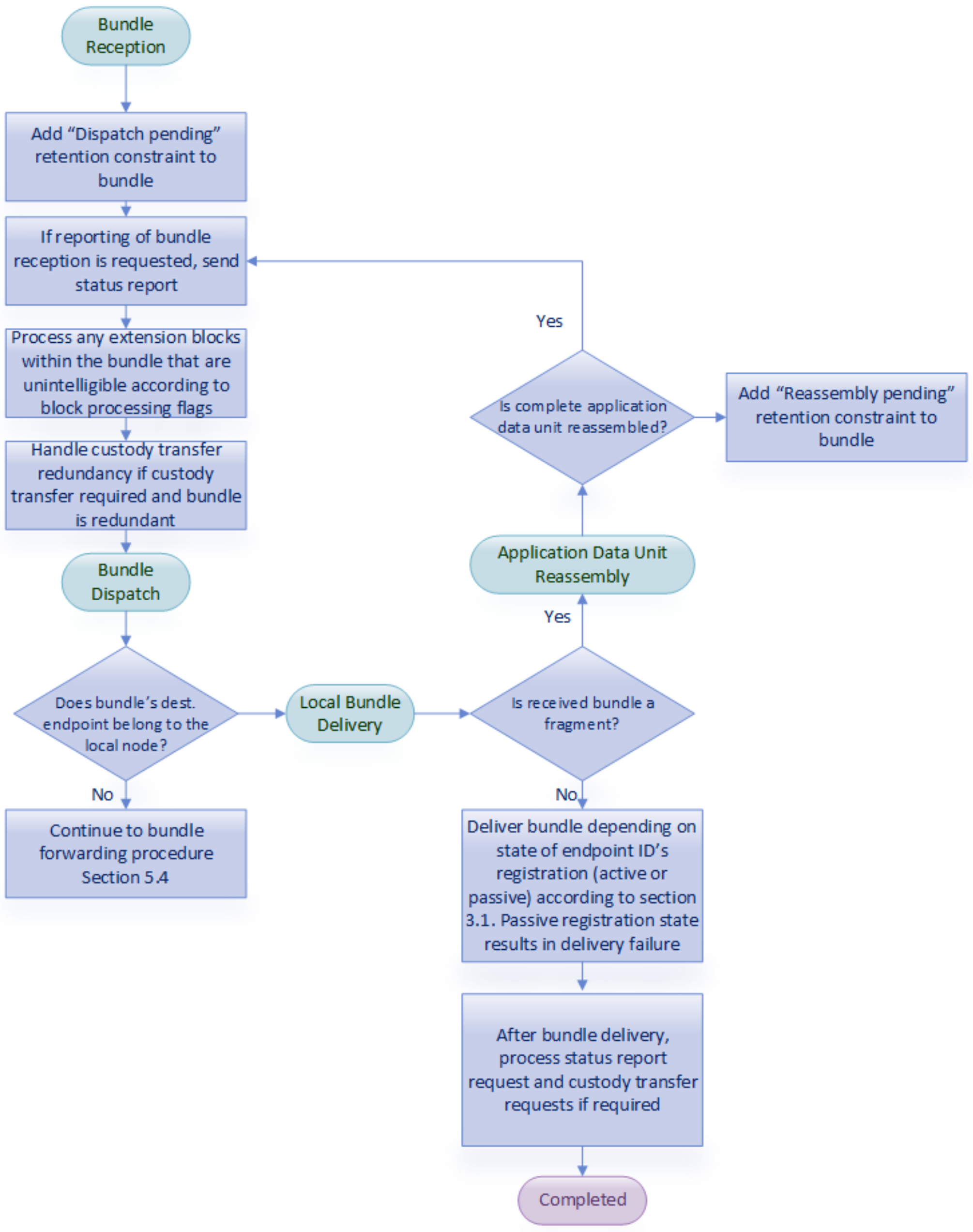

Bundle Reception
Add "Dispatch pending" retention constraint to bundle
If reporting of bundle reception is requested, send status report
Process any extension blocks within the bundle that are unintelligible according to block processing flags
Handle custody transfer redundancy if custody transfer required and bundle is redundant
Bundle Dispatch
Does bundle's dest. endpoint belong to the local node?
No
Continue to bundle forwarding procedure Section 5.4
Local Bundle Delivery
Is received bundle a fragment?
Yes
Application Data Unit Reassembly
Is complete application data unit reassembled?
Yes
Add "Reassembly pending" retention constraint to bundle
No
Deliver bundle depending on state of endpoint ID's registration (active or passive) according to section 3.1. Passive registration state results in delivery failure
After bundle delivery, process status report request and custody transfer requests if required
Completed

Figure 2.6. Bundle Protocol Reception Procedure[6]

2.2.3 DTN IP Neighbor Discovery

In addition to Bundle Protocol, the DTN protocols also define an opportunistic discovery mechanism, IP Neighbor Discovery (IPND)[8]. IPND allows nodes to announce themselves to previously unknown nodes and exchange connectivity information. In this way, it is not necessary to know node addressing and schedules in advance. If two nodes come in contact with one another, they will exchange information using discovery beacons in the form of UDP datagrams.

DTN IP Neighbor Discovery (IPND) is an Internet Draft protocol outlining the mechanism for previously unknown DTN nodes to exchange connectivity information to allow them to begin to communicate with one another [8]. It utilizes discovery beacons to allow nodes to advertise their existence to one another. The beacons are small UDP datagrams, so that any node using an IP-based convergence layer may participate in the discovery process. This is done as part of the Bundle Protocol overlay concept, so that heterogeneous networks may still exchange data using the commonality of the bundle layer and IP underlay.

IPND beacons are sent periodically to neighboring nodes. They may be either unicast or multicast messages. There is no specification for the time interval between periods, as this is quite application specific and may vary greatly depending on the type of nodes participating in the neighborhood discovery. A beacon period field indicating the frequency at which beacons will be sent is considered optional within the specification. Data sent between discovered nodes may substitute as a beacon, and the reception of data from a discovered node suppresses the output of discovery beacons between the two neighbors [8]. Figure 2.7 shows the beacon format. The discovery beacon consists of the following fields:

- Version: An 8-bit field representing the IPND version
- Flags: 8 bits which serve as processing flags. The flags indicate if various optional fields are present. A given bit is set if the related field is present

within the beacon. The flags consist of the following, (listed in order of least-signficant-bit to most significant):

  - Bit 0: Source Endpoint ID (EID) present
  - Bit 1: Service block present
  - Bit 2: Neighborhood Bloom Filter present
  - Bit 3: Beacon Period present

- Beacon Sequence Number: 16-bit field that is incremented each time a beacon is transmitted to a given IP address.
- EID Length: The byte length of the canonical EID contained in the beacon.
- Canonical EID: The canonical EID of the node advertised by the beacon.
- Service Block: An optional listing containing all or some of the following information: convergence layers available at the advertised node, a Neighborhood Bloom Filter, routing information and other implementation specific fields.
- Beacon Period: Optional field containing the period at which the advertised node will send discovery beacons.

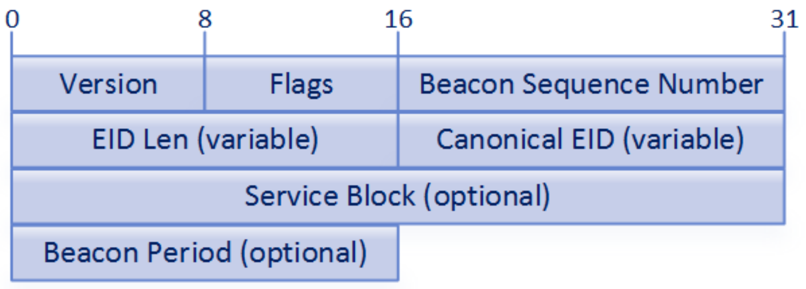


Figure 2.7. Discovery Beacon Format[8]

Due to the nature of DTNs and MANETs (Mobile Ad-Hoc Networks), it is not assumed the links are bidirectional. This is because there very often is a difference in antennae characteristics, transmit power and other factors between wireless nodes. For this reason, it is not assumed that just from receiving a beacon that data may be sent to the neighboring node. IPND uses a Neighborhood Bloom Filter (NBF) to determine the set of neighboring nodes relative to the transmitting node. If the receiving node is contained in the NBF, then it is assumed safe to transmit data to that node as the link is considered bi-directional [8].

Determination of link connectivity is left to implementation specific details and not outlined within the IPND specification. Upon receiving a beacon, a node may determine that the neighbor is available to receive data. Continued receipt of beacons within the beacon period specified or data is used to determine the link state. The actual process of establishing a connection between nodes is dependent on the lower level protocols used. The convergence layer information listed in the beacon service block will provide the information required to make the connection, such as addresses and port numbers on the listening node.

It has been noted that there are several security concerns regarding IPND, as well as DTNs and neighborhood discovery protocols in general [9], [10], [11]. Since nodes participating in discovery advertise their connectivity information, it is apparent that applications with specific security concerns should not use IPND without implementing a security strategy.

2.3 Machine Learning

Machine learning is a somewhat broad term that can encompass techniques from artificial intelligence, statistics, control theory, and data mining among others. Machine learning algorithms are often categorized as supervised or unsupervised. In supervised learning, a large set of data is used to train the learner. The learner develops rules from the dataset in order to classify new instances of the data based on the training set. Data in the training set is labeled and the learner makes predictions which may be correct or incorrect. In unsupervised learning, the goal is to draw inferences about the input data and learn more about associations within the data set by using techniques based on mathematics and statistics. Unsupervised learning doesn't develop a set of rules or decision criteria, instead it is used to learn about the characteristics of a dataset. Yet another subset of machine learning is reinforcement learning. In this case, the learner makes decisions initially in a trial-and-error method. Decisions which result in a positive outcome earn the learner a reward. In this way, the learner determines what are good and bad decisions in a given instance. Each of these methods has its own particular strengths and weakness and can be applied to a network routing problem in different ways.

2.3.1 Reinforcement Learning

Reinforcement learning is a subset of machine learning algorithms in which the learner discovers how to best map situations to a desired outcome by maximizing a numerical reward. The learner is not told what to do, it must learn through a trial and error process to act in a way that maximizes its reward. The reward may not be immediate, several actions may be completed before the final reward is received and previous actions may influence future rewards. Reinforcement learning differs from supervised learning in that there are no examples provided by an external supervisor. In reinforcement learning, the learner must interact with its environment and learn from its own experience [12].

Figure 2.8 shows an example scenario of a reinforcement learning problem. A robot must learn to navigate through a maze environment. The robot has a current state it determines from the environment and will take a specific action. It will receive a reward based on the action and the next state. If the robot makes a movement that leads it to an open path in the next state, it increases a numerical reward (+10 for example). If the movement leads to being blocked by a wall, it will not receive a reward.

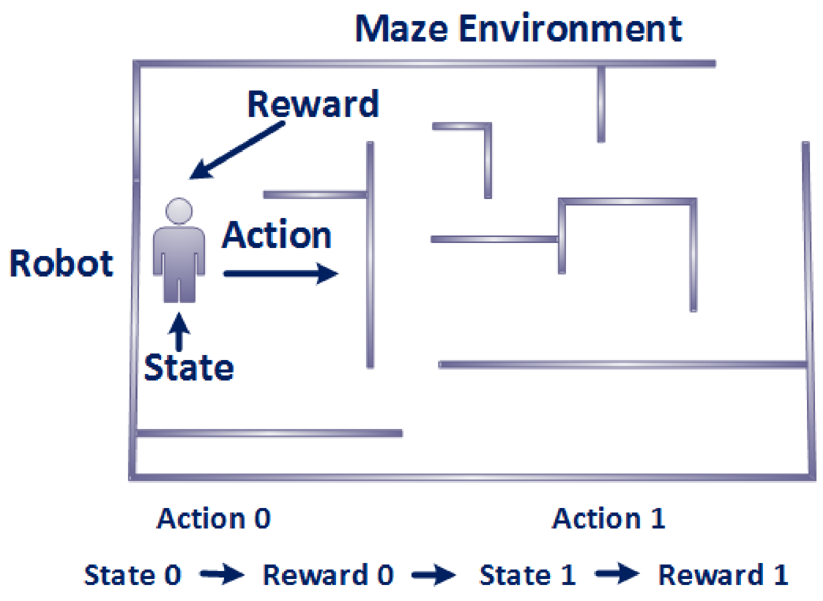


Figure 2.8. Reinforcement Learning Example

Reinforcement learning systems typically consists of four elements: a policy, a reward function, a value function, and in some cases, a model of the environment [13]. A policy describes the way in which the learner behaves in a given situation. The policy maps the perceived state of the environment to the action the learner will take in that situation. The reward function is based on the goal of the learning problem. Each state is given a reward value that informs the learner about the desirability of that state. The value function defines the total amount of reward the learner can expect to accumulate over time, starting from that state. The reward function specifies the immediate reward, whereas the value function looks ahead to the future rewards the learner will earn based on its previous state. The final element, the model of the environment, can be used by the learner to help to predict the expected rewards as possible next states based on the current state. The learner uses the model to plan its next action [13].

Q-Learning is a reinforcement learning strategy that is classified as temporal-difference learning (TD). Temporal-difference learning methods can learn directly from experience without a model of the system by which to make predictions. This type of learner bootstraps, meaning it updates its estimates based on other learned estimates without waiting for the final outcome or receiving its final reward [13].

Q-Learning itself is a fairly simple approach. The Q-Learning algorithm is defined by s_t , a non-terminal state visited at time t, the learned action-value function Q, and an action a. Q-Learning chooses the next action that will be taken by selecting the one which will maximize its current reward and the future rewards it expects to learn. The current value in Q-table is updated after the action is taken and the algorithm proceeds recursively to make the next decision and update to the Q-Table. Pseudo code for the basic Q–Learning algorithm is given in Algorithm 1. In this listing, r is the reward at time t, and γ is the discount rate parameter between 0 and 1[12].

Algorithm 1. *Q-Learning Pseudocode [12]*

```
Initialize Q(s,a) arbitrarily
ForEach(episode):
        Initialize s
        ForEach(step of each episode):
                Choose a from s using policy derived from Q
                Take action a, receive reward r
                Observe new state $s_{t+1}$
                $Q(s,a) \leftarrow r + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})$
                $s \leftarrow s_{t+1}$
        Until s is terminal
```

The Q-learning algorithm utilizes a table comprised of entries representing the reward associated with every possible state-action pair. Initially, all entries are initialized to zero or a random value. The learning agent must estimate the reward associated with each state-action pair. From the Q-table the agent will select the action that maximizes its reward at the current time. After it has selected an action, the learner will update its Q-table with the reward it received. This is intended to improve the estimates of the Q-table and allow the learner to constantly tune its policy to receive better rewards. After many iterations, the agent will converge to an optimal policy, though in some cases this may take several thousands of iterations [14].

2.3.2 Classification

Classification is a method of categorizing objects as having a set of attributes and an overall label which describe it. Labels typically consist of a finite set of discrete values. Attributes are a vector of several values which also are typically discrete, but may also be continuous values that are divided into discrete bins. Previous pairs of attributes and labels can be used to make predictions about new instances of objects. In the simplest classification methods, the frequency of occurrence of a certain set of attributes appears with a given label can be used to determine the most likely label for a new instance with similar attributes.

The performance of a classification algorithm can be measured in several ways. Typically, a data set that is being used to develop a classification model will be divided into several training and test sets. The training set(s) will be used to allow the algorithm to learn which labels coincide with a given set of attribute values. This will allow the algorithm to construct a model of the data set. Once the algorithm has been trained, it will be executed on the test data set, which the trained model currently has no knowledge of. The algorithm will only be given access to the attribute values and must predict what label to assign to a new data point. Once this is completed the number of correct classifications can be determined by comparing the algorithm output to the set of known labels that were withheld during the testing phase. In addition to simply counting correct classifications versus incorrect classifications, the performance can be evaluated using precision and recall. Precision and recall are calculated by counting the total true positives t_p, true negatives t_n, false negatives f_n and false positives f_p for examples classified as label l. The F1 score is as a weighted average of the precision and recall. Each of these metrics are discussed in more detail in Chapter 7.

There are several well known and rather simple classification techniques used in this research. Decision trees, K-Nearest Neighbors, Naive Bayes were used to classify neighboring nodes in terms of route suitability. This section will briefly discuss the basic concepts of each.

Decision Trees

Decision trees construct a graph like tree structure based on a series of yes or no decision criteria as learned by associating frequency of occurrences of a specific label value with a specific attribute value. The ID3 decision tree algorithm uses information gain, a function based on the entropy of training data set to determine which attributes should be used as the root decision of the tree, following down through the lower branches of the tree [12]. Information gain is used to determine the relevance of each variable to predicting the category label, it is the amount of information gained about the label from observing a specific attribute. The attribute which has the greatest information gain is selected as the root node, and is the evaluated recursively to generate the rest of the tree structure. The entropy of a data set S is defined as:

$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus} \tag{2.1}$$

where $p_{\oplus}$ and $p_{\ominus}$ are the proportion of positive and negative examples in S. From the entropy, information gain can be obtained in Eq. 2.2:

$$Gain(S, A) \equiv Entropy(S) - \sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v) \tag{2.2}$$

Figure 2.9 shows an example of the decision tree structure that could be used to classify types of animals. Each node in the tree examines the attributes of an animal and the value of the attributes contribute to the determination of a binary decision on whether or not it is a specific type of animal (in this case, a small carnivorous animal like a cat).

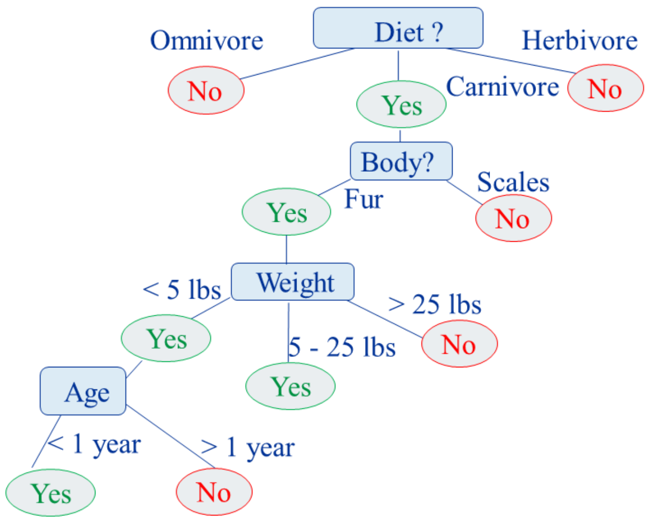


Figure 2.9. Decision Tree Illustration

Naïve Bayes

The concept of Bayesian machine learning is based on the conditional probability that a certain outcome has some likelihood given that it possesses a particular set of attributes. This learning method is used to classify a new instance or occurrence within a set of possible values based on previous training data. The learner will determine the probability that a certain set of attributes most likely correspond to a specific classification within the training data. When a new occurrence is presented to the learner, the training probabilities are used to determine the value v of the new instance from a finite set of values V based on its attribute vector $< a_1, a_2, ..., a_n >$ [12]. There are many benefits to the Naïve Bayes classifier in that it is a very simple method and performance is generally not affected by missing attributes. It has been used for many classification problems and is particularly well known for categorizing text based on words within the text. In this example, the learner will predict what the text is about based on the frequency that certain words (the attributes) appear within the text [12].

Bayesian learning is based on calculating the most probable outcome, often called the maximum a posteriori or MAP hypothesis. The Naïve Bayes classifier attempts to find the most probable value for a current instance V_{MAP} given its known attributes $< a_1, a_2, ..., a_n >$ [12].

$$V_{MAP} = \max_{v_j \in V} P(v_j \mid a_1, a_2, ..., a_n) \tag{2.3}$$

Bayes rule can be used to write Eq. 2.3 as:

$$V_{MAP} = \max_{v_j \in V} \frac{P(a_1, a_2, ..., a_n \mid v_j) P(v_j)}{P(a_1, a_2, ..., a_n)} \tag{2.4}$$

The term $P(a_1, a_2, ..., a_n)$ is dropped since it is a constant independent of v_j. This simplifies to:

$$V_{MAP} = P(a_1, a_2, ..., a_n \mid v_j) P(v_j) \tag{2.5}$$

Equation 2.5 calculates the MAP hypothesis as the probabilities of observing the value v_j in conjunction with the attributes $< a_1, a_2, ..., a_n >$. This is easily found by taking the product of the conditional probabilities of observing v_j given each individual attribute. The Naïve Bayes classifier becomes [12]:

$$V_{NB} = \max_{v_j \in V} P(v_j) \prod_i P(a_i \mid v_j) \tag{2.6}$$

K-Nearest Neighbors

K-Nearest Neighbors is a classification method which classifies objects based on a distance measure relative to other instances in the data set. The attribute vector $< a_1, a_2, ..., a_n) >$ can be thought of as x, y coordinates on a plot and each point will have its associated label. The distance will be calculated from a new point $< b_1, b_2, ..., b_n >$ to all the other neighboring points .The actual attribute vector can consist of more than two dimensions and this will not affect the distance formula, simply more attributes are added. Euclidean distance $d(a,b) \equiv \sqrt{\sum_{i=1}^{n}(a_i - b_i)^2}$ can be used as well as other distance metrics. The number of neighbors selected for the distance calculation is the number K, the K nearest neighbors to the point are chosen for the classification. The new point will be classified as the label that most frequently occurs among the K nearest neighbors. Figure 2.10 shows a graphical representation of classifying a new data point indicated by either a white or black x, representing a binary label such as true/-false or positive/negative based on its 5 closest neighboring points.

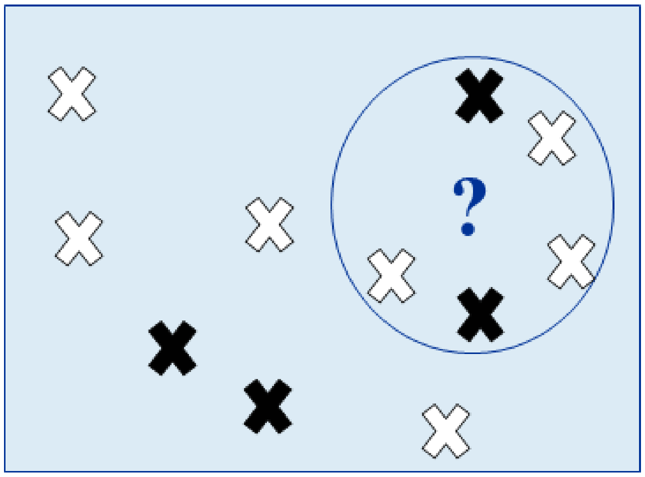


Figure 2.10. K-Nearest Neighbors Illustration

2.3.3 Multi-label Learning and Metrics

The classification methods described in the previous section discusses methods which produce a single label , either binary or selected from a finite set of possible labels, for a single instance of an attribute vector. Multi-label classification allows for there to be multiple categories of labels assigned to one instance of an object. An example of this could be in the classification of books. A multi-label classification of a book could be non-fiction, paperback, and biography. There are several approaches to multi-label classification that were used in this dissertation.

The simplest form of multi-label classification is the Binary Relevance (BR) method. In this case the multi-label classification is decomposed into several binary classifications, one for each label category. This can be considered a problem transformation approach. A single classification problem with multiple outputs is transformed into multiple classification problems. This method has been critiqued for the fact that all labels are classified independently, without taking into account interdependence between outputs.

A large amount of work in multi-label classification has been focused on determining the relationships between labels to improve classification accuracy. Classifier Chains [15] (CC) also transform the multi-label problem into a set of individual binary classifications, however the attribute space for each model is extended with the binary label relevances of all previous classifiers, forming a chain. This takes associations between previously classified labels into account when performing classification of the next label. For this reason, selection of the order in which labels are classified may influence the outcome of how the classifier performs. A poor choice of order can negatively impact performance. In addition, one drawback of the CC approach is that errors can be propagated through the chain, for example by the early choice of a poor order further impacting the performance down the rest of the chain. The Ensemble of Classifier Chains (ECC) attempts to correct this issue by using multiple chains of randomly ordered classifiers, so that the impact of order selection will be decreased overall [15].

To validate the multi-label classification performance, there are four well known multi-label prediction metrics used. Hamming loss calculates the fraction of labels that are incorrectly classified. This is in contrast to zero-one loss which considers the entire prediction incorrect if any label in the prediction is incorrect. Hamming loss is a more lenient metric which scores based on individual labels. In both Hamming loss and zero-one loss, values tending toward zero indicate good performance whereas values tending toward one indicate a higher percentage of misclassification. The Jaccard similarity score is the size of the intersection of two label sets (the predictions and true labels) divided by the size of the union of the two label sets. The F1 score is as a weighted average of the precision and recall. Micro-average F1 score calculates the score globally by counting the total true positives, false negatives and false positives. These metrics are discussed in detail in Chapter 7.

2.3.4 Clustering

Clustering is an unsupervised learning method which can be used to determine how similar two or more attribute vectors are two each other. The method used in this dissertation was the K-Means Clustering algorithm [16], which produces disjoint clusters. The attributes within the data set can be thought of as points on an x, y plot. Initially, K cluster centers are randomly selected among the possible x, y coordinates. Each point ion the data set is assigned to the cluster whose center is closest. Next, the K cluster centers are recalculated as the mean of the coordinates of each cluster. All of the points are reassigned based on the new cluster centers, and the process repeats until the cluster centers no longer change. An understanding of the data set can be developed from information about what points have been clustered together.

Equation (2.7) shows how the clusters are calculated [17]. The goal is to minimize J, the sum of squares of the distances of each point to the centroid of its assigned cluster μ_k. Here K is the number of clusters, N is the number of data points and x_n is each data point to be clustered. The variable r_{nk} is an binary indicator of whether the point x_n should be assigned to cluster k as shown in Equation (2.8).

$$J = \sum_{n=1}^{N} \sum_{k=1}^{K} r_{nk} \left\| x_n - \mu_k \right\|^2 \tag{2.7}$$

$$r_{nk} = \begin{cases} 1 & k = argmin_j \left\| x_n - \mu_j \right\|^2 \\ 0 & otherwise. \end{cases} \tag{2.8}$$

An iterative procedure is used to find values of the cluster centers μ_k and r_{nk} to minimize J.

3 Related Work

This section covers several routing algorithms that were closely related to and/or used in this dissertation. Algorithms such as flooding, epidemic and PRoPHET are well known in the DTN community and often used as points of reference or the basis for new routing approaches. In addition, several lesser known works that are closely related to the topic of machine learning in regard to DTNs are also covered.

3.1 DTN Routing Algorithms

Balasubramanian et al. classify most DTN routing protocols as either based on packet forwarding or packet replication [18]. Replication based routing, or epidemic routing protocols, create multiple copies of a packet to send to neighboring nodes with the intent that the packet will traverse multiple paths and have a greater likelihood to reach its final destination. Forwarding based routing protocols create a single instance of a packet and employ various methods to determine a suitable path, often requiring global knowledge of the network.

In the case of a space network, it can be seen that both of these approaches have their own benefits and drawbacks. As noted in [18], naïve flooding can consume excessive resources on any node by generating multiple copies of unnecessary bundles. In the case of satellite networks, on-board avionics are often quite processor and memory limited, making this unnecessary processing particularly troublesome. Benefits of replication include redundancy to prevent lost packets, and potentially simplified algorithms which require limited knowledge of the global network. The need for feedback regarding the network state in particular can be impractical for deep space communication, where information will likely be stale by the time it reaches its destination. In contrast, while forwarding-based protocols require fewer resources, they often have lower message delivery rates [19]. Furthermore, the use of an oracle with future knowledge of the network, or a knowledge base of the existing network may be difficult to implement in many real-life scenarios [18].

Table 3.1 shows a comparison of the routing algorithms studied in this work. The column "Replication" indicates if the algorithm uses message replication or it forwards a single copy. The column "Topology" lists the intended type of network topology, meaning a fairly randomly changing topology, a more stable graph-like topology and changing topologies particularly focused on VANETs (Vehicular Ad hoc Networks). The final column "Methodology" is a brief summary of the main routing mechanism. The five algorithms listed first (epidemic, DTLSR, PRoPHET, RAPID and CGR) are very well known DTN routing algorithms. The next two listed (Bayesian and Epidemic+C and Saw+C) are lesser known but included due to the similarity and relevance to this work. The final two algorithms (Classification and Clustering) are the algorithms developed in this dissertation.

Algorithm	**Replication**	**Topology**	**Methodology**
Epidemic	Replication	Frequently Changing	Replicate unknown bundles
DTLSR	Forwarding	Relatively Stable	Link state announcements
PRoPHET	Replication	Changing	Probabilistic history of encounters
RAPID	Replication	VANET	Maximize utility metric
CGR	Forwarding	Relatively Stable	Forward based on contact schedule
Bayesian [20]	Replication	VANET	Bayesian classification
Epidemic+C/SaW+C [21]	Replication	VANET	Decision tree
Classification	Replication	Frequently Changing	Ensemble of classifiers
Clustering	Replication	Frequently Changing	Cluster analysis by node location

Table 3.1. Summary of DTN Routing Algorithms

3.1.1 Flooding/Epidemic Routing

Flooding or epidemic routing is the simplest type of routing, aside from statically configured routes. Epidemic routing schemes assume minimal knowledge of the network topology. Messages are replicated each time the transmitting node comes in contact with a new neighboring node [22]. This approach has a high delivery probability and low delay, however it consumes a great deal of resources in terms of message storage and utilizing transmission opportunities for messages that already exist in the network. Nodes will need to determine when copies should be deleted and how to deal with receiving duplicate messages. Furthermore, this approach will not scale well with a large amount of data or a large number of nodes. Solving these issues has given rise to an entire class of epidemic-based routing algorithms. Spray-and-Wait [23] is one such approach in which the transmitting node begins with an initial "spray" phase where copies of each message are sent to each available relay node. The transmitting node then waits to see if one of the relay nodes was the message destination, if not the relay nodes will wait to see if they can forward the message on to its destination.

3.1.2 DTLSR

Delay Tolerant Link State Routing (DTLSR) is based on conventional link state routing [24]. Nodes attempt to learn the network topology by sending flooding messages containing connectivity information for the current state of the network. The network topology is stored by each node in the form of a network graph. Routes are computed using Dijkstra's shortest path algorithm. Link State Announcement messages may contain the source node's endpoint identifier, sequence number and link state information such as the next hop destination and queue status. DTLSR differs from standard link-state routing (LSR) in that currently unavailable nodes are still considered in the best path computation. For nodes that are available, hop count can be used as a simple metric to determine the best path. This does not allow the algorithm to take advantage of better paths that may not currently be available but will be in the future when the message arrives at a remote node. To account for this, DTLSR attempts to minimize the estimated expected delay. For nodes that are available, the delay is estimated based on the total size of all messages in the queue q_{len} (bits), number of messages in the link queue q_{num}, the per-message latency (s) and bandwidth (bps). The estimated delay is given by Eq. 3.1 [25]:

$$delay_{available} = q_{num} \times latency + \frac{q_{len}}{bandwidth} \tag{3.1}$$

The estimated delay associated with unavailable nodes is inferred from the duration of the current outage. This is based on the assumption that if a node has been unavailable for a long amount of time, it is likely to continue to be unavailable. The duration is limited to 24 hours [24].

3.1.3 PRoPHET

The PRoPHET (Probabilistic Routing Protocol using History of Encounters and Transitivity) routing protocol attempts to reduce the number of replicated bundles in the network by calculating the probability of successful message delivery to a given destination. PRoPHET is based on the human mobility model and the observation that a large number of contact opportunities between two nodes follow a non-random pattern [26]. Messages are replicated and sent to neighboring nodes that have a high probability of delivering it to its destination. PRoPHET determines this likelihood based on a delivery predictability metric. Each node maintains a vector of delivery predictabilities for all nodes encountered and exchanges this information with other nodes during an initial contact phase. The delivery predictability is calculated whenever two nodes are in contact. Nodes which are frequently in contact have a higher delivery predictability and as such the algorithm will choose that pair of nodes as the preferred path. The delivery predictability $P(A,B)$ for node A to destination B is calculated as follows [27]:

$$P(A,B) = P(A,B)_{old} + (1 - P(A,B)_{old}) \times P_{enc} \leq 1 \tag{3.2}$$

The probability of direct encounter P_{enc} is a configurable parameter meant to increase the delivery probability of nodes that are frequently encountered. Delivery predictabilities for other nodes encountered by B are updated for node A using the transitive property. The transitive property is based on the concept that if node A frequently encounters B and node B frequently encounters node

C, then node A can be used to forward messages to C via node B [27]. In Eq. 3.3, the value of β is a scaling factor for the transitivity of predictability and is a configurable parameter;

$$P(A,C) = P(A,C)_{old} + (1 - P(A,C)_{old}) \times P(A,B) \times P(B,C) \times \beta \tag{3.3}$$

To reflect changes in the network, the delivery predictability for each node i decays over time according to Eq. 3.4:

$$P(A,i) = P(A,i)_{old} \times \gamma^T \tag{3.4}$$

In Eq. 3.4, T represents the length of time since the probability was last aged and γ is constant. The PRoPHET Internet Draft recommends values of 0.75 for P_{enc}, 0.25 for β, and 0.99 for γ as a starting point, though they may be tuned for a particular application [26].

3.1.4 RAPID

The Resource Allocation Protocol for Intentional DTN (RAPID) was developed at the University of Massachusetts Amherst and was deployed as part of the DieselNet project. It attempts to conserve resources such as bandwidth, storage space, and power by only replicating bundles that optimize a specified routing metric [28]. The RAPID algorithm can be configured to optimize average delay, worst-case delay, or number of bundles delivered before they expire. This is done using a per-packet utility function specific to the desired routing metric. When two nodes encounter one another they exchange metadata about what bundles they have buffered, as well as information from past meetings. Bundles that can be directly delivered to their destination are transferred in order of creation time.

Bundles that are destined for another node in the network are replicated if they do not already exist in the neighbor's buffer. The utility function is calculated for each bundle and they are then selected for transfer in decreasing order of their marginal utility.

The functionality of RAPID is broken into three main elements. A selection algorithm determines what packets to replicate based on their contribution to the optimization of the desired metric. An inference algorithm estimates the bundle's contribution to the selected routing metric. A control channel is used to exchange information about bundles in the network with other nodes [28]. Table 3.2 shows the routing metrics used by RAPID. Here U_i is the packet's utility, or the packet's expected contribution to a given routing metric, $D(i)$ is the packet's expected delay, and S is the set of all packets in a given node's buffer [28].

RAPID estimates the delay in a three-step process. Each node maintains a queue of bundles for each destination in decreasing order of the time they were created. The delivery delay distribution is computed for each bundle as if it is to be delivered directly, based on the number of bytes ahead of it in the queue and the size in bytes of the expected transfer opportunity. This is done for each node possessing a copy of the bundle. The minimum is then found among all delay distributions for each replicated bundle [28].

Metric	**Per-Packet Utility Function**	**Explanation**
Minimize Average Delay	$U_i = -D(i)$	Replicate packets which reduce the delay most
Minimize Expired Bundles	$U_i = \{ \begin{matrix} P(a(i) & < L(i) - T(i)), & L(i) > (T(i) \\ 0 & otherwise \end{matrix}$	L(i) is the bundle time-to-live and T(i) is the time since creation. A bundle that has expired has a utility of 0.
Minimize Maximum Delay	$U_i = \{ \begin{matrix} -D(i), & D(i) \geq D(j) \quad \forall j \in S \\ 0 & otherwise \end{matrix}$	Replicate the packet which is causing the maximum delay

Table 3.2. RAPID Routing Metrics

3.1.5 Contact Graph Routing

Contact Graph Routing [29] (CGR) is a popular DTN routing algorithm, particularly for space networks which have highly periodic contacts between nodes. It is typical in such networks that communication will be scheduled (manually) days, weeks or months in advance. In addition to human scheduling constraints such as operator availability and sharing resources among multiple users, orbital constraints on the communication assets make it relatively straight forward to know when and for how long two nodes will be physically able to contact one another. Contact Graph Routing uses this upfront knowledge to determine suitable routes based on contact times. To do this, CGR uses a contact plan as input to the CGR routing algorithm. The information in the contact plan is entered by users either through update commands in a DTN administration interface program or as configuration files. The contact plan is read at system start up or initiated using an administrative command, requiring privileged access to the node. Alternatively, the CPUP (Contact Plan Update Protocol) [30] has been proposed to allow updates to the contact plan that can be triggered by events such as queue capacity limits.

CGR begins with basic network information which is obtained from user supplied configuration files. These files define a set of contact messages and a set of range messages. A contact message defines a transmission interval, containing the start and stop time that a given contact opportunity pertains to, the transmitting node number, the receiving node number and the planned data rate between the nodes in bytes per second. A range message defines the distance between two nodes in light seconds, consisting of the start and stop time that a given range pertains to, the transmitting node, the receiving node and the anticipated distance between the two nodes in light seconds. The following initial conditions exist:

- Destination variable D is set to the bundle's final destination
- Deadline variable X is set to the bundle's expiration time
- The list of proximate (neighboring) nodes is empty
- Forfeit time is set to infinity
- Best-case delivery time is set to zero
- The list of excluded nodes is populated with the node from which the bundle was received and all excluded neighbors for the destination node.

When a new bundle arrives to be forwarded to another node, the CGR algorithm begins with the contact review procedure as shown in Algorithm 2. In the pseudo code listing, "xmit" objects contain information about contact start time, stop time, transmitting node number, and data transmission rate for all contact messages with receiving node D. The Estimated Capacity Consumption (ECC) is the size of the bundle plus the overhead associated with each convergence layer frame that the bundle will be broken into for transmission. A flow chart of the CGR contact review procedure and forwarding decision as outlined in [29] are shown in Figures 3.1 and 3.2.

Algorithm 2. *Contact Graph Routing Pseudocode [29]*

```
Append D to the list of excluded nodes
For each xmit m in node D's xmit list:
    If m's start time is after the deadline X:
        Skip xmit m
    Else:
        If D is a neighbor of the local node S:
            Compute ECC of bundle from local node to D
                If m's residual capacity < ECC:
                    Skip xmit m
                Else:
                    If D is already in the list of proximate nodes:
                        Skip xmit m
                    Else:
                        Is m's stop time < forfeit time:
                            Set forfeit time to m's stop time
                            Add D to list of proximate nodes
                            Compute forfeit, best case delivery times to D
                            Remove D from excluded node
                            revert forfeit and best case times
    Else:
        If node S is already in the list of excluded nodes:
             Skip xmit m
        Else:
            If m's stop time < forfeit time
                Set forfeit time to m's stop time
                If m's start time > best-case delivery time
                    Set best-case delivery to m's start time
                    Compute estimated forwarding latency
                    Set D=S and X =min(T,L)
                    Invoke Contact Review Procedure recursively
                    Remove D from list of excluded nodes
                    revert forfeit and best case times
```

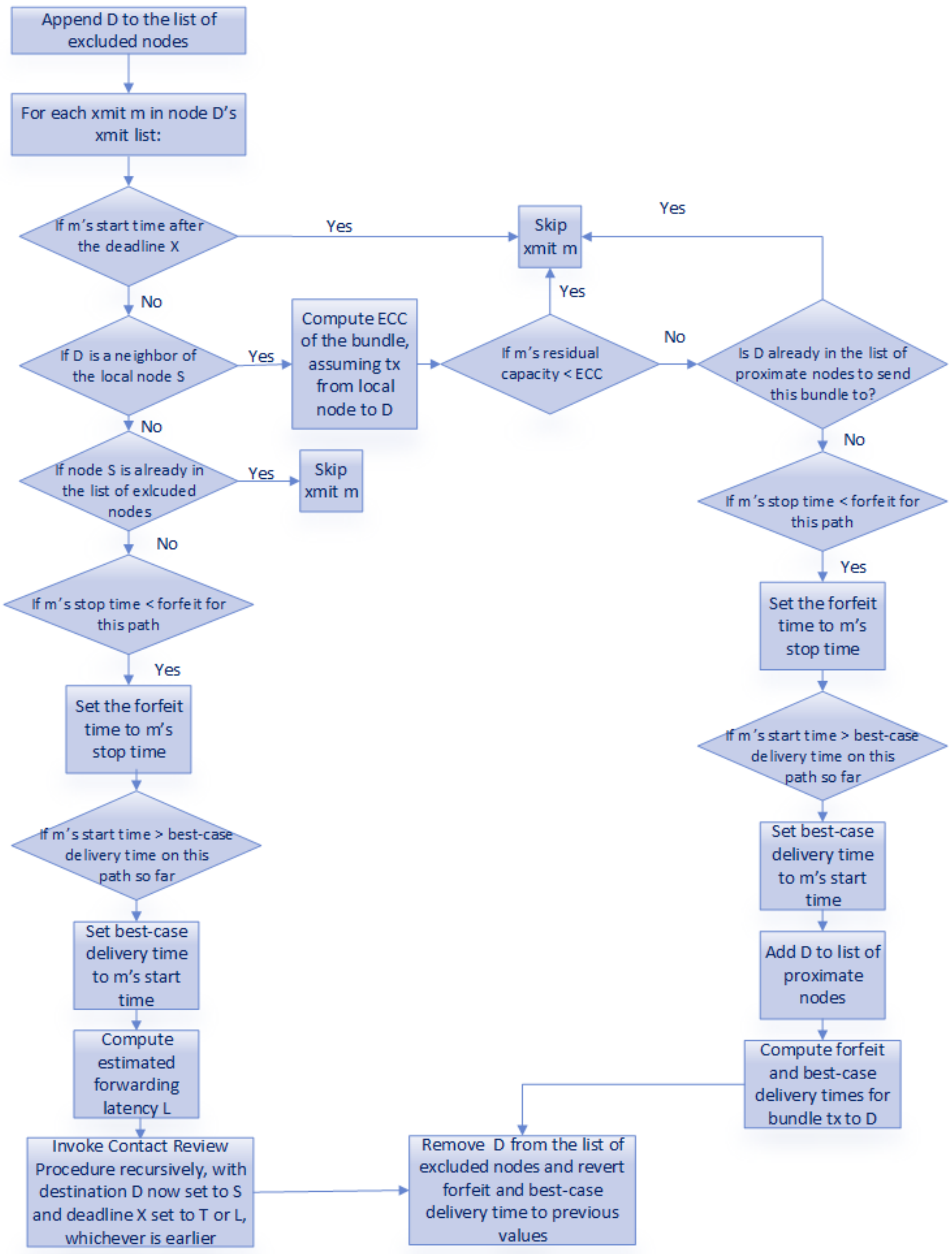

Append D to the list of excluded nodes
For each xmit m in node D's xmit list:
If m's start time after the deadline X
Yes
Skip xmit m
Yes
No
If D is a neighbor of the local node S
Yes
Compute ECC of the bundle, assuming tx from local node to D
If m's residual capacity < ECC
Yes
No
Is D already in the list of proximate nodes to send this bundle to?
No
If node S is already in the list of exlcuded nodes
Yes
Skip xmit m
No
If m's stop time < forfeit for this path
Yes
Set the forfeit time to m's stop time
If m's start time > best-case delivery time on this path so far
Set best-case delivery time to m's start time
Compute estimated forwarding latency L
Invoke Contact Review Procedure recursively, with destination D now set to S and deadline X set to T or L, whichever is earlier
Remove D from the list of excluded nodes and revert forfeit and best-case delivery time to previous values
If m's stop time < forfeit for this path
Yes
Set the forfeit time to m's stop time
If m's start time > best-case delivery time on this path so far
Set best-case delivery time to m's start time
Add D to list of proximate nodes
Compute forfeit and best-case delivery times for bundle tx to D

Figure 3.1. CGR Contact Review Procedure [29]

Once the list of proximate nodes has been populated and the Contact Review Procedure has completed, CGR then makes a forwarding decision based on which proximate node leads to a path to the destination with the smallest best

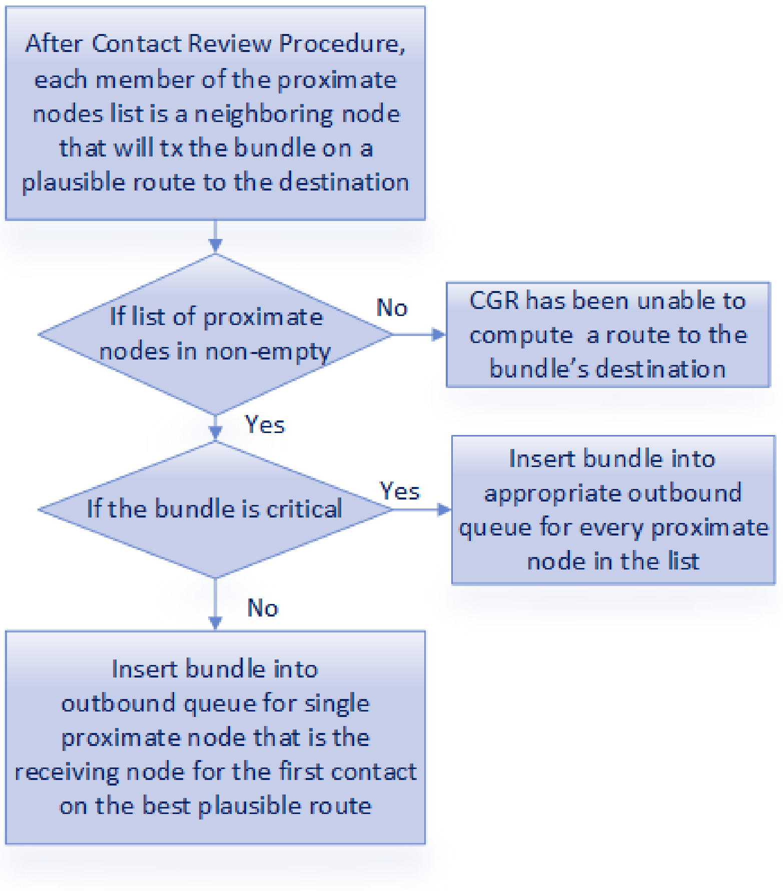

After Contact Review Procedure, each member of the proximate nodes list is a neighboring node that will tx the bundle on a plausible route to the destination
If list of proximate nodes in non-empty
No
CGR has been unable to compute a route to the bundle's destination
Yes
If the bundle is critical
Yes
Insert bundle into appropriate outbound queue for every proximate node in the list
No
Insert bundle into outbound queue for single proximate node that is the receiving node for the first contact on the best plausible route

Figure 3.2. Forwarding Decision [29]

case time. If the bundle is a critical bundle, it is inserted into the outbound queue of every proximate node. If it is a non-critical bundle, it is inserted into only the best proximate nodes' outbound queue.

For simplicity, the original algorithm is given from [29] and does not include the several updates that have been made to the algorithm such as ETO-CGR (Earliest Transmission Opportunity CGR) [31] and overbooking management [31], though it is recognized that these are very relevant improvements.

3.2 Machine Learning Based DTN Routing Algorithms

There are several works which apply machine learning techniques to routing in DTNs and MANETs [20, 21, 32, 33]. Decision tree-based classifiers are applied to make improved routing decisions for epidemic routing by classifying nodes using an attribute vector and a derived classification label[21]. The attributes considered are the node ID, a region code where the message was received, the message reception time, the lobby index (a measure of neighborhood density), the time interval τ between message reception and successful transmission, and the distance δ between where the message was received and transmitted. The region code is determined by dividing the grid of possible node locations into $1\,km \times 1\,km$ squares. The class label is calculated as $r = \frac{\delta}{\tau}$. The value of r is then made into discrete class labels $C = \{C_1, ..., C_m\}$ by separating each instance into approximately equal bins based on a threshold value.

The method explored in [20] focuses on people-based DTNs (Pocket Switching Networks) and therefore makes use of the relatively predictable patterns of human mobility. Nodes are classified based on an affinity index, with attributes consisting of the current time, location, contact probability and contact duration. Nodes individually store network data that is derived by tracing acknowledgments, neighboring nodes can exchange their calculated affinity indices with each other. Both methods from [21]and [20] use stored network traffic history as samples to train their classifiers.

Bayesian classifiers have also been used to tune the performance of broadcasting packets in MANETs [33]. An objective function is developed to assess if the mobility of a given node is contributing to the delivery of broadcast packets. A Naïve Bayes model of successful packet retransmission is constructed with one- and two-hop neighbors, speed, number of duplicates, link duration and traffic as variables. A new broadcast packet is classified by choosing the h_{MAP} (maximum a posteriori) hypothesis to predict if the transmission is successful or not.

These works, while related to the work developed in this dissertation differ from it in several ways. Perhaps most notably none of them attempt to address the particular challenges of space networking, such as long round trip times, or the extended time to receive acknowledgments or other feedback due to asymmetric link rates. In addition, this work developed a multi-label classification approach which takes into consideration multiple hops in a network path, as well as a clustering, rather than simple grid based approach to location determination. In addition, several base classifiers such as K-Nearest Neighbor, Decision Tree and Naïve Bayes are all tested for performance within the same classification framework.

4 Approach

4.1 Architecture

In this section, a software architecture for machine learning based routing is developed. There are several designs decisions that must be considered. Learning may be done by each individual node independently, or system data and learning decisions may be stored and evaluated in a central location, with instructions being disseminated to the individual nodes. There are performance trade-offs to each approach and the type of learning algorithm selected may be more suitable to one architecture over the other. Centralized versus distributed control has been a popular topic for a wide variety of systems, and the same thought process applies to DTN routing.

4.1.1 Centralized Versus Distributed Learning Architecture

A centralized learning architecture as shown in Figure 4.1, consists of a network of multiple "worker" nodes and a central processing and storage location. The worker nodes might be satellites, rovers, science instruments or other mobile nodes. They would have a local storage for intermediate data holding and processing capabilities to perform their specific tasks, execute a local portion of the routing algorithm and other communication activities. They would be fed

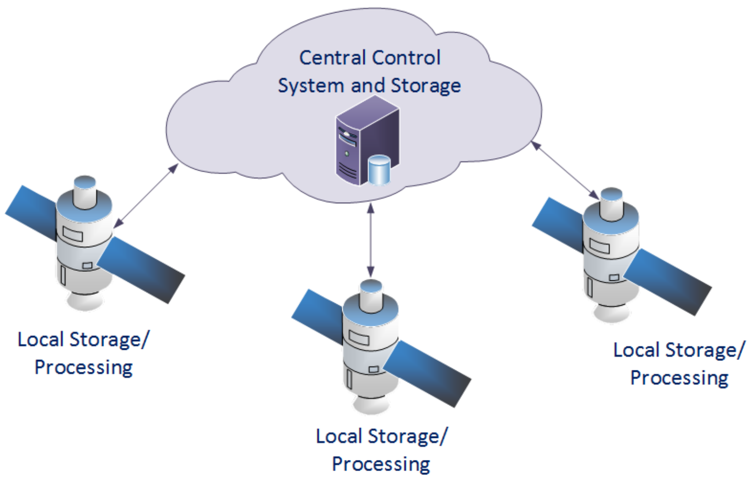


Figure 4.1. Centralized Learning Architecture

potential data path suggestions, schedules and other decisions from the central control system. The central control system would receive network statistics, node locations and conditions which it would use to update the global routing information. In this way, the mobile nodes would be free from the burden of having to have a large storage area for such statistics, or the need to run potentially lengthy data analysis or algorithm training phases. The centralized architecture would work well for supervised learning techniques and unsupervised learning. It could be suitable for reinforcement based learning if most nodes are frequently in contact with the central processor and there is not a long propagation delay between them. This is because reinforcement learning requires feedback from the previous decision, in order to make improvements for a future decision. If nodes receive no response for an extended period of time, the algorithm will not converge on a set of satisfactory policies.

In a distributed learning architecture as shown in Figure 4.2, each node is treated as an equal peer which has sufficient storage and processing capabilities to make independent decisions. The full routing algorithm would be executed on

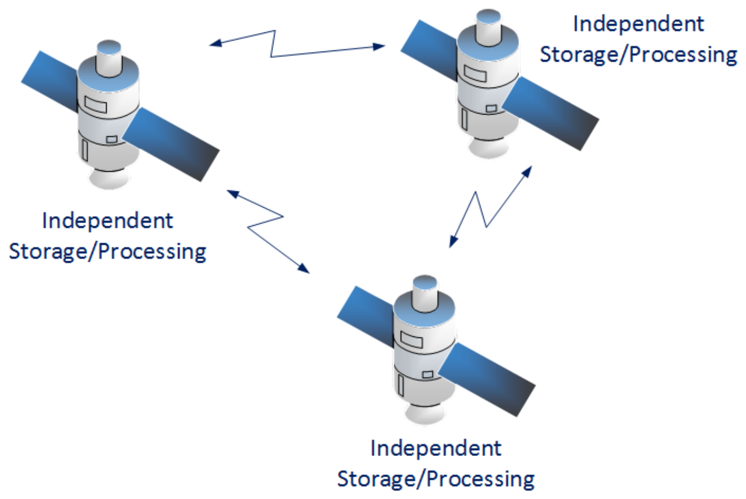


Figure 4.2. Distributed Learning Architecture

each node, but nodes will likely share information with one another when they come in contact with a neighbor. They may trade partial or complete network information from each node's point of view. The distributed approach works well for nodes which may encounter other nodes very infrequently. If the network grows very large, it may be impractical for the node to have a complete topology of the network. The distributed approach would not work well for supervised learning, since the accuracy of most algorithms improves with larger, more complete data sets and typically there is a training phase which can be computationally costly as well as lengthy. Each node would have to complete the training phase on its own, and copy the necessary data to one another. The distributed architecture would work well for reinforcement learning, since intuitively each node acts as an independent learner, which makes a decision and then receives a reward depending on how successful the produced outcome was.

4.1.2 Online versus Offline Learning

There are many trade-offs to be analyzed to determine the fundamental decision of whether the learning algorithm should be performed in real-time (online) or if learning should be done as a sort of post processing approach (offline). Learning done online allows for a more dynamic, flexible approach which may allow for the algorithm to adapt more rapidly to changes in the network. This approach would likely require more processor and memory resources on-board each node, as each node would be performing the computations required by the algorithm in real-time, and would likely have to have access to some database, either local or centralized in another location that houses network statistics information. Online learning could be done in a distributed fashion or centralized, although worker nodes might require frequent access to the central node.

The offline approach would likely require fewer on-board resources, since data can be recorded, stored and analyzed in a centralized area to produce a set of decisions or models which are then disseminated throughout the network. In addition, for systems in which safety is critical, offline learning may provide the advantage that computations and models generated by the learner can be checked for validity before an action is taken using the model. The drawback to this approach is that as the network changes, this may be reflected more slowly as a new updated model would need to be generated.

In this work, both centralized and distributed architectures as well as online and offline learning methods have been implemented. Table 4.1 summarizes each approach and the algorithm used. The next sections of this chapter discuss each method in detail.

Algorithm	Learning	Architecture	On/Offline	Use
Q-Routing	Reinforcement	Distributed	Online	Dynamically update delay estimate throughout network
Classification	Supervised	Centralized	Offline	Determine reliability of Neighbors based on global network knowledge
Clustering	Unsupervised	Centralized	Offline	Determine regions and Similarities within Network

Table 4.1. Algorithm Summary

4.2 Q-Routing

Q-routing is an adaptation of Q-learning developed for packet routing [34]. Q-routing uses the estimated end-to-end packet delivery time for the basis of the Q-table. The table contains a row for each neighbor that a node has. Each column corresponds to a destination node. The entry for the row-column pairs in the table is the estimated time required for a packet to be received at the destination if it was sent from one of the possible neighboring node choices. Each node in the network starts out with an initial Q-table. There are several approaches that can be taken for the initial estimate. One method is to initialize all entries to zero. Similarly, all entries can be initialized to a random value. Finally, a method can be developed to try to calculate an initial estimate of the end-to-end delays. The learner will determine what neighboring node to send a packet to based on which node minimizes the delivery time. Once the packet has been sent to the chosen neighboring node, the neighbor will reply back with what it believes the remaining time will be to deliver the packet to its final destination. This response will be used by the first node to update its Q-table. Each update should incrementally improve the accuracy of the Q-table, since nodes closer to the destination should have a more accurate idea of the remaining delivery time [34].

Pseudo-code for the Q-routing algorithm is shown in Algorithm 3. The value of $Q_x(y,d)$ is the delivery delay estimate for node x to deliver a packet to the destination d via neighboring node y [14]. This serves as the reward function $Q(s,a)$ as discussed in Chapter 2. The selected neighbor $\bar{y}$ has the minimal $Q_x(y,d)$ of all neighbors in the Q-Table of node x. Once the packet has been sent to $\bar{y}$, $\bar{y}$ will simiarly select the node z with the smallest Q-value in its Q-Table to send the packet to and send node x an updated delivery estimate based on the Q value of selected node $\bar{z}$. The variable t is the transmission time from x to $\bar{y}$, q is the time spent waiting in the transmission queue and α is the learning rate. The learning rate is the weight that will be given to new values added to the current estimate.

Algorithm 3. *Q-Routing Pseudocode [14]*

```
While(forever):
        select a packet from the queue
        select $\bar{y}$ from neighboring nodes with minimal Q(y,d)
        wait for a reply from $\bar{y}$
        update $Q(\bar{y},d)$ in current node using new estimate from $\bar{y}$
        $Q(\bar{y},d) = Q(\bar{y},d) + \alpha[Q_{\bar{y}}(\bar{z},d) + t + q - Q(\bar{y},d)]$
end while

When a node receives a packet, interrupt the while and do:
        Receive packet p from node s
        select $\bar{z}$ with minimal Q(z,d)
        send the  value of Q(z,d) back to node s
```

J. Boyan and M. Littman developed the Q-routing algorithm and discuss it in “Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach” [34]. They studied Q-routing by implementing it for a variety of network topologies, including the 7-hypercube, a 116-node LATA telephone network, and an irregular 6x6 grid. For their experiments, they varied the network load and measured the average packet delivery time once the learner had settled on a routing policy. They compare the results to the shortest path algorithm.

Their results indicate that Q-routing performs better than the shortest path algorithm when the network load is high. This is due to the fact that the shortest path algorithm establishes a routing policy on startup and maintains it throughout, with no regard to changes in congestion in the network. Q-routing is able to adapt its routing policy based on the feedback received from each network node. As end-to-end delays increase for a given node, Q-routing is able to help compensate by redirecting future packets to less congested nodes. The delay estimates updated from selected nodes give an indication of the queuing times and level of traffic at each node.

In this work, a simulation model for the network simulator OMNeT++ was developed. The network is implemented as a set of nodes, with each node representing a node in the DTN, which could be relay satellites, cube sats, ground stations or other network assets. Each node consists of three modules: an application module, a routing module and a queue module. The application generates and receives network packets, just as a software application would generate network traffic. The routing module determines where to send the packets that are generated by the application module. The queue module implements a vector of queues for transmitting and receiving the packets. There is one queue for each neighbor that a given node is connected to. The structure of the simulation modules are shown in Figure 4.3.

The connections between nodes are defined as bidirectional. Network packets are generated by the application module. The rate at which packets are generated is configurable. The packet format is very simple and consists of a source address field, a destination address field, the current hop count, the last hop taken and the first hop taken. These fields are used by the routing modules to

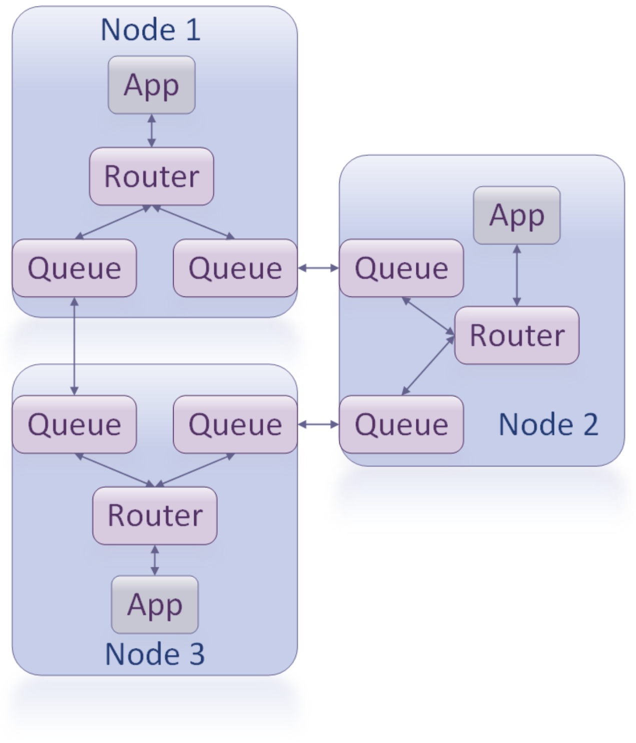
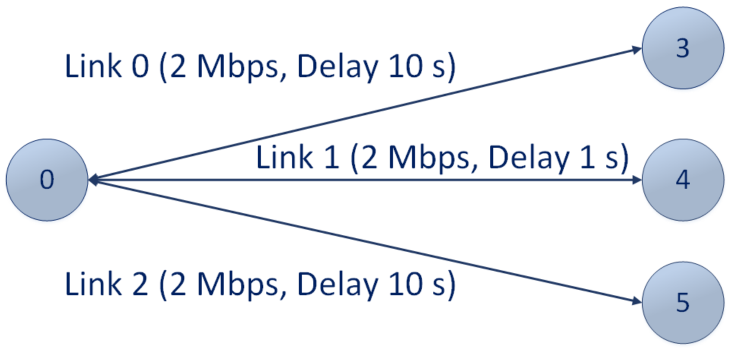


Figure 4.3. Q-Routing Simulation Modules

help determine where to send the packet. A feedback message is generated by the routing module of each node and sent as a reply to the node that last transmitted a packet to the current node. The feedback message contains the address of the node sending the feedback message, the final destination address of the packet that replying node received, the remaining time estimate until the packet reaches its destination and the creation time of the packet. This information is used by the router modules to update their Q-table time estimates.

In addition to generating the network traffic, the application modules perform most of the performance metric measurements. Since the application is the top level module, it generates the packets and is the final module to receive the packets, so it is here that the end-to-end time is measured. The application module also tracks the total number of bytes transmitted, total number of


Link 0 (2 Mbps, Delay 10 s)
Link 1 (2 Mbps, Delay 1 s)
Link 2 (2 Mbps, Delay 10 s)
0
3
4
5

Node Address	0	1	2	3	4	5	6	7	8
Link 0	0	Q_{01}	Q_{02}	Q_{03}	Q_{04}	Q_{05}	Q_{06}	Q_{07}	Q_{08}
Link 1	0	Q_{11}	Q_{12}	Q_{13}	Q_{14}	Q_{15}	Q_{16}	Q_{17}	Q_{18}
Link 2	0	Q_{21}	Q_{22}	Q_{23}	Q_{24}	Q_{25}	Q_{26}	Q_{27}	Q_{28}

Figure 4.4. Example Node and Q-Table

bytes received and periodically calculates the transmission data rate and reception data rate.

The routing policy implemented was to generate a table of initial time estimates for delivery to each node based on the number of hops required to reach the destination node. For each hop in the path, the time is estimated by dividing the number of bits in a packet by the data rate, to get an estimate in seconds. Delivery times are summed for each hop to calculate the total time for the entire path. This is used as the initial entry for the Q-table. Beyond the initialization step, once packets are being transmitted and received, the time estimates in the Q-table will get updated with actual times as measured by the receiving nodes. An example Q-table is shown in Figure 4.4.

The routing module will check an incoming packet's destination address and choose which of its neighboring nodes to send the packet to based on which one has a smaller delay associated with reaching the packet's destination. In addition to this policy, network exploration is encouraged by forcing the node to choose randomly on every 100th packet. This will help ensure that all paths will continuously get updated once the receiving node sends its response with the improved time estimate. If a node fails, other paths will have been updated recently and the learner will have an accurate estimate to use to choose an alternate path.

4.3 Classification Based Routing

This section attempts to solve the routing problem as a machine learning classification task. The motivation behind this method is that the routing solution should be adaptable to a variety of conditions, new nodes entering the network, leaving the network and operating in potentially different time regimes (meaning that surface nodes might be following a certain route which follows a pattern predictable over several hours, other nodes might follow an orbit which repeats every 90 minutes, other nodes may follow a pattern which repeats once a day). In addition,it should be able to determine patterns of disruption or patterns of network traffic which are not immediately obvious. The techniques of machine learning can use data derived from the network environment to determine such patterns.

Classification and supervised learning in general may have some advantages over reinforcement learning for DTN routing. Reinforcement learning has been recommended for routing protocols in several works [32, 34, 35]. There are however some drawbacks in the case of DTNs. One is that a function must be determined to enable the learner to receive rewards. In DTNs, the goal is usually to minimize delivery time and maximize delivery probability. Using time as a metric in DTNs may lead to inconclusive results since delay times may vary either on network conditions which are out of the control of the learner (propagation delays between nodes, for example) or delays may occur because of poor routing choices. Delivery predictability is a good indicator of routing success, however in many cases in DTNs, it may be unknown at the source node if the message was in fact delivered.

Protocols such as TCP and LTP can ensure reliable delivery but since these rely on acknowledgments or retransmission requests from the destination, there may be considerable delay before this is known at the source node depending on the distance between nodes and the data rate. It may be preferred in terms of speed and efficiency to send data in simple datagrams (UDP or LTP green segments). Within Bundle Protocol, delivery receipts and custody transfer can be requested at the bundle layer, but again it is limiting to assume that these mechanisms will always be used. It can be prohibitive to assume that acknowledgments and receipts will be propagated back to the sender and much work has been done in the DTN community to try address the drawbacks of having potentially long round-trip times to send a message and receive an acknowledgement back. Therefore, it is preferred to avoid a protocol that relies on acknowledgement, delivery receipts or status packets, particularly the case in which the timeliness of receiving such feedback is important to the performance of the algorithm. In the case of reinforcement learning, if the learner relies on positive feedback from the destination to make better decisions, this feedback could come at quite a time later and result in a series of poor performance.

Learning the best possible path to a given destination within a mobile network can be viewed as a time series prediction problem. The goal is essentially to determine the future network state based on the history of the network. There are several factors that will influence which route a bundle should take and will be used as input data (attributes) to the learning algorithm. They are the current and future topology of the network (the set of neighboring nodes for the source, destination and relaying nodes), the duration of the contact period, data rate, buffer capacity and location of each neighbor. Each of these features will change in time, though we expect that some or all will follow some predictable periodicity.

The time interval in which a pattern of behavior reoccurs over time can be considered the learning epoch. An example of this learning epoch could be the period of one year, over which from year to year a series of recurrent events are observed (the seasons change in a predictable pattern throughout the year). The term learning epoch is kept intentionally generic since this time interval will be different depending on what is the subject of the learning model. For example, the epoch could be orbital periods, however these periods would be different depending on the orbit of the Earth, a satellite or another planet. The term is used to avoid confusion with other periodic events. Each epoch will be divided into time slices so that the continuous value of time can be treated as discrete attributes or bins. This is often done for classification attributes since there should be some finite set of values which each attribute can take on. This is essentially the concept of having one day (epoch) which is then divided into 24 hours. Each node will have some pattern of mobility and data generation within the epoch that will likely repeat itself over time.

Three well-known classifiers (Naive Bayes, Decision Tree and K-Nearest Neighbors) were selected to determine which would provide the best performance. These classifiers are both simple and intuitively fit the described problem. Nodes are following a given pattern throughout the epoch (for example, humans driving to work every day at the same time) and so it is reasonable to expect that they will continue to follow this pattern in the next time segment (drive home at the same time as well and return back at a similar time the next day). The input to the classifier is based on an attribute vector X consisting of the time index in the epoch, the source node, the destination node, and if the message was delivered or not (1 or 0). The label data Y, or output of the classifier, is the set of nodes that the message was forwarded to. This is encoded as an n-bit string where n is the number of nodes in the network. If the message has visited node i, then the bit in position i is set, it is zero otherwise. This data is obtained from network logs during an initial phase in which routes are determined using epidemic routing. The delivery of bundles and information on forwarding nodes is saved at each node and then compiled in a central location to gain a complete view of how bundles propagated throughout the network. This is discussed in further detail in Chapter 5.

The classifier is trained with historical values for each message sent in a test emulation consisting of attributes X and the forwarded node string corresponding to each message. A subset of the test data is withheld to validate the model. Only the X attribute string (time, source and destination) is given to the model and it will output a prediction for the most likely set of nodes a message will be forwarded to. The performance of the classifier is evaluated by comparing the actual output Y of the test set to the output of the prediction. Once a suitable model has been obtained, this can be integrated into a routing software module that will supply a set of nodes that are the best candidates to forward a message to based on the current time, source node and destination node. Figure 4.5

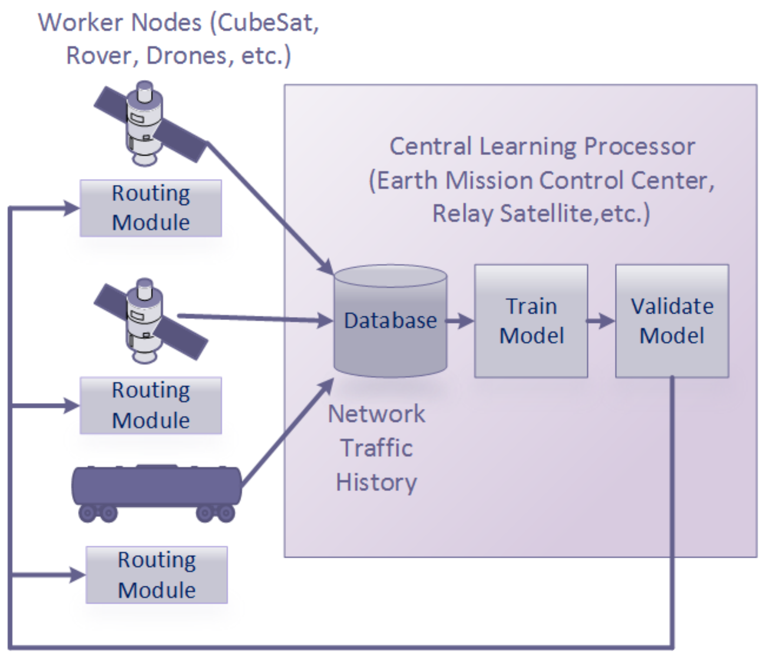

Worker Nodes (CubeSat, Rover, Drones, etc.)
Routing Module
Routing Module
Routing Module
Central Learning Processor (Earth Mission Control Center, Relay Satellite,etc.)
Database
Train Model
Validate Model
Network Traffic History

Figure 4.5. High Level Learning Architecture

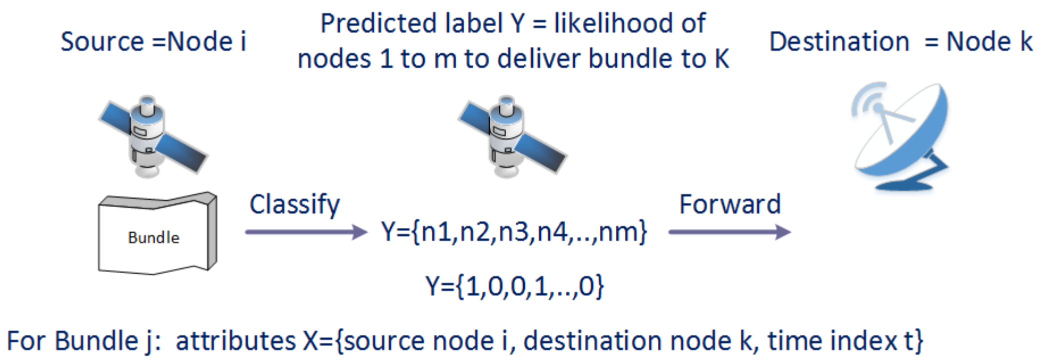

Source =Node i
Predicted label Y = likelihood of nodes 1 to m to deliver bundle to K
Destination = Node k
Bundle
Classify
Y={n1,n2,n3,n4,..,nm}
Forward
Y={1,0,0,1,..,0}
For Bundle j: attributes X={source node i, destination node k, time index t}

Figure 4.6. Example Classifier Attribute Vector and Prediction

shows the high level architecture for this routing scheme,using a centralized architecture as discussed earlier in this chapter. Figure 4.6 shows the concept of the attribute vector X and output variable, or label, Y.

The method described divides the routing classification problem into n separate problems, meaning one classification for each node in the network. Each classifier produces a binary output indicating a node is or is not a member of the set of nodes along a given route. This can be considered a multi-label classification approach, in particular the Binary Relevance method (BR) [36]. As noted in Chapter 2, the Binary Relevance method has the draw back of considering classifications independently. For this reason independent classification (BR), Classifier Chains and an Ensemble of Classifier Chains [15] are each tested for performance in Chapter 7.

4.4 Clustering

Clustering techniques can be used to recognize patterns and similarities over a large set of data. An intuitive use of clustering could be to divide nodes into similar location regions. This could be used to predict the movement of nodes such as μ sats, drones, wireless sensor nodes or other systems in which there may be a large number of worker nodes entering different areas and exchanging data (performing as data mules or message ferries). One of the strengths of machine learning and classification is to take into account multiple previously observed attributes to give an overall probability for a given outcome. Additional features such as location, buffer capacity and data rate may improve performance by taking into account possible delays caused by slow links, or excessive queuing times. For this reason, clustering could be used in two different ways to gain insight into node patterns of movement. The first way would be to solely use the output of the cluster analysis to make decisions about what actions nodes should take, for example selecting nodes to forward data to based on their current or future region (assigned cluster). Another method would be to use the assigned cluster as an attribute to a classifier, along with other attributes such as delivery history, data rate or predicted buffer capacity.

The approach in this dissertation is to use the K-means clustering algorithm [16] (discussed in Chapter 2) to determine regions in which nodes frequently visit, much like the region code used in [21]. Rather than simply dividing the area into a grid based system, the K-means clustering algorithm will provide a data-driven approach to grouping node locations. It can be used on its own to make decisions based on past, present and likely future node locations and associations between neighboring nodes and clusters. Node location clusters are used to determine whether the transmitting node will be within the destinations cluster region in time $t + \Delta t$ seconds.

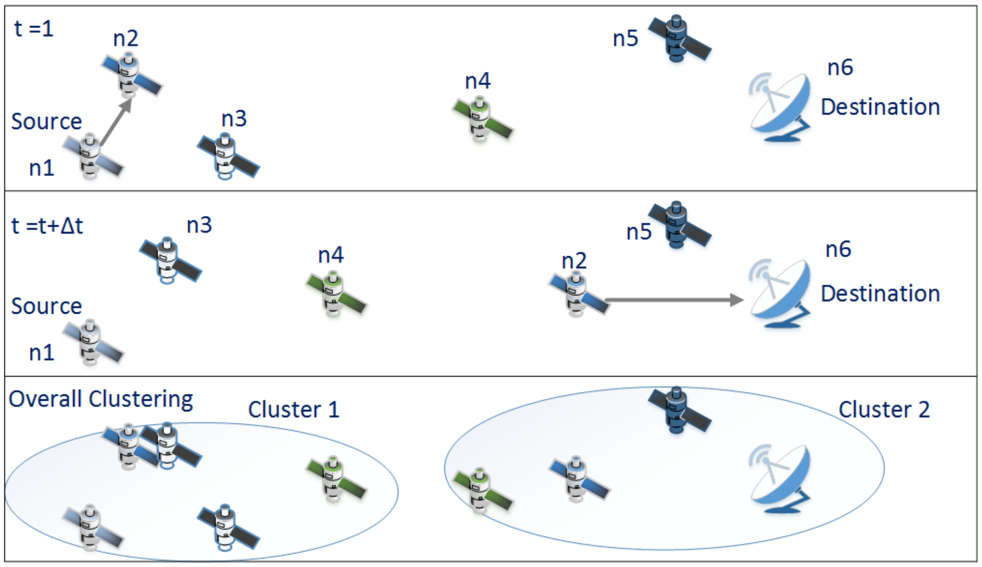


Figure 4.7. Example of Clustering Over Time with K=2

Figure 4.7 shows an example with K=2 clusters. The node $n1$ has a message to send to node $n6$. The locations of the nodes are shown at time $t = 1$ and later at time $t = t + \Delta t$. Clusters could be assigned based on previously recorded node locations which are now similar to the behavior at $t = 1$ and $t = t + \Delta t$ as shown in the overall clustering. In this case, node $n1$ should choose $n2$ to forward its message to $n6$ since this is a node in its current cluster which is known to enter the cluster to which $n6$ belongs.

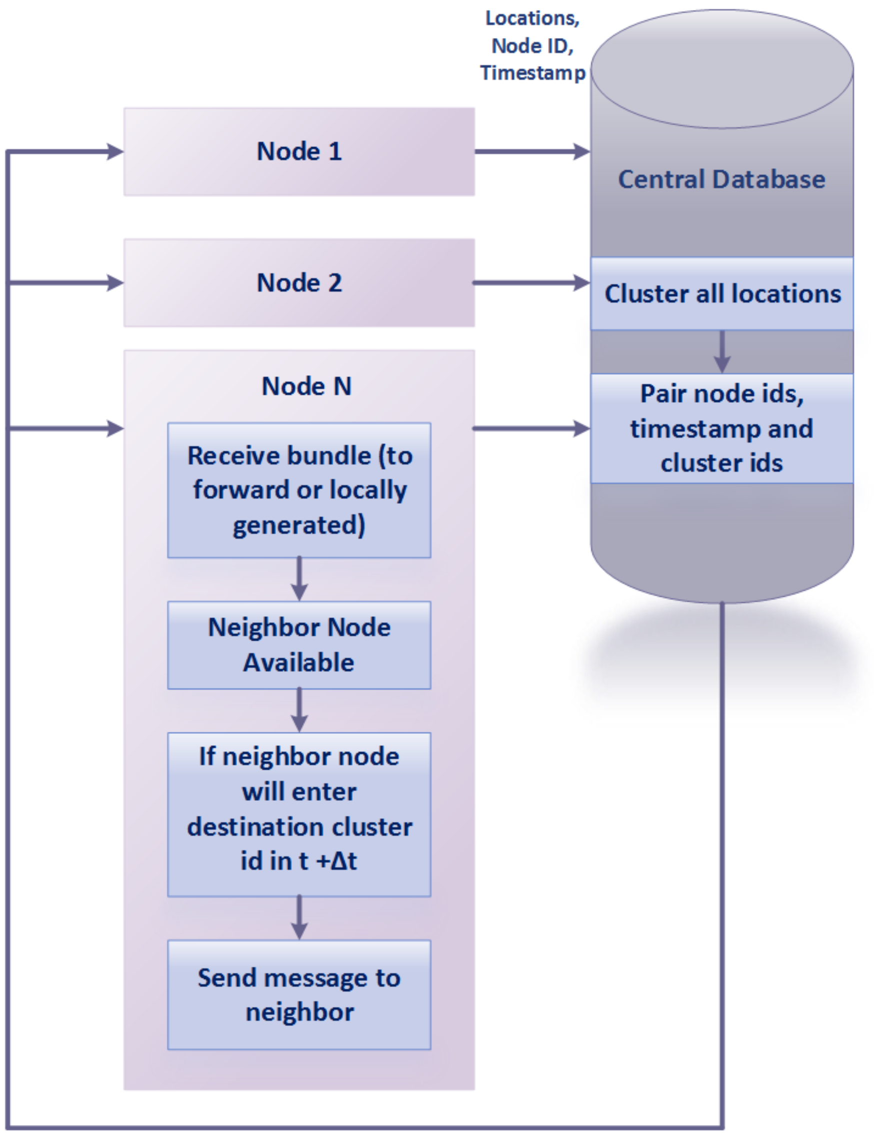

Locations, Node ID, Timestamp
Node 1
Node 2
Node N
Receive bundle (to forward or locally generated)
Neighbor Node Available
If neighbor node will enter destination cluster id in t +Δt
Send message to neighbor
Central Database
Cluster all locations
Pair node ids, timestamp and cluster ids

Figure 4.8. Clustering Block Diagram

A block diagram of the process is shown in Figure 4.8. The worker nodes can periodically report their location to the central processing node, or it can be known by some other means (observed by another node or following a planned path). The central processing node will perform the clustering over the learning epoch as discussed earlier in this chapter, meaning the range of time over which it is predicted that there will be a pattern of recurrent behavior. The central processing node will disseminate the routing information to all nodes in the form of a series of node ids, time indices and cluster ids pertaining to each node at the given time. The worker nodes will use this information to select appropriate neighbors to forward their messages to. If the neighboring node will enter the destination's cluster with in a given interval of time Δt, it should forward the message. The selection of Δt should at least be shorter than the message's time to live, otherwise it is possible that it will expire before the message can be delivered.

5 Implementation

This chapter covers the selection of a bundle protocol implementation and how it was modified for use with machine learning. The various DTN packages that are available were evaluated to determine which was the most suitable for the basis of this work. It was not desirable to develop a completely new bundle protocol implementation due to the length of the time involved and also the fact that there are several available, as well as the focus of this research being more on routing algorithms rather than solely the bundle protocol implementation itself. Several goals were determined to evaluate each DTN implementation. The DTN software should:

- Provide a complete bundle protocol implementation and several convergence layers
- Provide a IP Neighbor Discovery (IPND) implementation
- Provide several example routing protocols
- Be efficient enough to run on embedded targets or an emulation environment
- Provide an up to date version with minimal dependencies

It was found that among ION (Interplanetary Overlay Network), DTN2, IBR-DTN and μPCN, that IBR-DTN was the best match to these goals. The Common Open

Research Emulator (CORE), discussed in Chapter 6, was used to emulate a network of multiple nodes running Ubuntu Linux and IBR-DTN [37] bundle protocol implementation. Scikit-learn [38] and scikit-multilearn [39] Python libraries were used to implement machine learning algorithms within IBR-DTN.

5.1 DTN Implementations

This section discussed several open source DTN implementations that are available. The motivation and use-cases for each implementation, as well as what specific features are provided are covered.

5.1.1 DTN2

DTN2 is the reference implementation provided by the DTN Research Group [40]. It was developed as part of the PhD research of Michael Demmer at University of California at Berkeley [25]. DTN2 was developed based on the idea of using store-and-forward techniques to provide network connectivity to developing, rural areas with otherwise very limited network access. DTN2 is written in C++ and runs primarily on UNIX based systems. It provides a number of convergence layers including TCP/IP, UDP, Bluetooth, and NORM [41]. It also provides a collection of built-in routing protocols (Static, epidemic, PRoPHET and DTLSR). In addition, it provides an XML-based interface to allow user developed routing protocols to send routing commands to the DTN2 daemon. For this reason, a number of external routing protocols have been developed for DTN2, including RAPID [18] and HBSD [42]. DTN2 hasn't been updated since 2012, with current version being 2.9.

5.1.2 ION

ION was developed by the NASA Jet Propulsion Laboratory [43]. ION is focused on communication between interplanetary spacecraft, as well as their associated ground segment. In this case, communication between network nodes is often scheduled well in advance. ION supports TCP/IP, UDP and LTP. It is interoperable with DTN2, though they use slightly different addressing schemes. ION supports Linux, RTEMS, VxWorks, Android and Windows. It continues as of the present (2018) to be updated with new features added often. The current version at this time is 3.6.1. ION supports Contact Graph Routing, which was specifically designed with interplanetary routing in mind (predetermined contact times, efficient operation and does not require the use of network status packets). Newer versions support an opportunistic form of Contact Graph Routing as well.

5.1.3 IBR-DTN

IBR-DTN, Instituts für Betriebssysteme und Rechnerverbund (Institute of Operating Systems and Computer Networks) Delay Tolerant Network, is a bundle protocol implementation that is focused on embedded systems [37] developed at the Technical University of Braunschweig. It has a much smaller footprint and RAM usage in terms of the DTN daemon and associated libraries when compared to both DTN2 and ION. IBR-DTN is written in C++. It supports TCP/IP, UDP, HTTP and IEEE 802.15.4 (LoWPAN) convergence layers [44]. It also supports IPND neighbor discovery, static routing, and epidemic routing with bloom filter, PRoPHET and bundle forwarding based on discovery. IBR-DTN has been tested on Linux, Android, Mac OS X, Raspberry Pi and BeagleBone. IBR-DTN was currently updated as of November 2017.

IBR-DTN [37], [44] is a lightweight Bundle Protocol package that was designed specifically with embedded systems in mind. Several performance analysis studies have shown it to have bundle throughput comparative to ION [7] and DTN2 [41] Bundle Protocol implementations, while also being able to run on very resource constrained platforms such as the Technologic Systems TS-7500 SBC (ARM 9 running at 250 MHz with 64 MiB RAM) [45], [46]. IBR-DTN is an event driven architecture. The arrival of new bundles, discovery of new neighbors, transfer of bundles and loss of link connectivity all trigger events within the system which run in parallel threads. IBR-DTN provides implementations of flooding based routing, epidemic routing and PRoPHET routing, which served as examples for the development of the classification-based router.

5.1.4 μPCN

The Micro Planetary Communication Network (μPCN) is another lightweight bundle protocol implementation focused on embedded systems [47]. It supports POSIX operating systems, FreeRTOS, and can also be compiled to run on bare metal for the ARM Cortex STM32F4 microcontroller. μPCN provides the bundle protocol, IPND discovery protocol and a message ferry based routing. Much work has been done regarding the use of μPCN in Ring Road networks [48], the concept of using LEO CubeSats and DTN protocols to create a message ferry system which would provide low cost network connectivity to developing regions.

5.2 Machine Learning Routing Implementation

IBR-DTN was selected as the basis of this work from the several available DTN packages. It is very resource efficient, allowing it to perform satisfactorily in an emulated environment. It supports several convergence layers, such as TCP/IP and UDP which are convenient for testing routing performance. It provides an IPND discovery implementation, which is used for opportunistic networking. It also provided multiple routing extensions as examples for development and testing. The next sections cover the basic architecture and components of IBR-DTN as shown in Figure 5.1.

5.2.1 IBR-DTN Architecture

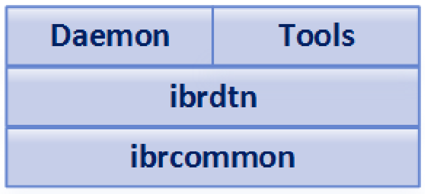


Figure 5.1. IBR-DTN Software Structure [49]

5.2.2 Daemon

The main functionality of IBR-DTN is within the daemon libraries. All of the main components of the DTN node capability are developed here.

- API - Provides underlying functionality for a management interface.
- Core - Provides bundle core, bundle event, bundle filter, event types, fragmentation management, node classes, and timing functionality.
- Net - Provides bundle transfer and reception classes, connection management, convergence layers, and discovery agent.

- Routing - Provides the base router class, neighborhood database, node handshake class, scheduling, retransmission and static routing. Also provides extensions to the base router including epidemic routing, flooding, and PRoPHET routing.
- Security - Provides security certificate and key management.
- Storage - Defines bundle storage, bundle index and result functions, memory-based bundle storage, SQLite bundle storage, and metadata storage.

5.2.3 ibrcommon

Utility functions used throughout IBR-DTN are stored in the ibrcommon library.

- Data - Provides classes for BLOB data type, bloom filters and IO buffers.
- Link - Provides link management functions for POSIX and Win32, as well as link monitoring.
- Net - Provides socket streams, address and interface functions.
- SSL - Provides functions such as RSA SHA256 streams, MD5 streams, hash and cipher streams.
- Thread - Provides mechanisms for thread safety such as mutex lock, semaphores, thread safe queue, timers and clocks.
- XML - Provides XML stream reader, writer and handler functions,

5.2.4 Tools

IBR-DTN provides several application layer tools. They include dtnsend, dtnreceive, dtnping, dtntracepath, dtninbox, and dtnoutbox.

5.3 IBR-DTN Modifications

In order to implement machine learning enabled routing, several modifications were made to the IBR-DTN source. The epidemic routing module was used as a basis for the new routing module. The IBR-DTN daemon runs the routing extension in its own thread, which listens for and raises routing related events. These events monitor the status of the bundle queue, bundle transfer, connection status, and routing handshakes.

5.3.1 IBR-DTN Routing

Figure 5.2 shows a high level overview of how bundles are processed by the routing module. Bundles may be received from other nodes via one of the convergence layers (CL) or they may be bundles that were created by an application on the local node through the DTN application interface (API). In either case, an event will be generated to notify the base router module that new bundles are available. The base router class essentially provides management functions to all routing extensions used in IBR-DTN. It communicates with the routing extensions by generating and listening to events related to the network topology and status of bundles in storage and queued for transmission. When the new bundle is processed it will generate a bundle queued event. This will alert the API and routing extensions that a new bundle is available. The API will be used if the bundle is destined for delivery to a local application, otherwise it will be processed by the routing extensions.

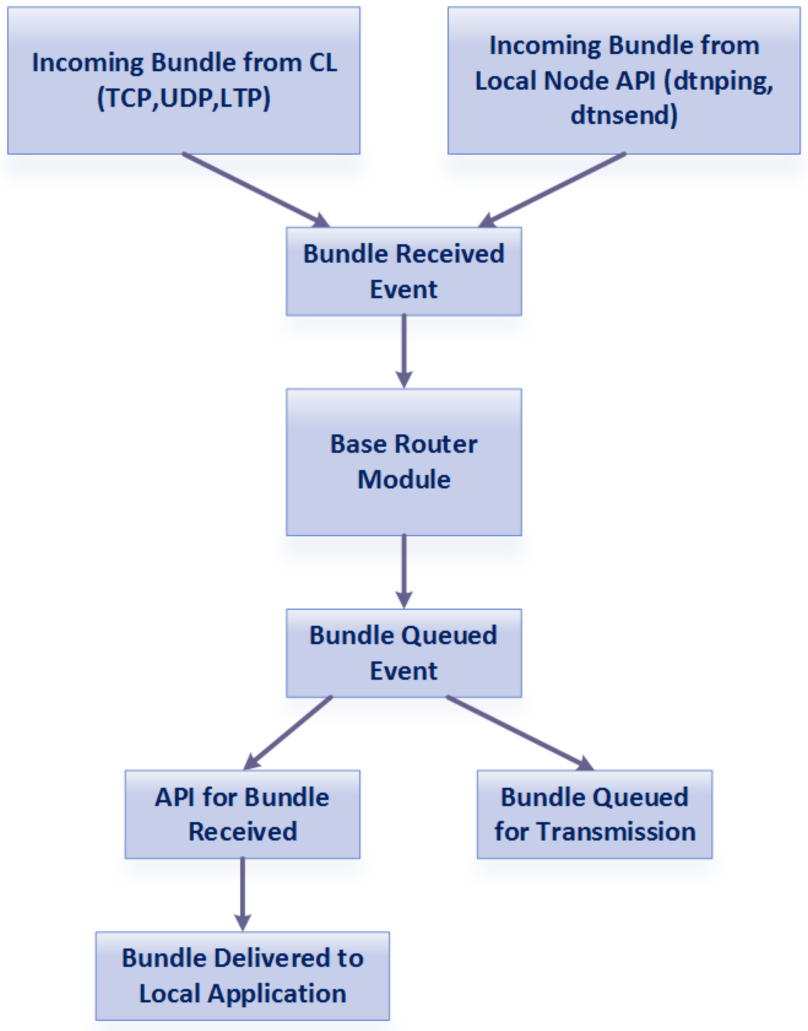

Incoming Bundle from CL (TCP,UDP,LTP)
Incoming Bundle from Local Node API (dtnping, dtnsend)
Bundle Received Event
Base Router Module
Bundle Queued Event
API for Bundle Received
Bundle Queued for Transmission
Bundle Delivered to Local Application

Figure 5.2. IBR-DTN Bundle Reception[49]

IBR-DTN uses an inheritance structure in which the base router defines some limited capabilities which are then further implemented by routing extensions. The routing extensions consist of a default neighborhood routing, static routing, flooding, epidemic and PRoPHET routing, as well as custom defined modules. The base router has access to the neighbor database which contains a list of all known neighbors as well as a summary vector of all the bundles which are known to that neighbor. It also maintains a local summary vector of all bundles in the local storage as well as a vector of purged bundles.

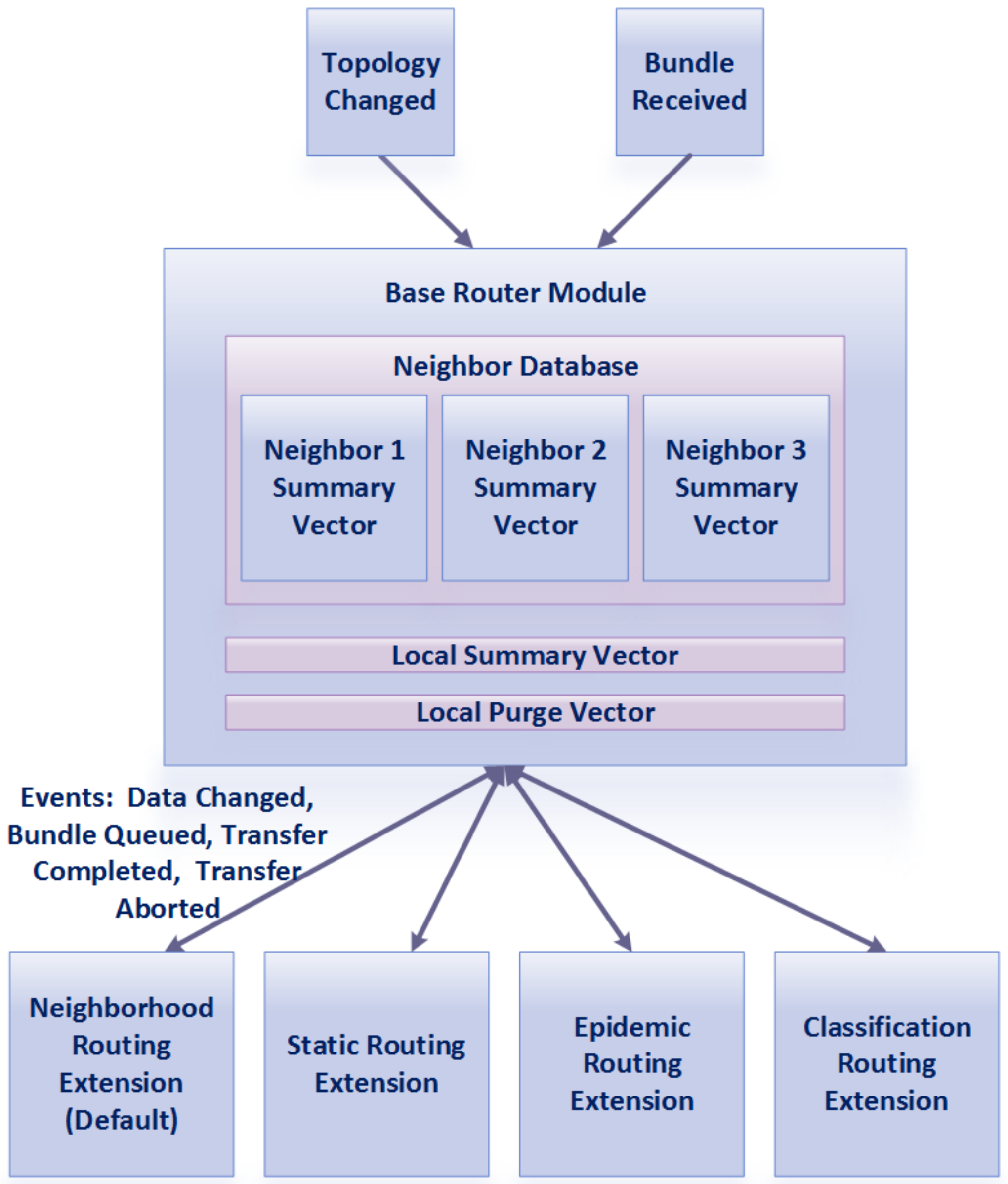

Topology Changed
Bundle Received
Base Router Module
Neighbor Database
Neighbor 1 Summary Vector
Neighbor 2 Summary Vector
Neighbor 3 Summary Vector
Local Summary Vector
Local Purge Vector
Events: Data Changed, Bundle Queued, Transfer Completed, Transfer Aborted
Neighborhood Routing Extension (Default)
Static Routing Extension
Epidemic Routing Extension
Classification Routing Extension

Figure 5.3. IBR-DTN Base Router[49]

Figure 5.3 shows the relationship between the base router and routing extensions. When the network topology has changed or a bundle is received, the base router is notified. The base router will issue and listen to data changed (something in the network has changed), bundle queued, transfer completed and transfer aborted events between the routing extensions. These events are the signaling mechanism for when routing modules should begin the search for new bundles, when bundles have been successfully transferred or when a transfer has failed.

IBR-DTN uses neighborhood routing as the default routing extension if no other routing is configured. The neighborhood routing extension will only deliver bundles that are destined for a direct neighbor. In addition, static routes can be configured for when a certain destination EID should always be transferred through a specific path.

Figure 5.4 shows a block diagram of the neighborhood routing extension. The data changed, bundled queued, transfer completed and transfer aborted events will trigger the routing extension to begin a bundle search. The routing module will iterate through each neighbor in the neighbor database. A bundle filter is created to determine which bundles in the bundle storage should be sent to the current neighbor. The routing module will access the bundle storage to obtain the bundle metadata (bundle id, origin, destination, time to live, hop limit). For neighborhood routing, the criteria are the hop limit should not be zero, the bundle should not be for local delivery, the bundle should not be for other nodes (since this is direct delivery, not forwarding), it should not be over the payload limit for the neighbor, it should be a singleton bundle and it should not be in the neighbor's summary vector (it is already in the neighbor's bundle storage). If these criteria are met, the bundle is added to the bundle filter and all bundles in the filtered list will attempt to be sent to the neighbor. The bundle is not copied from storage when it is added to the bundle filter, only the bundle id is used to refer back to which bundle will be transmitted.

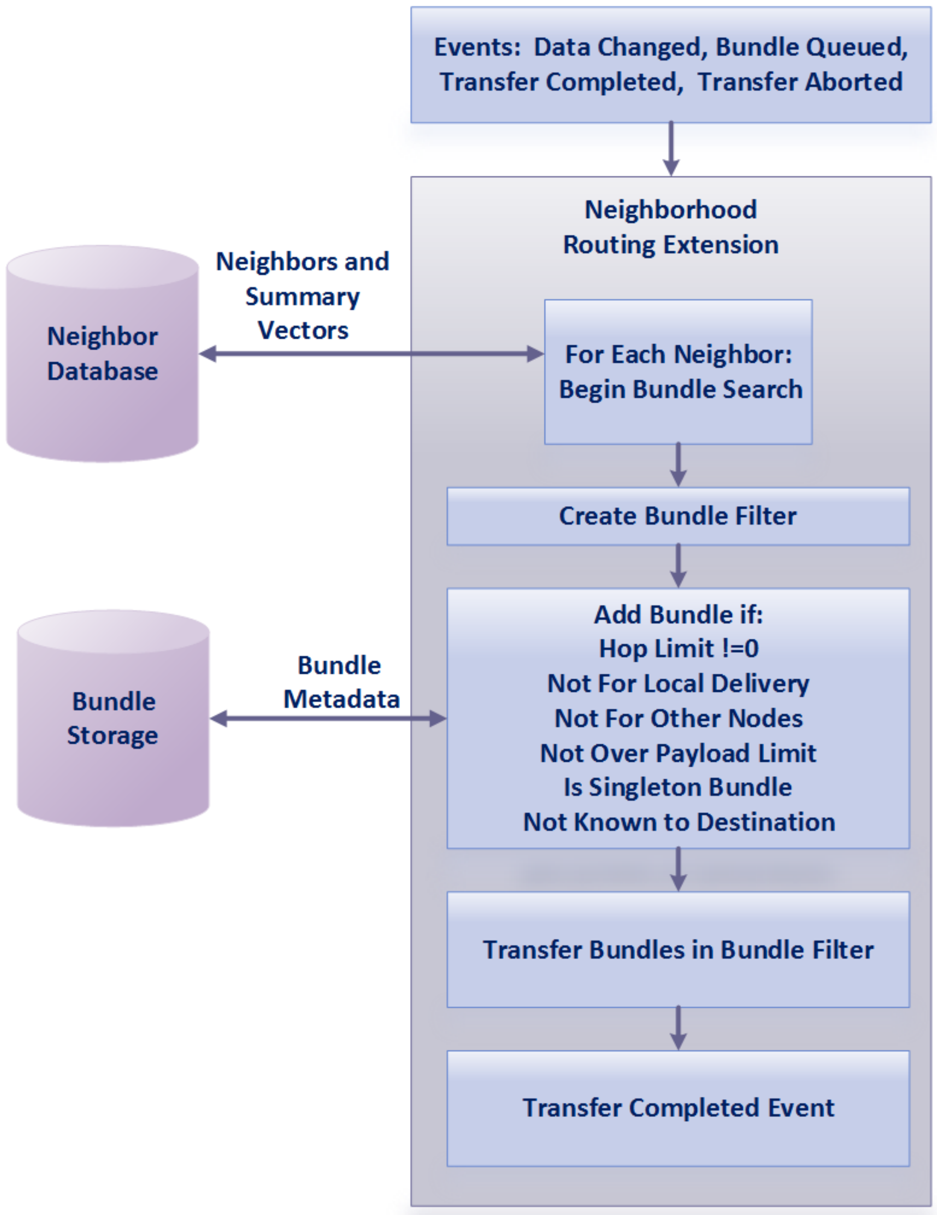

Events: Data Changed, Bundle Queued, Transfer Completed, Transfer Aborted
Neighborhood Routing Extension
Neighbor Database
Neighbors and Summary Vectors
For Each Neighbor: Begin Bundle Search
Create Bundle Filter
Bundle Storage
Bundle Metadata
Add Bundle if: Hop Limit !=0 Not For Local Delivery Not For Other Nodes Not Over Payload Limit Is Singleton Bundle Not Known to Destination
Transfer Bundles in Bundle Filter
Transfer Completed Event

Figure 5.4. IBR-DTN Neighborhood Routing Extension[49]

The classification and clustering based routing extensions follow in a similar manner to the neighborhood routing extension. Figure 5.5 shows a block diagram of the classification based router. The same events which trigger the neighbor routing extension will trigger the classification routing extension.

The first step in the classification based routing extension is to load the learning model into memory. This is the trained model that has been generated by the central processing node from the stored network statistics obtained from the worker nodes as discussed in Chapter 4. Next, each neighbor will be queried in the neighborhood database. The bundle filter will now look for bundles which have a hop limit greater than zero, are not for local delivery, are not over the neighbor's payload limit, are singleton bundles, are not known to the neighbor, are not for direct delivery to the neighbor and additionally must be predicted for delivery to the neighbor. The learning model will be used to determine if the bundle should be forwarded to the neighbor based on the inputs of the local EID (the endpoint id of the local node), the endpoint id of the destination, the endpoint id of the current neighbor being considered, and the current time index in the learning epoch (described in Chapter 4). If the prediction returns true, the bundle will be added to the list of bundles to transfer to the neighbor. The next section discusses the learning procedures in greater detail.

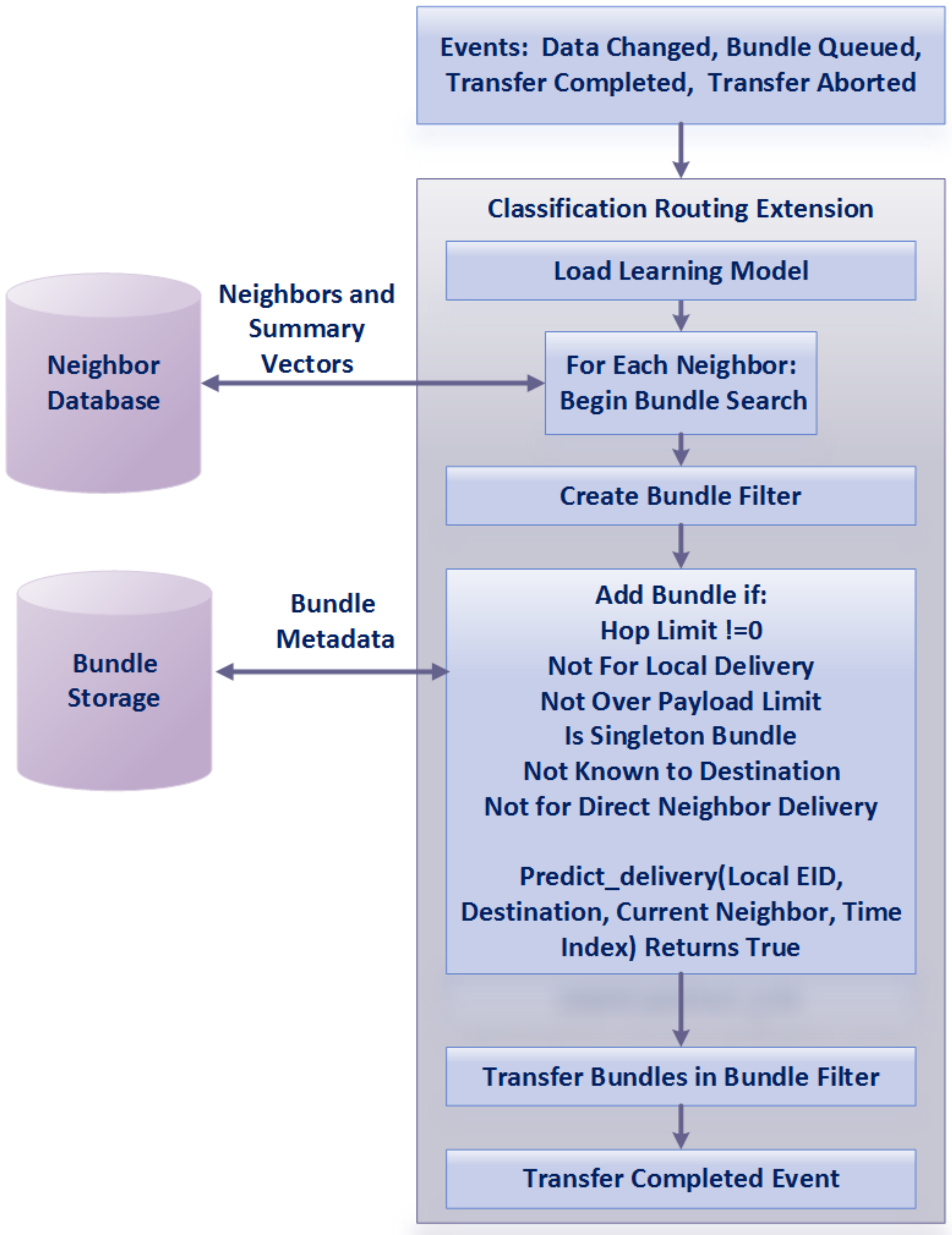

Events: Data Changed, Bundle Queued, Transfer Completed, Transfer Aborted
Classification Routing Extension
Load Learning Model
Neighbor Database
Neighbors and Summary Vectors
For Each Neighbor: Begin Bundle Search
Create Bundle Filter
Bundle Storage
Bundle Metadata
Add Bundle if:
Hop Limit !=0
Not For Local Delivery
Not Over Payload Limit
Is Singleton Bundle
Not Known to Destination
Not for Direct Neighbor Delivery
Predict_delivery(Local EID, Destination, Current Neighbor, Time Index) Returns True
Transfer Bundles in Bundle Filter
Transfer Completed Event

Figure 5.5. Classification Routing Extension

5.4 Learning Procedures

For maximum flexibility in the choice of machine learning algorithm and ease of coding and changes, the machine learning portion of the routing extension was developed in Python. IBR-DTN is written in C++. While Python natively provides the capability and header files needed to call the Python interpreter within a C/C++ application, this can be a cumbersome process. Wrappers must be defined to handle the differences between Python and C/C++ data types. In addition, since IBR-DTN is multi-threaded application and the routing module will be called from within a thread, all functions and data types must be made threadsafe. In addition, since Python itself is not technically multithreaded or thread safe, functions are defined to lock the Python interpreter so that only a single instance is active at a time. The library Boost.Python is used to facilitate this [50]. Boost provides the thread safety mechanisms, function wrappers and data type conversions to simplify embedded Python. The first step is that the Python interpreter must be initialized within the C/C++ application before it is used and this must be done only once. Functions may be defined in a Python module and called from within C/C++ but exceptions must be handled using Boost. In order to use Python in a multi-threaded application, Python uses a Global Interpreter Lock (GIL). Boost provides function to acquire the GIL, call the Python function and when completed, release the GIL.

The main mechanism of the IBR-DTN routing extension is the bundle filter, which is instantiated when a new routing extension is run. When any of several events such as a bundle transfer completed, bundle transfer aborted or new neighbor nodes are discovered, a bundle search is run on the list of existing bundles on the local node. The search uses the bundle filter as a set of criteria to determine what bundle should be sent to the neighboring node. The bundle filter uses bundle metadata such as the destination, hop count, and source as the basic criteria to determine if the bundle should be forwarded to the neighbor. It is here that the results of the machine learning analysis can be used as additional criteria.

The learning function takes the current node EID (the DTN equivalent of a URI or address), the destination EID, the neighboring node EID and a learning specific time index as parameters. It will return a value of 1 if the bundle should be forwarded to the neighboring node and 0 otherwise. This is a simple format that is used for both the classification and cluster based learning methods, such that the IBR-DTN C++ code does not change and only the Python scripts change for the different learning methods. This was done to simplify modifications and allow for experimentation with the learning approach.

In both cases for the clustering and classification methods, the network nodes consist of a central processing node and multiple worker nodes as discussed in Chapter 4. The central processing node will store the global network data received from worker nodes and will perform the calculations to generate the learning model. The central processing node can be a ground station, satellite or any other type of network asset provided that it has sufficient data storage capacity and processing capabilities. What is termed the worker nodes can be mobile nodes such as CubeSats, surface landers, rovers, drones and other network assets. The worker nodes will support primary mission goals by collecting science data or performing automated tasks. The worker nodes will seek to select neighboring nodes to relay data back to a mission operation center as needed.

5.4.1 Multi-label Classification Method Overview

There are several steps to the multi-label classification method consisting of compiling network statistics, developing the classification model, distributing the classification model and routing. An overview of the basic procedures for the central processing node and the worker nodes are given in Algorithms 4 and 5 . Each step will be discussed in this section.

Algorithm 4. *Classification Algorithm Overview for Central Node*

```
Receive network statistics from nodes
Format data for learning
Store data
Select train and test data sets
Select algorithm:
 Base Classifier: NB, KNN or DT
 Multi-label Classifier: CC, ECC, LP
Train model
Test model
Score Results
Disseminate model to all nodes
```

Algorithm 5. *Classification Algorithm Overview for Worker Nodes*

```
Record network statistics
Receive trained model from central node
Upon bundle search events:
 If a valid model is known:
   If bundle passes criteria for epidemic routing:
     Determine time index
     Classify(local id,neighbor id,destination id,time index)
       If bundle should be sent:
         Add to bundle list
              Transmit bundle
 Else
   Route based on epidemic criteria
After period time t, transmit stored network statistics
```

5.4.2 Multi-label Classification Central Processing Procedure

In this classification scheme, a central node is used to store all of the network statistics used for generating the learning models for all of the working nodes in the network. This is done for several reasons. It is anticipated that many nodes may have limited resources in terms of processing speed, power and memory capacity. Since machine learning training is the most computation intensive phase of the learning process, this task will be offloaded to a more powerful aspect of the network. This may be a ground station facility, a satellite or some other node. In this way, all nodes can benefit from the learning enhancements without having to be concerned with storing and processing large amounts of data. In addition, this approach will allow for a more complete view of the network. Nodes will be able to gain information about nodes several hops away in the network without having to directly have had contact with them.

Central Processing Node Train and Test Procedure

The following steps are summarized in Algorithm 4.

- **Receive network statistics from nodes**. Each worker node participating in the learning based routing method will save network statistics based on bundles it has sent, to what neighbor it has been sent, whether the bundle is known to be delivered and the time when it was sent. Nodes can store this data into a batch based on a time period when it will be convenient to transmit to the central processing node.
- **Format data for learning**. Several formatting issues must be handled for the raw data. First, the EIDs of the DTN nodes concerned must be converted into unique numeric identifiers. Next the time stamp must be

converted into an index within the learning epoch. The learning epoch is a time period over which the learning takes place. It is the period of time in which it is expected that a repeated pattern of contacts will exist, such as an orbital period, an Earth day, a Martian day, etc. depending on the network itself. Each time stamp is then matched to an index with the epoch that has been divided into bins of a specific length, dependent again on the network operations. Essentially, the continuous time stamp is digitized to values within a specific length of time.

- **Store data**. The data is now stored in the format of x attributes and y labels. The delivery status, source id, destinations id and time index are store as the x vector attributes. The series of node ids that the bundle has been forwarded to are store as a y label vector.
- **Select train and test data sets**. The data is split into train and test sets using K-fold cross validation with 5 folds. K-fold cross validation is a resampling technique that allows a machine learning model to be trained and tested on a limited amount of data. Data is shuffled and divided in K groups. Each group will be withheld as a test set and the model will be trained on the remaining data. This is repeated for each group, essentially producing K different data sets to train the model on. The training data is then normalized and this pre-processing is also applied to the test set.
- **Select algorithm**. The algorithm and associated parameters is selected. This consists of Decision Tree (DT), Naive Bayes(NB) and K-Nearest Neighbor (KNN) for the base classifier. The multi-label method must also

be selected from Chain Classifiers (CC), Ensemble of Chain Classifiers (ECC) and Label Powerset (LP).

- **Train model**. The model is trained using the training folds.
- **Test model**. The model runs predictions on the x data attributes of the test data set.
- **Score Results**. Metrics are calculated on the true values and predicted values of the test data set.
- **Disseminate model to all nodes**. If the metrics show a successful update of the model, the model is saved to a Python Pickle archive and transmitted to each participating worker node for use.

5.4.3 Multi-label Classification Worker Node Routing Procedure

This section covers the process that will be followed by the participating worker nodes. Initially any node can use the epidemic routing method and can revert to this method if needed (there is some problem with the learning model). In this case, a bundle will be forwarded to any node encountered unless the neighbor node already has a copy of this bundle.

Worker Node Routing Procedure

The following steps are summarized in Algorithm 5.

- **Record network statistics**. This initial phase is used to collect data on what paths were taken from a source node to a destination at a particular point in the learning epoch that resulted in successful delivery. Epidemic routing or a previous learning model can be used for the routing decisions. The data is saved until a convenient time occurs to transfer it to the central processing node. This can be on a reoccurring schedule or as needed.

- **Bundle search event**. When the routing extension receives an event which triggers a search for bundles to send, either epidemic routing or the learning model can be used.
- **Evaluate criteria for epidemic routing**. The bundle should not already exist on the neighboring node if it is to be forwarded and not destined for the current local node (local delivery).
- **Determine time index**. The current time stamp should be converted into a time index within the learning epoch.
- **Classify**. The learning prediction function is called with the local id, neighbor id, destination id, time index as parameters. The function will perform multi-label classification which will determine a set of nodes that will most likely lead to the bundle being delivered to the destination. If the neighboring node is within the set, the function will return true.
- **Bundle list**. If the neighbor node is predicted to deliver the bundle, it is added to the bundle list that will be queued for transmission to the neighbor.
- **Transmit bundle**. The bundle is sent to the neighboring node.
- **Transmit statistics**. The recorded statistics are transmitted as needed to update the model. In this way the model can be continuously updated and improved. Statistics recorded include when and to where a bundle was sent from a source node, when and where bundles are forwarded, and when bundles are delivered. The statistics from each node

are compiled together to create a complete trace of a bundle through the network. In addition, the total number of bytes sent and received, the number of replicated bundles, the number of queued, expired and aborted bundles are also reported at the end of each emulation. The statistics used are covered in more detail in Chapter 7.

5.4.4 Clustering Method Overview

The section covers the process that nodes will follow to use location clustering to make routing decisions. The central and worker nodes perform similar functions as within the classification method with a few differences as listed in Algorithms 6 and 7. The steps of acquiring data and disseminating the learned model are the same as in Algorithm 4, however the analysis performed on the data is K-Means clustering rather than the classification methods discussed in the previous section. Similarly, the worker nodes will execute a similar routing procedure, however the cluster regions will be used to determine which bundles should be sent to each neighbor. This is intentionally done since all of the routing software is highly modular and object oriented, so the overall routing framework, objects and events are reused.

Algorithm 6. *Clustering Algorithm Overview for Central Node*

```
Receive network statistics from nodes
Format data for learning
Store data
Cluster data based on location
Disseminate model to all nodes
```

Algorithm 7. *Clustering Algorithm Overview for Worker Nodes*

```
Record network statistics
Upon bundle search events:
 If a valid model is known:
   If bundle passes criteria for epidemic routing:
     Determine time index
     QueryClusterData(neighbor id, destination id, time index)
       If bundle should be sent:
         Add to bundle list
             Transmit bundle
 Else
   Route based on epidemic criteria
After period time t, transmit stored network statistics
```

Algorithm 8. *QueryClusterData Function*

```
t = current time index
Get list of clusters A for neighbor during t until $t+\Delta t$
Get list of clusters B  for destination during t  until $t+\Delta t$
If A and B are not disjoint:
 Return True
Else
 Return False
```

5.4.5 Clustering Method Central Processing Procedure

The central processing node performs the task of storing sampled locations from the other nodes in the network. The steps outlined in Algorithm 6 are explained below.

Clustering Method Central Processing Procedure

- **Receive network statistics from nodes**. In this case, the nodes will store their local id, their location and current time at a specified sampling interval. This data will be transmitted to the central processing node at a convenient time as described above.
- **Format data for learning**. The data is formatted according to the node id, time index and location coordinates.

- **Store data**. The data is stored in a central database.
- **Cluster data based on location**. The locations are clustered using K-means clustering. A cluster id is matched back to the associated node id and time index.
- **Disseminate model to all nodes**. The cluster id, node ids and time indices is transmitted to each node. Each node now has an idea of where every node is in the network based on cluster location region.

5.4.6 Clustering Method Worker Node Routing Procedure

Once the worker nodes have obtained a list of the cluster regions and nodes belonging to them based on time, this information can be used to make routing decisions as outlined in Algorithms 7 and 8.

Clustering Method Worker Node Routing Procedure

- **Record network statistics**. This initial phase is used to collect data on where each node is located at a particular point in the learning epoch. Epidemic routing or a previous learning model can be used for the routing decisions. The data is saved until a convenient time occurs to transfer it to the central processing node. This can be on a reoccurring schedule or as needed.
- **Bundle search event**. When the routing extension receives an event which triggers a search for bundles to send, either epidemic routing or the learning model can be used.
- **Evaluate criteria for epidemic routing**. The bundle should not already exist on the neighboring node if it is to be forwarded and not destined for the current local node (local delivery).

- **Determine time index**. The current time stamp should be converted into a time index within the learning epoch.
- **Query Cluster Data**. The learning prediction function is called with the local id, neighbor id, destination id, time index as parameters. The function will query the current and future locations from time $t + \Delta t$ for the neighbor node and destination node, where t is the current time index and Δt is a configurable duration of time. If the neighbor node will enter the cluster region of the destination node within that time, the function will return true, meaning the neighbor is likely to deliver the bundle.
- **Bundle list**. If the neighbor node is predicted to deliver the bundle, it is added to the bundle list that will be queued for transmission to the neighbor.
- **Transmit bundle**. The bundle is sent to the neighboring node.
- **Transmit statistics**. The recorded statistics are transmitted as needed to update the model. In this way the model can be continuously updated and improved.

5.5 Machine Learning Libraries

Two Python libraries were used to perform the learning algorithms, data preparation and evaluation of metrics. Scikit-learn [38] was used for the base classifiers (Decision Tree, Naive Bayes, and K-Nearest Neighbors). Functions to normalize the data and split it into training and test sets were also used. Scikit-multilearn [39], which is based on Scikit-learn was used to perform the multi-label classifications (Chain Classifiers, Ensemble of Chain Classifiers, and Label Powerset).

The details of the algorithms are discussed in Chapter 2 Background. The metrics used to score the classifier performance were also provided by scikit-learn and are discussed in Chapter 2 and Chapter 7 Results. Scikit-learn also provided the K-means clustering algorithm.

6 Simulation and Emulation

This chapter covers specifically the techniques and tools that were used for the simulation and emulation environments, how the DTN network was emulated and details of the automation tool chain. This was a challenge of studying DTN routing, which was how to create an environment to test the DTN software, obtain data to perform machine learning and evaluate routing performance. The goals for this study were to obtain the following:

- Model the behavior of at least 10s of nodes (20 or more)
- Execute actual DTN software packages rather than models of protocols
- Allow node mobility and link behavior to be scripted or adjusted dynamically
- Provide a simple way to obtain network and performance statistics for analysis
- Provide a graphical interface for easy configuration
- Allow for scenarios of data transmission and reception to be easily scripted and automated.

In order to achieve these goals, several open-source tools were evaluated for the simulation and test environment as well as the basis for software development.

Initial simulations of a Q-routing reinforcement learning based router was developed using OMNeT++ [51]. OMNeT++ is an open-source discrete event simulator that has a generic framework which can be used to develop simulations of communication networks, multiprocessors and distributed hardware systems and protocol modeling. Next, routing tests were done using DTN2 [25] bundle protocol reference implementation and the NASA DTNbone [52], which is a network of multiple Ubuntu Linux virtual machines. Finally CORE (Common Open Research Emulator) [53] was used to emulate a network of multiple mobile nodes.

6.1 Simulation Environment

OMNeT++

The simulation experiments were done using OMNeT++. OMNeT++ is a discrete event simulator that has a generic framework that can be used to develop simulations of communication networks, multiprocessors and distributed hardware systems and protocol modeling. OMNeT++ uses a component based architecture defined by C++ classes. Components may be provided as base classes by OMNeT++ or developed by the user. External interfaces between the modules are defined in network description (NED) files. The connections between two modules are also described in NED files. OMNeT++ uses a message class to pass information between modules. The messages can be further defined by the user to represent network packets, status and timing internal messages or any other

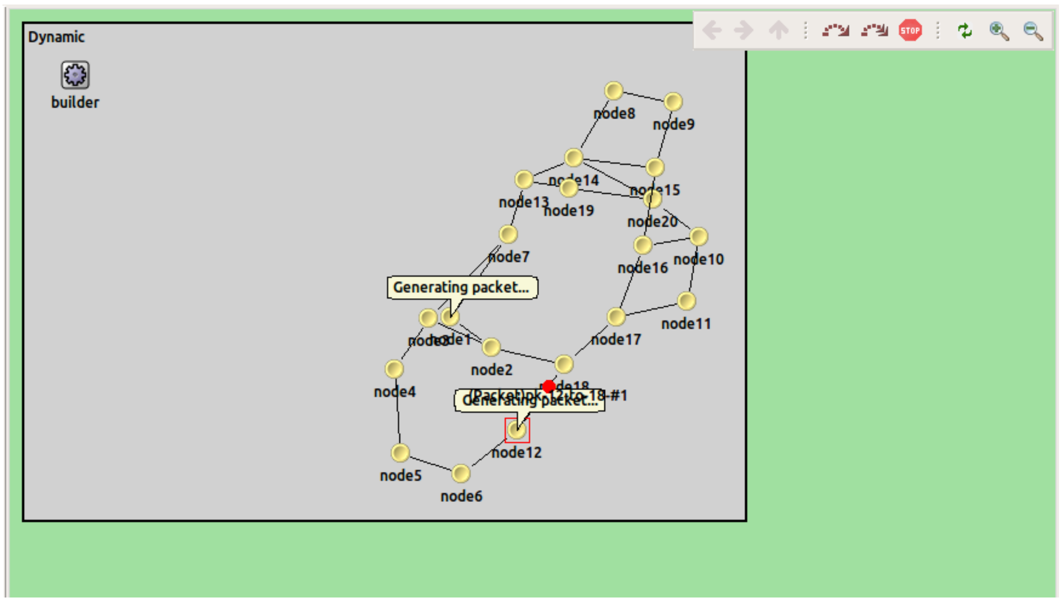


Figure 6.1. OMNeT++ Simulation Environment

type of message the user requires. OMNeT++ also provides a signaling and statistics framework so that users can collect and analyze data and statistics from the simulation. Figure 6.1 shows the OMNeT++ graphical interface.

There are several popular network simulators that have been modified to simulate the characteristics of DTNs (mobile nodes, long delay times, asymetric links, intermittent connectivty, for example) or have been created with DTNs and opportunistic networking specifically in mind. The simulators considered in this work are OMNeT++ [51], the ONE simulator [54], and ns-2/ns-3[55]. While OMNeT++ is a more of general purpose networking simulator, several related works have used it as the basis for DTN or satellite network simulation [56], [32], [57], [58]. Initial work was completed using OMNeT++ [32], however it was found that significant customization would be required in order to create a higher fidelity model of the delay tolerant networking characteristics essential to the routing problem, as well as to execute/simulate any of the well known DTN implementations' software.

The ONE

The ONE (Opportunistic Networking Environment)[54] simulator is a discrete event simulator written in Java, designed especially with delay tolerant and opportunistic networks in mind. It has been used quite extensively for simulating DTNs and studying routing protocols [59–62]. The ONE models node movement, inter-node contacts and messaging, and routing. Lower level networking details and operating system specifics are abstracted away. Each node has a simplified model of the radio interface, persistent storage, energy consumption as well as message routing and movement models. The radio interface and storage are configured through several parameters such as data rate, communication range and storage capacity. There is extensive flexibility for custom routing protocols and movement models, as well as the inclusion of several well known routing protocols and mobility models. The built-in routing protocols provided with the ONE installation include First Contact, Direct Delivery, Epidemic, Spray-and-wait, Max-Prop and PRoPHET. New routing modules can be user defined based on the existing routing protocols provided. A new routing protocol will inherit simple buffer management and called backs for message events from the provided MessageRouter module.

The ONE provides several mechanisms for simulating node mobility. The synthetic mobility models can be characterized as either random movement models, map-based random movement and human patterns of mobility. Included with the installation are the Random Walk and Random Way Point mobility models, as well as random map based mobility, shortest path map-based mobility and routed map-based mobility. In addition, an interface is provided for either

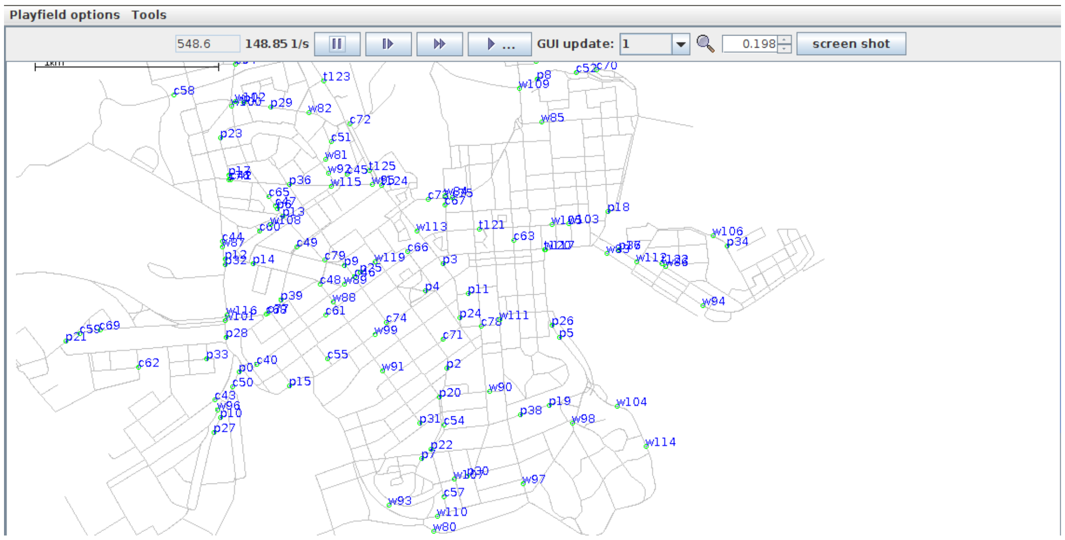


Figure 6.2. The ONE Simulation Environment

creating custom synthetic mobility models, as well as for importing real-world mobility traces[54]. Figure 6.2 shows the ONE graphical interface.

While extensive work has been done in the DTN community using ONE, it was decided that there were several implementation specific hurdles regarding its use. Currently developed software cannot be directly executed within the ONE. Some type of interface would need to be created to allow C/C++ software to be executed using the ONE simulation. Several papers discuss this process, involving using the Java Native Interface (JNI) wrappers and abstracting away lower level libraries used by the DTN software (ION in this case) [59].

ns-2/ns-3

The simulator/emulator ns-3 is the updated version of the simulator ns-2. Both have widely been used as general networking simulators and as the basis for simulations and emulations of DTNs and satellite networks [63–67]. The current ns-3 package can perform as a discrete event simulator and also has a real time operating mode. In this way, it can interface to a physical network or interconnect virtual machines as a channel emulator [68]. The ns-3 simulator has been used as a standalone simulation/emulation environment and has also been integrated with other packages such as CORE (Common Open Research Emulator)[69] and Systems Tool Kit [70], [64].

6.2 Emulation Environment

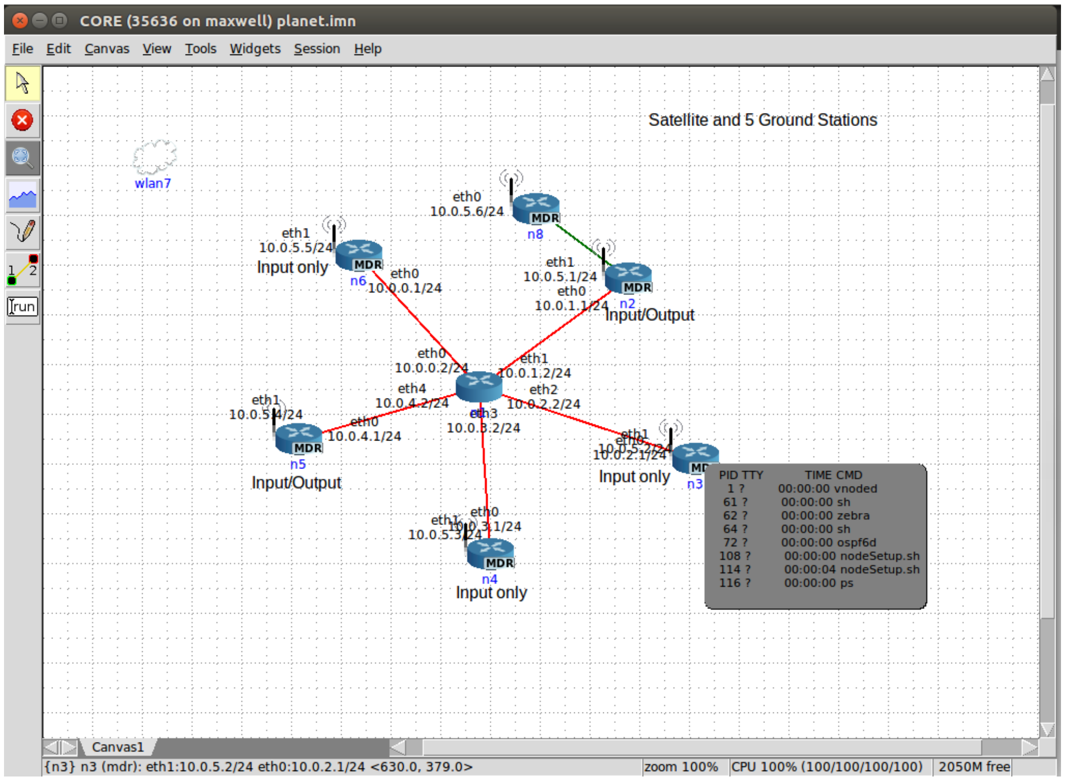
This section discusses the emulation environment used to model the network scenarios. Several tools are discussed. STK (Systems Took Kit)[71] was used to compute node access times according to their orbital parameters and/or locations. CORE (Common Open Research Emulator)[72] was used to emulate network layers 3 and above (network, transport, session, application). EMANE (Extendable Mobile Ad-hoc Network Emulator)[73] was used to model the data link and physical network layers. Sdt3d (Scripted Display Tools) [74] was used for 3-dimensional network visualization. This section also provides an overview of other simulation/emulation tools that are often used in DTN research and considered as potential test environments.

Many network simulators abstract the operating system and network protocols into a simulation model to perform a statistical analysis of the network [75]. Simplification of the underlying protocols and hardware provide an estimate of the performance of the particular networking scenario. This allows for more complex scenarios with a greater number of nodes to be evaluated on a single host system. In addition many scenarios may be evaluated relatively quickly since simulations can often be executed in faster than real time [70]. Emulation provides a more in depth analysis of the software and protocols used, since only the hardware is modeled. Emulations run in real time and may require a more powerful host machine or multiple host workstations.

CORE and EMANE together emulate the complete network stack of the network nodes and simulate the links connecting them. In this way a very realistic environment is created for testing software applications and protocols while requiring only a single host machine rather than a hardware test bed of multiple nodes. In addition, CORE uses lightweight Linux containers which reduce the processing and memory consumption per node when compared to complete virtual machine with a full installation of a guest operating system. This allows for more nodes and more complex scenarios to be executed on the host system and places fewer demands on its hardware resource requirements.

6.2.1 CORE and EMANE Network Emulation

Differing from the discrete event simulators discussed in the previous section, CORE and EMANE are two open sourced emulation tools that have been developed for network modeling and emulation. Both tools were developed by the U.S. Naval Research Laboratory and are written in a combination C++ and Python. The tools can be run separately and have also been integrated together to provide emulation of the complete network stack. CORE emulates layers 3 and above, while EMANE emulates the data link and physical layers. Both tools run on Linux and utilize several features of the Linux kernel version 2.6.24 or newer (Linux namespaces/containers, Ethernet bridging, and the TUN/TAP device driver)[76]. Figure 6.3 shows an screen shot of the CORE GUI with a network scenario from the MITRE DTN Development Kit [77].



```
PID TTY         TIME CMD
  1 ?       00:00:00 vnoded
 61 ?       00:00:00 sh
 62 ?       00:00:00 zebra
 64 ?       00:00:00 sh
 72 ?       00:00:00 ospf6d
108 ?       00:00:00 nodeSetup.sh
114 ?       00:00:04 nodeSetup.sh
116 ?       00:00:00 ps
```

```
{n3} n3 (mdr): eth1:10.0.5.2/24 eth0:10.0.2.1/24 <630.0, 379.0>
```

Figure 6.3. CORE Screen Shot [77]

CORE

CORE is an open source network emulation environment that provides a convenient graphical interface and Python based API for creating virtual networks of mobile nodes. The CORE graphical interface provides several tools for network and process visualization and monitoring. Nodes are arranged on the GUI canvas where pixel locations map to a user defined meters-to-pixel scale. Node locations can be read from a mobility script file or sent dynamically through Python commands.

CORE provides a graphical interface for accessing and configuring nodes by clicking on the node icon. A Linux terminal is launched on the virtual machine from which commonly used Linux command, scripts and applications can be launched just as they would be on the host computer. In addition, CORE provides several built-in routing protocols and also the ability to launch custom scripts and protocols at start up. It provides several tools which show system status information such as running processes, routing adjacency neighbors, IP configuration information and network traffic by simply hovering the mouse over the node icon. Packet capture programs such as tcpdump and wireshark can be launched from each virtual node as well.

A screen shot of the CORE graphical environment is shown in Figure 6.3. Nodes in the network are shown with a router icon and display the network interface name and IP address of the node. Each node in the emulation may be thought of as corresponding to a satellite, surface vehicle, ground station, or any other type of device that might function as a DTN node in a real network. Communication ranges, bandwidth and delays can also be configured for each node. Nodes may communicate on a wireless local area network or they may have direct wired connections. Each of these emulated nodes are created from a Linux container (LXC) [53] and function similarly to a light weight virtual machine. A temporary file system is created for each node based on user defined directories, either representative of the host machine or directories that are specific to the virtual node. This virtual machine can be thought of as functioning like the flight computer/avionics of a real node, which will execute the DTN software, generate and process data, create system logs and provide an administrative interface.

Linux containers are not a complete virtual machine but rather a light weight virtual machine with its own isolated resources (CPU, memory, block I/O, network allocation) using a resource control mechanism (cgroups) that run on the same host system kernel [78]. This is accomplished using the namespace and control group features of the Linux kernel. Since each node functions similarly to an individal virtual machine,software applications and protocols can be executed unmodified, just as they would be on a physical machine. This removes the need for the use of built-in simulator models and classes, or wrappers and interface software often required to run various protocols packages in many simulator environments [59].

The LXC virtualization used by CORE differs from machine virtualization tools such as Virtual Box and VMware in that it does not run complete instances of an operating system on a host computer of the same or different operating system. Linux containers are primarily used to isolate multiple instances of the same Linux server environment on the same physical machine. Processes are isolated within their own namespace with a separate stack but hardware such as disks, video cards are not emulated but shared between nodes [75]. CORE uses the Linux tc (traffic control) and netem (network emulation) features to create basic network characteristics such as delay, loss and bandwidth limitations. Wireless networks are emulated using Linux bridging. All node interfaces are connected to the same WLAN network, using Linux bridging firewalls (ebtables) to provide rules that will either restrict or allow network traffic between nodes.

EMANE

EMANE provides emulation of the lower network layers (physical and data link). This allows predefined radio models or custom defined models to be used to determine when emulated nodes have communication access to each other. This creates a higher fidelity emulation of real radio hardware, rather than the simplified communication parameters of CORE (range, bandwidth and delay). Figure 6.4 shows the integrated architecture of CORE and EMANE.

EMANE is based on network emulation modules (NEM) which are processes and shared libraries configured through XML files [53]. The modular style of the MAC and PHY models provides a framework for developers to implement custom radio models if desired. EMANE also includes an 802.11 model and a generic RF pipe model with configurable parameters. One or more NEMs may run on a single host machine. Each NEM has an OTA (Over-the-Air) adapter which interfaces with an overall OTA manager that allows communication between each NEM.

EMANE provides a raw transport interface using packet capture from a real world interface or virtual interfaces can be used deliver packets to and from the emulated network. Virtual interfaces utilize the Linux TUN/TAP device framework. TUN/TAP devices are software based interfaces that allow user space programs to see raw network traffic [79]. They appear to applications as a regular networking interfaces but exist only in the kernel. When the kernel would normally read/write data to the hardware file descriptor, it is instead accessing another user space application rather than a hardware network device.

To integrate the operation of CORE and EMANE into a single emulation environment, CORE uses run-time stages in order to account for the timing of bringing up various processes required to perform the network virtualization. The TUN/TAP device consists of a user space side and a kernel space side.The user space side is implemented as a socket connected to a process that can be written to and read from the user application. The kernel side of the TUN/TAP device appears as any other network device and is installed into a namespace, which hides it from the host machine, making it visible only within the namespace.For this reason the EMANE user-space process must open the user-side socket to the device before it is installed in the namespace. To simplify this, CORE enforces multiple runtime stages, such as edit (stopped), configuration, instantiation and execute (start)[53]. The EMANE and CORE emulation run within the back-end CORE daemon process. The CORE GUI runs as a separate process which sends event signals about the node and link configuration to the daemon.

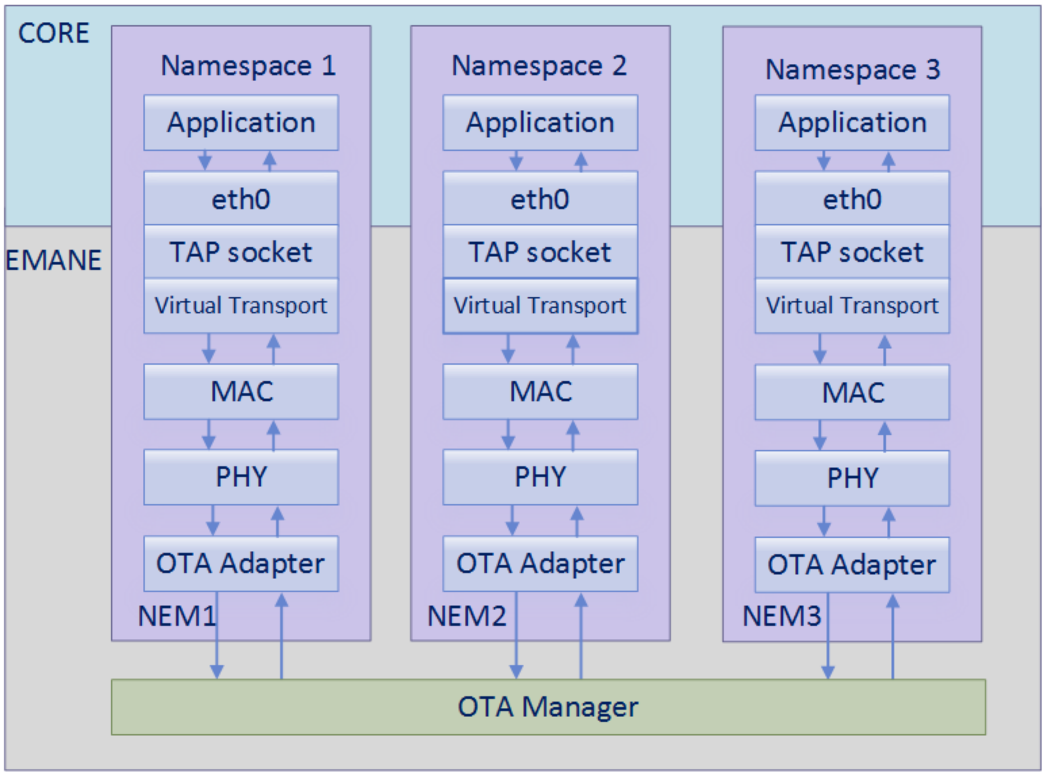


Figure 6.4. CORE/EMANE Architecture[53]

STK

The Systems Tool Kit [71] is used to model the orbital characteristics planets, satellites and other objects. It provides a variety of freely available functions as well as paid license plugins. The base functionality allows for access to a database of planets, ground stations and air/spacecraft orbital data. The ephemeris data can be exported in a variety of formats and calculations are performed to determine the line-of-sight access between two nodes.This work uses only these base capabilities of STK, in conjunction with the radio interface emulation of EMANE to determine when nodes in the network are accessible to each other.

6.3 Scenarios

This section discusses the details of several scenarios that were developed using the tools described in the previous section. The first scenario is based on the International Space Station communicating with three TDRS (Tracking and Data Relay Satellite). The next scenario uses 10 nodes with a Random Walk mobility model. Finally, real world GPS traces are used from the ZebraNet [80] experiment for an emulation of 10 nodes. These three scenarios provide a variety of conditions spanning from a more realistic scenario in low earth and geosynchronous orbit, to an abstract mobility model and finally traces from a real world experiment.

6.3.1 International Space Station

This scenario models the International Space Station (ISS) in low earth orbit, three TDRS satellites in geosynchronous orbit and the earth ground stations located at the White Sands Complex (WSC) and Guam Remote Ground Terminal (GRGT). The ISS periodically has access to each of the TDRS which then relay data to either the White Sands Complex or Guam Remote Terminal. Since the TDRS are in geosynchronous orbit, they have constant access to the ground station.

The scenario was constructed using several tools. First the locations of each node are computed using STK, based on standard objects available from the STK database. STK performs orbital analysis to determine a satellite's location given a time of the year and analysis duration. Next since the locations are known, the access that each node has to each other is also computed by STK. ISS to the TDRS has multiple intervals of communication, while the access from TDRS to the ground is constant as shown in Figures 6.5 and 6.6.

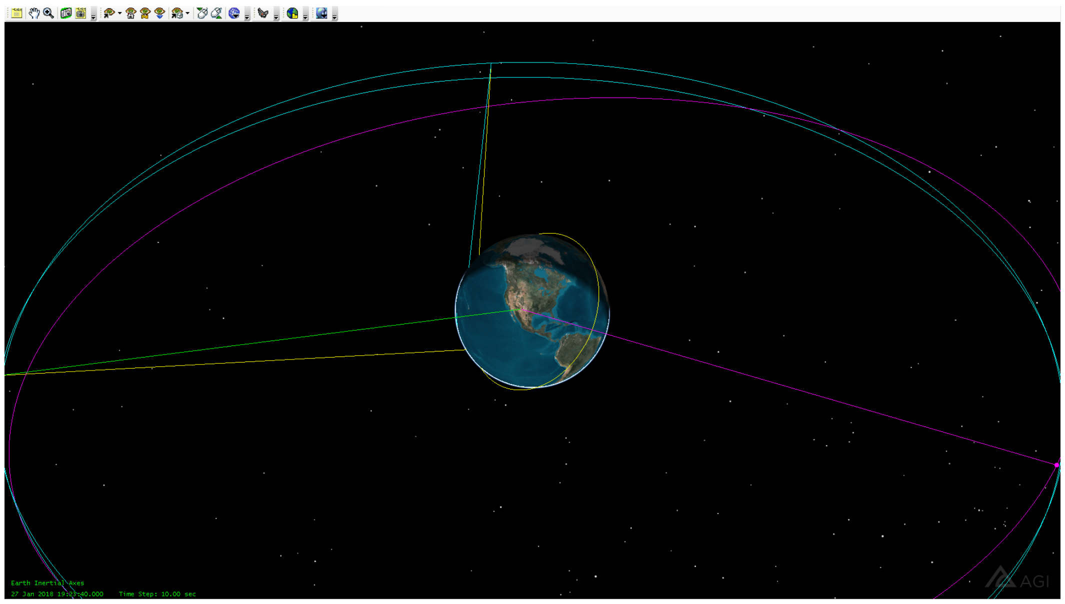
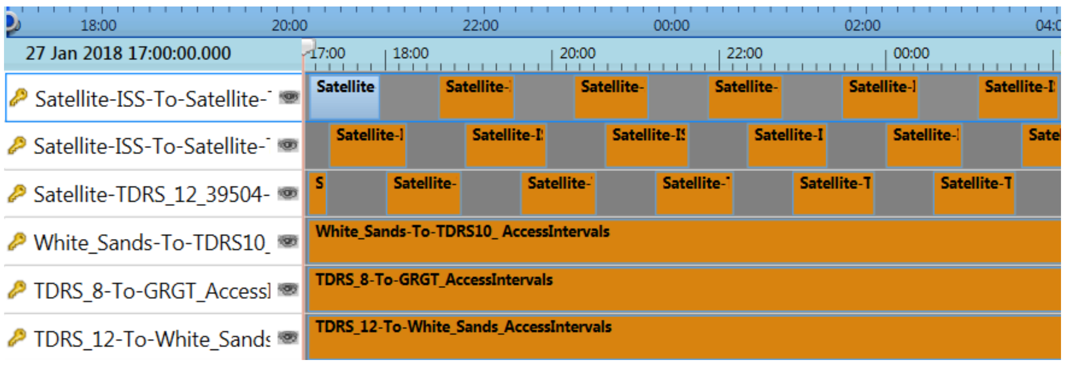


Figure 6.5. STK Analysis for ISS Scenario

Once the orbital analysis has been performed by STK, the latitude and longitude (degrees) and the altitude (km) over every one minute interval can be exported to a comma separated file. In addition, the access times and duration between the ISS and TDRS are exported to a comma separated file as well. These are then converted to a format suitable for sdt3d [74]. The node positions are scripted using the longitude, latitude and altitude values. The access times are used to create node link and unlink commands for both sdt3d and CORE. In this way a 3D visual animation is created using sdt3d and the network emulation is performed by CORE. Commands are able to be sent between both sdt3d are

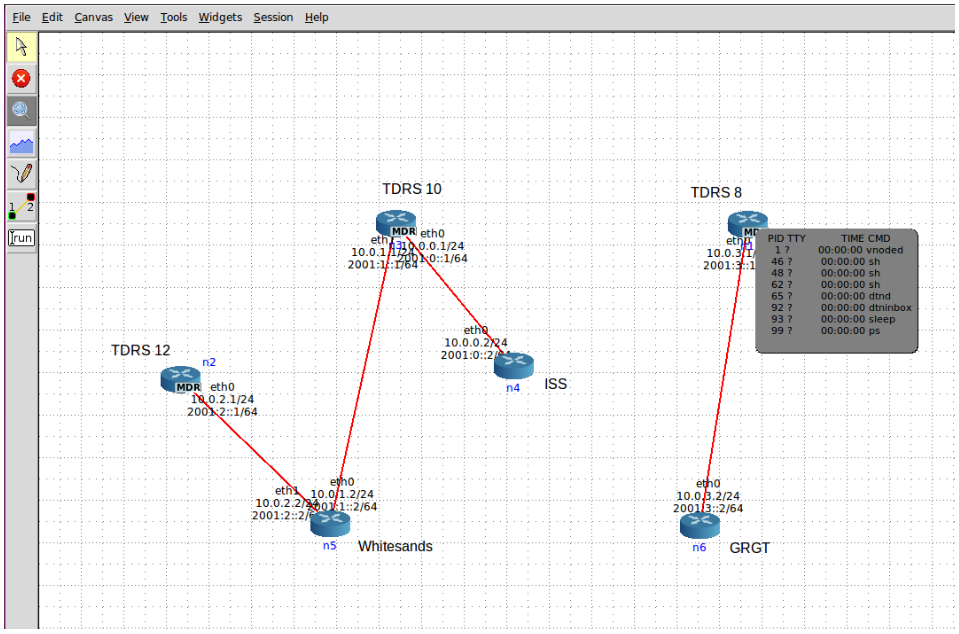
CORE using Python and TCP ports which each application (sdt3d and CORE) listen to for commands.



18:00
20:00
22:00
00:00
02:00
27 Jan 2018 17:00:00.000
Satellite-ISS-To-Satellite-
Satellite-ISS-To-Satellite-
Satellite-TDRS_12_39504-
White_Sands-To-TDRS10_
TDRS_8-To-GRGT_Access
TDRS_12-To-White_Sands
White_Sands-To-TDRS10_ AccessIntervals
TDRS_8-To-GRGT_AccessIntervals
TDRS_12-To-White_Sands_AccessIntervals

Figure 6.6. STK Access Times for ISS Scenario

The actual network emulation is performed by CORE. In this case, there are 6 virtual machines the represent ISS, the three TDRS and two ground stations. The links between TDRS and the ground stations are constant. The links between ISS and the three TDRS are controlled using a Python script which reads the time and duration from the exported access files and uses the CORE python API to link and unlink the appropriate nodes. In this case, the node position within CORE is not used to determine which nodes are able access on another. The rest of the emulation process including automated data generation, bundle protocol store and forward and routing are covered in the next section as this procedure is the same for all scenarios. Figure 6.7 and 6.8 show the emulation in CORE and the corresponding 3D display in sdt3d.



```
PID TTY          TIME CMD
  1 ?        00:00:00 vnoded
 46 ?        00:00:00 sh
 48 ?        00:00:00 sh
 62 ?        00:00:00 sh
 65 ?        00:00:00 dtnd
 92 ?        00:00:00 dtninbox
 93 ?        00:00:00 sleep
 99 ?        00:00:00 ps
```

Figure 6.7. ISS Scenario in CORE

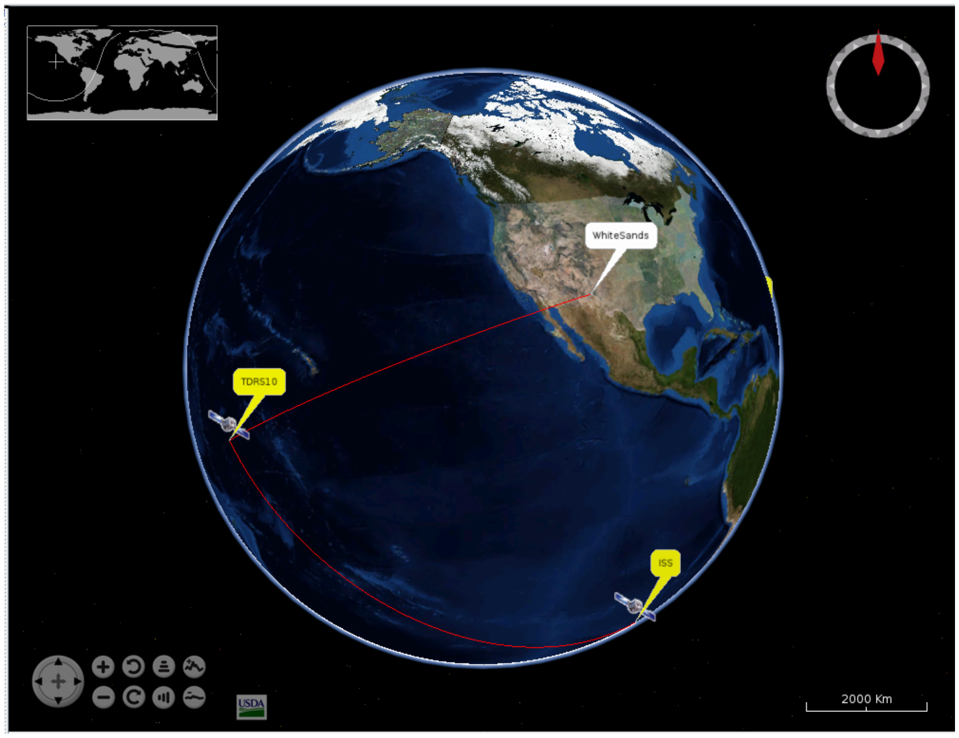


Figure 6.8. ISS Scenario in sdt3d

6.3.2 Random Walk

This scenario consists of 10 mobile nodes. In this case nodes can be thought of as abstract mobile nodes, or perhaps mobile node in a wireless sensor network. The node locations are determined based on the Random Walk mobility model. Boundaries for the grid locations, number of nodes and speed are given to the tool BonnMotion which will generate location coordinates for each node over the duration of the emulation. In this case, the locations simply represent pixel locations which CORE will scale to kilometers to determine connectivity between nodes based on distance and the communication range set for the nodes. This scenario was considered since the Random Walk model is frequently used in other works studying DTN routing and mobile nodes. In addition, this provides an interesting data point to determine how well the classifier will be able to predict future routes as discussed in Chapter 7. It is a significantly more complicated routing problem than the ISS scenario, and for this reason is studied more in depth in Chapter 7.

6.3.3 ZebraNet Traces

This scenario also uses 10 nodes. Node mobility was generated from GPS traces from the ZebraNet experiment. This scenario models any type of opportunistic DTN network in which nodes follow a non-random pattern of movement such as humans, vehicles or animals. In this case CORE uses the GPS locations to determine the distance between nodes and the range of the communication links to determine when nodes have access to each other. The coordinates are based on the UTM system rather than longitude and latitude. This is a grid based system which maps easily to the pixel locations of CORE once a scaling factor has been set to adjust the screen appearance of the nodes. This scenario was used for the majority of testing since it is complicated enough to provide a challenging routing environment and is also more realistic since it is based on real world mobility traces.

6.4 Tool Chain and Automation

This section discusses the use of CORE and other tools to develop a DTN emulation environment to test the routing software. There are several aspects to be addressed in creating an automated test environment. Figure 6.9 shows an overview of the emulation tool chain.

Inputs to the emulation process are the number of nodes and the data files to send. Several scripts and configuration files are then created for each node to control the behavior of IBR-DTN, as well as node mobility and data transmission. The emulation will execute for a selected amount of time and generate network data logs to be used as the input to the learning models. Once trained on the data from previous emulations, the routing module can use this information to make predictions about best paths through the network. The steps of the emulation are described in detail in the remainder of this section.

- **Node configuration**. The first step of using CORE is to develop an emulation configuration. For the most basic scenario, this includes selecting the number and type of nodes, selecting the number and type of each network interface (wired or wireless), configuring the link behavior (based on EMANE radio model or range information), and selecting the protocols and services used by the nodes. CORE has a number of

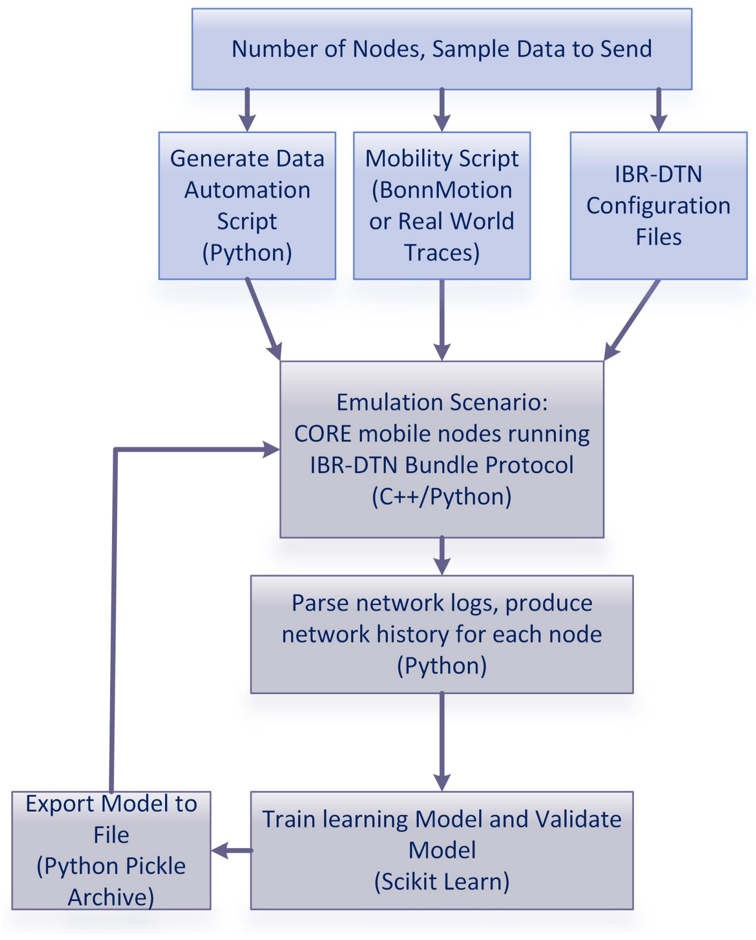

Number of Nodes, Sample Data to Send
Generate Data Automation Script (Python)
Mobility Script (BonnMotion or Real World Traces)
IBR-DTN Configuration Files
Emulation Scenario: CORE mobile nodes running IBR-DTN Bundle Protocol (C++/Python)
Parse network logs, produce network history for each node (Python)
Train learning Model and Validate Model (Scikit Learn)
Export Model to File (Python Pickle Archive)

Figure 6.9. Emulation Tool Chain

predefined services for standard network operation such as OSPF, OLSR, SSH, FTP, DHCP, HTTP and many others. To emulate a DTN, CORE must be configured with a custom service, so that routing and discovery will be performed using IBR-DTN. The DTN Development Kit [77] for CORE provides examples using ION. This served as the basis of the emulation scenario, however nodes were configured with IBR-DTN rather than ION.

The CORE emulation utilizes a temp directory that is created at run time. This will serve as the basis of the file system for the emulated nodes. Prior to run time, the nodes' directory structure and content must be configured. In the configuration folder, each node has a directory that will serve as the root directory of the emulated nodes. Files needed for start up can be stored here. Shell scripts are used to link the content of the configuration directories to the temporary emulation directory and to set any necessary environment variables. Any needed start up scripts can be launched from here as well.

Specific to the emulation of IBR-DTN, the IBR-DTN configuration file is needed for each node. This contains information about the nodes addresses and EIDs, what type of storage to use and storage limits, what type of routing to use, static routes, and to enable or disable functionality such as IPND discovery. Each node has a unique configuration.

- **Mobility Scripts**. The next step of the emulation is to enable node mobility. This can be accomplished several ways. CORE can accept node positions based on longitude, latitude and altitude or in UTM (Universal Transverse Mercator) format, which is a grid based coordinate system. CORE provides a Python interface such that nodes can receive coordinate updates over a socket interface. Mobility scripts can also be generated from a scenario file which contains the node id, simulation time and coordinates for the entire emulation. This file is based on the NS2 mobility format. For a very lengthy or complex scenario, it is not practical to generate these coordinates by hand. Instead the file can be generated automatically using several tools.

  One tool which is very helpful in generating mobility scripts is Bonnmotion [81]. BonnMotion consists of a variety of stochastic mobility models such as Random Walk, Random Waypoint, grid models and others. The user can input the desired number of nodes, length of emulation time, node speed, the desired model to use and other parameters. BonnMotion will then generate the coordinates for each node in a variety of formats, include NS2. Another option for mobility scripting is to use traces from real world experiments. Dartmouth College hosts an archive of traces from wireless experiments called CRAWDAD. This work used the ZebraNet [80] dataset that is archived on CRAWDAD. This dataset contains the GPS coordinates from a DTN experiment in which tracking collars were fitted to multiple zebras in the wild and their locations were recorded. This allows for a more realistic pattern of mobility, as opposed to models which have synthetically generated coordinates.

- **Traffic generation**. In order to create a realistic environment, data must be exchanged between nodes to test the routing performance. A Python script is started at run time to send a tar file in bundle format to a randomly selected destination at a random interval of time. Each node executes this script and also uses the IBR-DTN tool dtninbox to wait for files that are inbound to the local node.
- **Emulation scenario**. All of the above mentioned scripts are executed at start up to create the emulated nodes, launch and configure the IBR-DTN daemon, run the node mobility scripts and generate network traffic. As nodes generate, forward and receive bundles, these events are captured in a log to trace the paths of the bundles through the network. The status of the bundle storage and number of bytes sent and received from the convergence layer are also periodically sampled.

- **Data preparation**. Once the emulation run-time has expired, the network logs must be prepared and formatted to be used for machine learning. Each bundle that was sent is traced through the logs using the unique bundle id which is the bundle creation time and the bundle source EID. In this way the path of each bundle is determined with a specific time stamp for creation, forwarding and delivery. This data is then formatted into the attribute data of time stamp, source node, destination and labels of known paths in the network. The node coordinates for clustering are known from the mobility scripts. This data is formatted into SQLite data tables which are then read into pandas (Python Data Analysis Library) [82] data frames for processing.
- **Training phase**. Once the data has been formatted in the vector of attributes and vector of labels, it is read into pandas [82] data frames and converted into NumPy [83] (Python library for data structures and mathematics) arrays for mathematical manipulation. The data is divided into train and test sets using K-fold cross validation as discussed in Chapter 5. The data is normalized as well. The selected machine learning algorithm is executed on the training data sets. Next the generated model is used to predict the labels of the test set. The test set's predicted labels (the classifier output) are scored against the true values. If the model appears promising from the metric scores, it will be exported for the worker nodes to use. If it is not, more data can be collected and processed to improve the model, different parameters specific to each model can be changed or the algorithm itself can be changed.

- **Export model**. Once a satisfactory model has been achieved, it is archived in a Python Pickle. This is a python specific format that allows the structure and data of Python objects to be saved to a file. In the case of Scikit learn models, this is generally a series of weights. This allows the trained model to be accessible to the emulated nodes, without having to actually perform the training calculations. This model can then be extracted back into a Python object and used by SciKit learn to predict new label values.

The next section will detail the experiments that were performed using OMNeT++ and CORE. The metrics used for machine learning as well as routing are discussed. The results of the experiments are analyzed using these metrics.

7 Performance Measurements

This chapter covers the experimental emulations that were conducted, the results of each of those studies and the metrics which are used to analyze the performance of the machine learning and routing algorithms. There are several routing metrics and statistics which are used to evaluate the effectiveness of the DTN routing overall. In addition, prior to using the machine learning model for routing, there are several metrics that are used to determine how well the model is able to make correct predictions. The next subsections will cover both of these classes of metrics in detail.

Following the discussion of metrics, an initial survey of the performance of several DTN routing algorithms was completed using DTN2. This was a foundational step in working with the DTN implementations available, determining the appropriateness for continuing work as well as the concepts for developing a DTN emulation environment. Next, the Q-Routing algorithm was simulated using OMNeT++. Finally, CORE was used to create a higher fidelity emulation environment to perform the development and testing of a classification based and cluster based routing algorithm.

7.1 Metrics

7.1.1 DTN Routing Metrics

There are several statistics that were collected using the IBR-DTN API [49]. These metrics are based on counters of various events within the bundle storage and routing modules.

- **Bundles Queued**. This is the number of QueueBundleEvents that have occurred during the emulation. When a new bundle is received from a local application or a neighboring node, the bundle is forwarded to the routing module which generates a QueueBundleEvent. The bundle must be a unique bundle, not a copy of a bundle that has already been received. The value is incremented throughout the emulation and is the total number of bundles that were queued.
- **Bundles Requeued**. This is the number of RequeueBundleEvents that have occurred. If a transmission fails with a temporary error, the bundle may be requeued.
- **Bundles Aborted**. This is the number of TransferAbortedEvents that have occurred. This event occurs when the transmission of a bundle is aborted due to an error condition in the connectivity of a neighboring node.
- **Bundles Transmitted**. This is the number of TransferCompletedEvents that have occurred. This event is raised when a bundle is successfully transmitted to a neighboring node.
- **Bundles Stored**. This is the number of bundles currently in storage. It is a count of the bundles in storage at the end of the emulation.

In addition to the bundle agent statistics, several other metrics are calculated from the bundle generation script and bundle reception log. The number of application data bundles sent from each node is counted as well as the number of data bundles delivered at the final destination. The IBR-DTN statistics do not differentiate between bundles generated by an application (actual payload data) and routing bundles. IBR-DTN routing handshakes are done using the bundle format. For this reason, the number of actual application bundles sent and delivered are counted separately. This will also help to determine the number of replications of each bundle that were generated. Essentially the number of replications per unique application bundle and the number of routing specific bundles constitutes the overhead associated with a particular routing algorithm.

- **Delivery Ratio**. The delivery ratio [84] is often seen as the most important metric in DTN routing. It is the ratio of messages successfully delivered to the number of messages created. This is calculated based on the number of application data bundles created and delivered.
- **Delivery Cost**. This is essentially a measure of the overhead associated with replication based routing. It is the total number of transmitted bundles minus the number of bundles delivered, divided by the number of bundles transmitted.
- **Bundles Replicated**. This is the count of each bundle that has been selected for forwarding to a neighboring node. At a minimum, the bundle must not be a direct delivery to the neighboring node or already exist in the neighboring node's bundle storage. For the classification and clustering based bundle filters, the bundle must also pass the criteria outlined in Section 5.3.

In addition, the delay from the time of bundle reception and bundle delivery is calculated for each successfully delivered bundle. For each emulation, the minimum, average and maximum delay are recorded. Using end-to-end delay as a metric in delay tolerant networks incorporates whatever delay is inherent in the network itself at a particular time and is out of control of the routing algorithm. However, the emulations performed used the same mobility scripts and same set of node source and destination pairs, number of files sent, and bundles were generated at the same rate. This will help to standardize the emulation scenario and eliminate the number of variables impacting the delay measurement between emulations.

7.1.2 Clustering Metrics

Two clustering metrics were used to help to determine the best number of clusters to use. The Silhouette Coefficient [85] and Calinski-Harabaz Index [86] were both used since there are no ground truth labels for the data. For this reason, the cluster results cannot be evaluated based on how well the clusters represent the actual label values. Instead, both give an indication of how dense and well separated the clusters are.

- **Mean Silhouette Coefficient**. The Silhouette Coefficient ranges between -1 and 1, with higher values indicating better defined clusters. To calculate the Silhouette coefficient, the value $a(i)$ is the mean distance between a point i and all other points in the same cluster. The values $b(i)$ is the mean distance between i and the nearest cluster that i is not a member of as shown in Eq. 7.1.

$$s(i) = \frac{b(i) - a(i)}{max(a(i), b(i))} \tag{7.1}$$

The mean of $s(i)$ for all points i is reported as the clustering metric.

- **Calinski-Harabaz Index**. The Calinski-Harabaz Index is a measure of dispersion within and between the clusters. A higher value indicates better clustering. It is defined as:

$$s(k) = \frac{SS_B}{SS_W} \times \frac{N-k}{k-1} \tag{7.2}$$

where k is the number of clusters, SS_B is the overall between-cluster variance, SS_W is the overall within-cluster variance, and N is the number of data points [86]. The between-cluster variance is defined as :

$$SS_B = \sum_{i=1}^{k} n_i \, \|m_i - m\|^2 \tag{7.3}$$

In Eq. (7.3), n_i is the number of points in cluster i, m_i is the center of cluster i, m is the mean of all of the points and $\|m_i - m\|^2$ is the Euclidean distance between m and m_i The within-cluster variance SS_w is calculated as:

$$SS_W = \sum_{i=1}^{k} \sum_{x \in c_i} \|x - m_i\|^2 \tag{7.4}$$

where c_i is each of the k clusters, x is a data point and m_i is the center of cluster c_i.

7.1.3 Classification Metrics

There are several basic classification metrics that are used to analyze the performance of a model's predictions.

- **Accuracy**. The accuracy is the count of correctly predicted labels, meaning the prediction is equal to the corresponding entry in the test set.
- **Confusion Matrix**. The confusion matrix is a more detailed break down of correct and incorrect classifications [87]. For binary classifications, the count of true positives t_p, false positives f_p, true negatives t_n and false negatives f_n will be tallied as compared to the ground truth known values of the test set. An intuitive example of these metrics can be seen in a medical test for a disease. The test will correctly predict some sample of patients as healthy that do not have the disease (t_n) and also correctly predict some patients as ill that have the disease (t_p). There is also a percentage of patients that are incorrectly identified as having the disease that actually do not (f_p) and patients that have been incorrectly identified as healthy that actually do have the disease (f_n).

	Predicted Positive	**Predicted Negative**
Labeled Negative	False Positive	True Negative
Labeled Positive	True Positive	False Negative

Table 7.1. Confusion Matrix

- **Recall**. From the values calculated in the confusion matrix, the recall of the classifier can be calculated. Recall is calculated as shown in Eq. (7.5), where t_p and f_n are the number of true positives and false negatives, respectively. Recall is an indication of how well a classifier identifies points of interest (actual value of positive) .

$$t_p/(t_p + f_n) \tag{7.5}$$

- **Precision**. Similarly, the precision is calculated from the true positives t_p and false positives f_p in Eq. (7.6). Precision is an indication of how many points identified as relevant actually were relevant.

$$t_p/(t_p + f_p) \tag{7.6}$$

  The relationship between recall and precision is essentially a tradeoff between being aggressive in identifying points of interest when there is little consequence for over identification versus more precisely identifying points of interest with a chance of missing some points.

7.1.4 Multi-Label Classification Metrics

To validate the multi-label classification performance, there are four well known multi-label prediction metrics used. Two related metrics for multi-label classification are Hamming loss and zero-one loss [88]. Hamming loss calculates the fraction of labels that are incorrectly classified. That is:

$$L_H(\mathbf{y}, \mathbf{h}(\mathbf{x}))) = \frac{1}{m}\sum_{i=1}^{m} [\![y_i \neq h_i(\mathbf{x})]\!]. \tag{7.7}$$

In Eq. (7.7), $L_H(\mathbf{y}, \mathbf{h}(\mathbf{x})$ is the Hamming loss function, where $\mathbf{y}$ is the set of m observed labels for a given instance and $\mathbf{h}(\mathbf{x})$ is the output of the classifier (the m predicted labels). The expression $[\![X]\!]$ evaluates to 1 if X is true and 0 otherwise.

The Hamming loss is in contrast to zero-one loss which considers the entire prediction incorrect if any label in the prediction is incorrect, as shown in Eq. 7.8:

$$L_s(\mathbf{y}, \mathbf{h}(\mathbf{x}))) = [\![\mathbf{y} \neq (\mathbf{h}(\mathbf{x}))]\!]. \tag{7.8}$$

Hamming loss is a more lenient metric which scores based on individual labels. In both Hamming loss and zero-one loss, values tending toward zero indicate good performance whereas values tending toward one indicate a higher percentage of misclassification.

The F1 score is as a weighted average of the precision and recall. Precision and recall are calculated by counting the total true positives t_p, true negatives t_n, false negatives f_n and false positives f_p for examples classified as label *l*. Micro-average precision and recall are defined in Eqs. 7.9 and 7.10, respectively [89]:

$$P_{micro-avg} = \frac{\sum_{i=1}^{m} t_{pi}}{\sum_{i=1}^{m} (t_{pi} + f_{pi})} \tag{7.9}$$

$$R_{micro-avg} = \frac{\sum_{i=1}^{m} t_{pi}}{\sum_{i=1}^{m} (t_{pi} + f_{ni})}. \tag{7.10}$$

In a similar manner to Equations (7.5) and (7.6), the difference between the micro-averaged precision and recall is the consideration of false positives f_{pi} and false negatives f_{ni}. Micro-averaged F1 score is given by Eq. 7.11:

$$F1_{micro-avg} = \frac{2 \times P_{micro-avg} \times R_{micro-avg}}{P_{micro-avg} + R_{micro-avg}} \tag{7.11}$$

The Jaccard similarity score, or multi-label accuracy [90] is the size of the intersection of two label sets (the predictions and true labels) divided by the size of the union of the two label sets. In both Jaccard similarity score and F1 scores, 1 is the best score and 0 the worst. Several metrics are selected as it is well known in machine learning problems that it is often the case that there is a trade-off between metrics, with classifiers performing well in some metrics, performing poorly by other standards. Therefore, to get a complete picture of the performance it is necessary to consider several metrics.

7.2 Survey of DTN Routing Performance

This section discusses initial testing using DTN2, the DTN reference implementation. The NASA DTNBone test bed was used to evaluate the several current state-of-the-art routing algorithms for delay tolerant networks. The NASA DTNBone [91] consists of thirteen virtual machines running Ubuntu 14.04.5 LTS. Each virtual machine runs the current version of DTN2 (version 2.9.0). The nodes are networked together in a mesh topology and link delays and disruptions are simulated using channel-emulating software. Link disruptions are simulated hourly, with each link following its own schedule. The configuration of the network for initial testing is shown in Fig. 7.1. Table 7.2 shows the data rates and one-way delays associated with each link. TCP was used as the convergence layer for this initial testing. It should be stated for clarity that the development of the DTNBone testbed was not part of the work of this dissertation and rather was used since it was an existing system already developed for DTN testing.

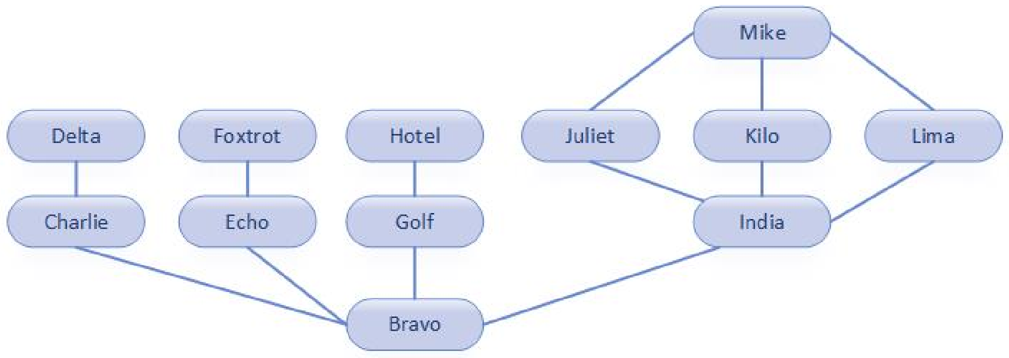


Figure 7.1. Network Topology of the NASA DTNBone

Link	**Delay**	**Rate Limit**	**Availability**
bravo-charlie	None	None	Toggles every 30 min.
bravo-echo	None	None	Toggles every 3 min.
bravo-golf	5 s	128 Kb/s	Always available
bravo-india	None	None	Always available
charlie-delta	None	None	Toggles every 3 min.
echo-foxtrot	None	None	Toggles every 30 min.
golf-hotel	200 ms	256 Kb/s	Always available
india-juliet	1250 ms	512 Kb/s	Up for 20 min. at the beginning of the hour
india-kilo	1250 ms	512 Kb/s	Up for 20 min. at 20 min. past the hour
india-lima	1250 ms	512 Kb/s	Up for 20 min. at 40 min. past the hour
juliet-mike	200 ms	1544 Kb/s	Toggles every 6 min.
kilo-mike	200 ms	1544 Kb/s	Toggles every 5 min.
lima-mike	200 ms	1544 Kb/s	Toggles every 2 min.

Table 7.2. DTNBone Availability Schedule

DTN2 provides a framework for DTN software research and development. It includes a bundle protocol implementation as well as DTLSR, flooding, and PRoPHET routing implementations. In addition, it provides an interface for external routers to communicate with and control the DTN2 daemon, allowing developers to easily integrate custom software with the existing bundle protocol implementation. This is accomplished by sending XML message to a port used by the DTN2 daemon. The RAPID protocol was implemented using this approach and as such can be used as an example for further software development.

For purposes of exercising each algorithm, the dtnperf tool included in DTN2 was used to send a series of bundles to a specified node in the network. The dtnperf tool allows the user to configure the bundle size, number of bundles and a destination server node in the network to send the bundles to. It generates a time stamped log of bundle forwarding and delivery status to allow the user to analyze network performance. Most testing was done sending bundles from node Bravo to node Mike since this is the most complex path for the algorithm to navigate as it has the most hops, possible paths and intermittent disruptions. The algorithms selected were the DTLSR, PRoPHET and flooding implementations provided by DTN2, as well as the RAPID implementation developed by University of Massachusetts Amherst as an external router to DTN2. Of the three routing protocols internal to DTN2 that were tested, DTLSR performed the best, followed by flooding. The results from initial testing a summarized in Table 7.3.

Algorithm	**Average Delay (s)**	**Average # Replications**
DTLSR	78.15	2.24
Flooding	99.15	5.22
RAPID	511.69	5.38

Table 7.3. Results for 50 1 KB Bundles

To study the effects of bundle size on the routing algorithms, a simpler destination to reach in the network was chosen. The path from node Bravo to node Hotel consists of only two hops and has links which are always available. DTLSR continued to perform better than RAPID as shown in Fig. 7.2. In addition to having a lower average delay, it also did not replicate unnecessary packets. In the case of RAPID, there were still typically an average of 4 bundles replicated per delivery, even though there was a direct path to the destination. The PRoPHET

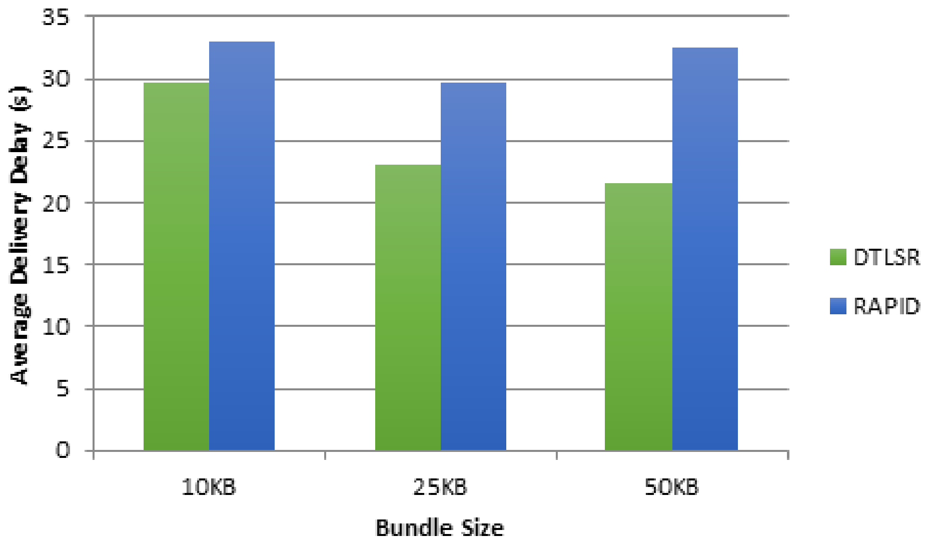


Figure 7.2. Average Delay for 2 Hop Path

routing algorithm is not included in the preliminary results as its performance was quite unstable and it was difficult to send any number of bundles to even directly connected nodes. Forwarding bundles to a destination requiring multiple hops was even less successful. The initial parameters used were configured as recommended in the PRoPHET Internet-Draft and also several attempts at adjusting them to improve performance were made. This is not to say that further testing could not be done to determine the cause of the poor performance, whether it be due to configuration parameters needing to be tuned for each node or some other factor in the PRoPHET implementation or DTN2.

There are a number of studies of characterizing DTN routing algorithm performance that have shown PRoPHET's performance to be inferior to MaxProp, Spray and Wait, Epidemic, and RAPID [28], [92], [27] in some cases, particularly depending on the mobility scenario used, as well as the amount of time given to allow the algorithm to converge. For the case of this initial testing, two problems were noted that impacted performance. The first was that links with a delay associated with them (bravo-golf: 5 seconds, india-juliet, india-kilo, india-lima: 1250ms) seemed completely missed in the exchange of PRoPHET routing bundles containing the delivery predictability information. This is no doubt due to the expiration of some timeout value, however increasing the value of the hello_interval (20 s) and hello_dead value (allow up to 10 hello_intervals before a node is considered unreachable) did not help to solve the problem.

The PRoPHET algorithm, or specifically the DTN2 PRoPHET implementation, seemed to have a difficult time reacting to availability changes in links that toggle frequently (bravo-echo, toggle link availability every 3 minutes). When the link was available, the delivery predictability would approach 1, however the link would become unavailable and this would not be reflected in the delivery predictability, causing the algorithm to continue to repeatedly attempt to contact the unavailable node. These types of problems are noted in [27], where the authors discuss improvements for a second revision of the PRoPHET protocol. They note that when the frequency of encounters is disproportionate throughout the network and encounters occur frequently enough that the delivery predictability is not reduced quickly enough by the aging procedure, the algorithm can fail to produce an accurate representation of the current network state. It is possible that performance could be improved for this test case by further investigating the recommendations of PRoPHETv2, as well as further refining the PRoPHET configuration parameters for each node. In the initial test case, all nodes were configured with the same parameters, but it would likely be beneficial to customize the parameters based on the link characteristics of each node, essentially to make nodes with links that change frequently adjusts their delivery predictabilities more aggressively.

It was decided after this experimentation that DTN2 was not a good candidate for further development. This is largely due to the fact that it requires multiple dependencies that are no longer being updated and becoming increasingly difficult to install on newer operating systems. In addition, the resource usage is considerably larger than IBR-DTN as discussed in [49]. However, this work with DTN2 was helpful in becoming familiar with the concepts of DTN, as well as working with some alternative routing methods.

7.3 Q-Routing Simulation Results

This section discusses the simulation of Q-Routing using OMNeT++. The details of the Q-Routing algorithm are discussed in Chapter 4.2. The first set of simulations (1-4) were performed with a short delay between nodes for initial testing. Several nodes are selected to reject all packets to simulate failure or unavailability of a node. This will test the ability of Q-Routing to learn which paths lead to the shortest delay, yielding a greater reward value in the Q-Table. Later simulations (5) were performed with a slightly longer propagation delay between nodes to create an environment closer to an actual DTN.

7.3.1 Simulation 1 – 10 Node Mesh

The purpose of this experiment was to compare the shortest path algorithm (Dijkstra) to that of Q-routing. A random mesh of 10 nodes was generated with connections between nodes as shown in Table 7.4. The links between nodes are bidirectional.

Source Node	**Destination Node**	**Delay (s)**	**Date Rate (bps)**
1	2	5	1 kbps
2	3	0.04	1 Mbps
1	4	1.5	1 kbps
4	5	0.01	1 Mbps
4	7	0.01	1 Mbps
7	8	10.5	1 kbps
8	5	2.01	1 Mbps
8	9	0.01	1 Mbps
9	6	8.01	1 kbps
6	3	2.7	1 Mbps
6	10	0.01	1 kbps
10	2	6.1	1 Mbps

Table 7.4. Network Configuration for Simulation 1

In this simulation, packets were first routed using the shortest path algorithm. Each node application generates 50 kB packets to transmit to a randomly selected node. Packets were generated at intervals of 10 milliseconds for 1 min, 5 milliseconds for 1 minute, and then 2 milliseconds for 1 minute. It should be noted that these times pertain to time in the simulation model, not actual clock time. In addition to increasing network traffic, a failure in node 3 occurs at random intervals. The failing node will periodically drop all packets, and then it will recover and perform normally until the next failure is scheduled to occur. The same test was then conducted using the Q-routing algorithm to compare its performance to the shortest path. This is to test the ability of Q-Routing to learn the pattern of failures based on feedback from successful and unsuccessful packet deliveries. The data rate is ramped from 10 millisecond intervals to 2 millisecond intervals over 3 minutes. The performance of the two algorithms are shown in Figure 7.3.

Data rates as shown above to attempt to saturate the network as much as possible. With network links set to lower data rates, nodes are able to keep up with

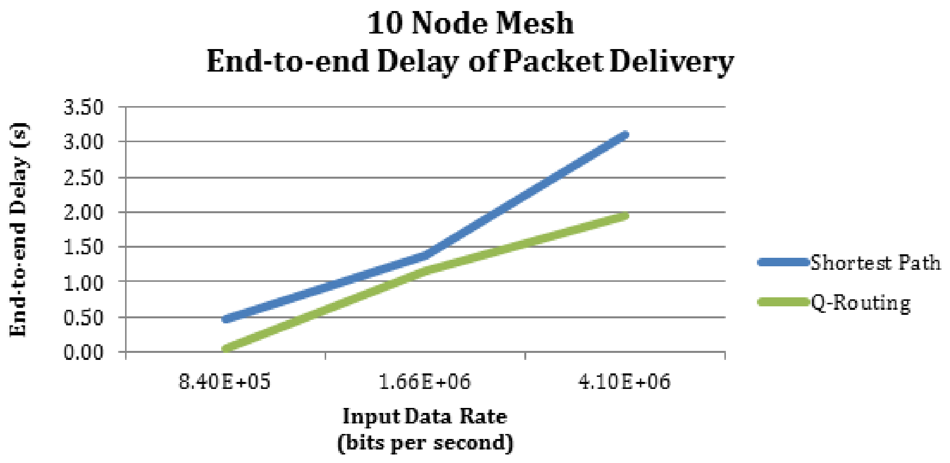


Figure 7.3. Average Delay 10 Node Mesh

incoming packets and the delay in delivery to the destination node is less than one second for both the shortest path algorithm and Q-routing. Due to resource constraints, it was not possible generate packets any faster than 4 Mbps for each node. The simulation times increase dramatically as the data rate increases or as more nodes are added in the network. It can be seen that as the network load increases, the shortest path algorithm starts to perform poorly compared to Q-routing. This intuitively makes sense due to the fact the Q-Routing uses a delay estimate to determine the best path versus the distance between nodes or number of hops.

7.3.2 Simulation 2 – 20 Node Mesh

This experiment was similar to simulation 1. The shortest path algorithm is compared to the Q-routing algorithm. The network in this case consists of 20 nodes in order to test the scalability of Q-Routing. The network configuration is shown in Table 7.5.

Source Node	**Destination Node**	**Delay (s)**	**Date Rate (bps)**
1	2	0.01	1 kbps
1	7	0.01	1 Mbps
2	3	2.01	1 Mbps
3	4	0.01	1 kbps
4	5	0.01	1 Mbps
5	6	0.01	1 Mbps
6	12	0.01	1 Mbps
12	18	5.01	1 kbps
18	17	0.01	1 Mbps
17	11	1.01	1 Mbps
11	10	0.01	1 Mbps
10	16	1.01	1 Mbps
16	17	0.01	1 kbps
16	15	0.01	1 Mbps
15	9	0.01	1 Mbps
9	8	0.01	1 kbps
8	14	0.01	1 Mbps
14	15	10.01	1 kbps
14	13	0.01	1 Mbps
13	7	0.01	1 Mbps
13	19	2.01	1 Mbps
19	20	0.01	1 Mbps
1	7	4.01	1 Mbps
20	14	0.01	1 Mbps
20	10	6.01	1 Mbps
18	2	2.01	1 kbps
7	3	1.01	1 kbps

Table 7.5. Network Configuration for Simulation 2

Figure 7.4 shows the comparison in end-to-end delay for Q-routing versus the shortest path for the 20 node mesh. Again, Q-routing performed better than the shortest path algorithm, especially as the network load increased. It is also interesting to note that the trend line for the shortest path increases much more steeply than Q-routing. This indicates that performance would likely continue to suffer, whereas with Q-routing the delay increase begins to level off.

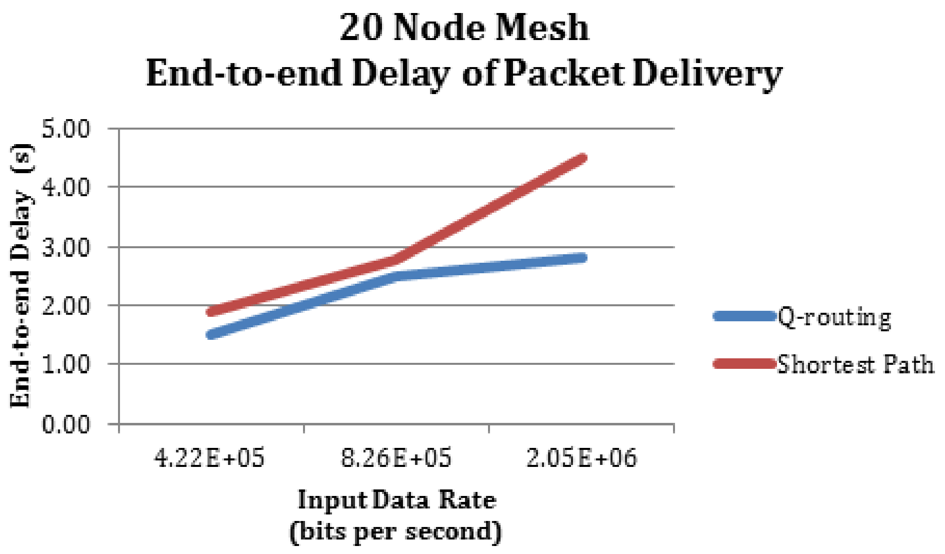


Figure 7.4. Average Delay for 20 Node Mesh

7.3.3 Simulation 3 - Multiple Node Failures

For this simulation, the number of nodes forced to failure are incrementally increased in the 20 node mesh. The nodes selected to fail are shown in the Table 7.6.

Simulation Number	**Node Address**
1	3
2	3,6
3	3,6,15
4	3,6,15,13

Table 7.6. Nodes Selected for Failure

Figure 7.5 shows the average delay increase as the number of nodes forced to failure increases. While the performance does start to decline as more nodes fail, it doesn't decline dramatically. This shows that Q-routing is a good candidate for a routing policy in an unpredictable network [6].

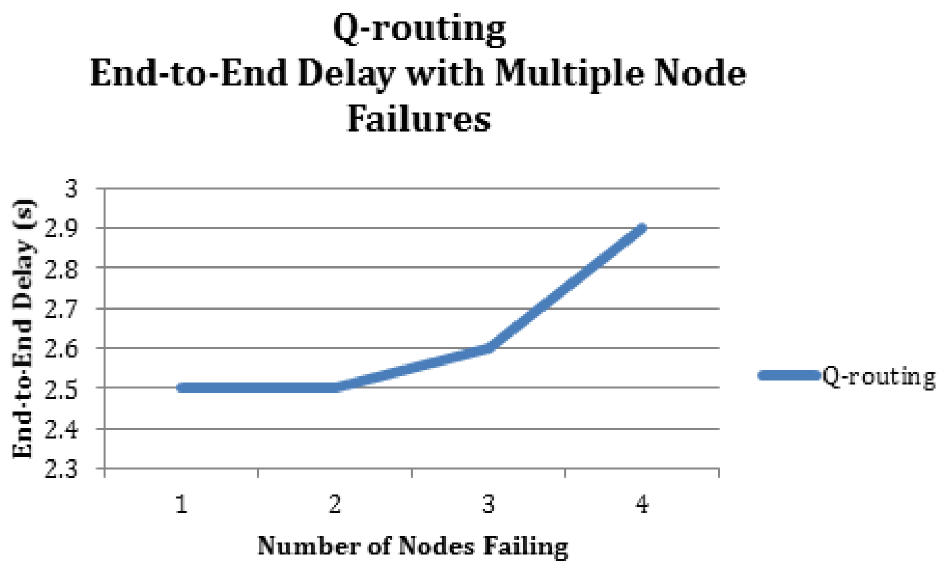


Figure 7.5. Average Delay Versus Number of Nodes Failing

7.3.4 Simulation 4 – Forgetting Policy

Simulation 4 involved implementing a forgetting policy to improve performance when a node failure occurs as discussed in [6]. The forgetting policy attempts improve the Q-table once the algorithm has converged by rediscovering other paths that may be more efficient than the current policy. It was not found that there was any performance advantage to this approach. Instead, a better approach was to occasionally, randomly choose a different node than what is suggested by the Q-table. This method allows other paths to be explored, rather than constantly tending to the best paths that the learner has converged to. Figure 7.6 shows the average delay using the forgetting policy.

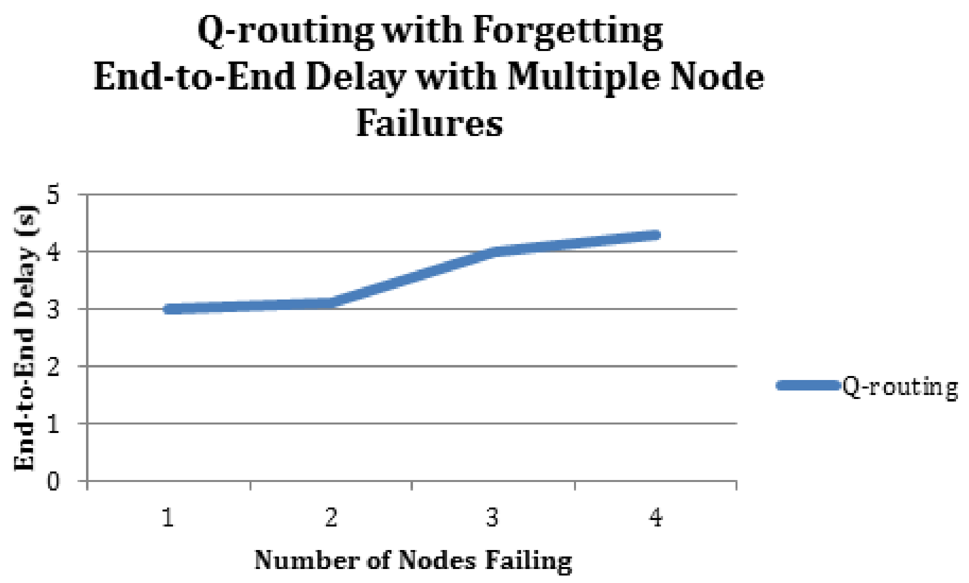


Figure 7.6. Average Delay with Forgetting Policy

The same nodes were selected to fail as in Simulation 3 and kept all other parameters the same as well. Overall, the delay was larger for all cases.

7.3.5 Simulation 5 – Longer Delay

Simulations with a longer propagation delay between nodes were conducted using a 9 node mesh. Each node application generates 50 kB packets to transmit to a randomly selected node following an exponential distribution with a mean which was varied from 1 to 0.1 seconds. Each simulation executed for 2.7 hours in simulation time (10000 simulation seconds).

Table 7.7 shows the link characteristics used. The link characteristics have been selected such that multiple possible paths to a destination will have the same number of hops but a longer propagation delay and/or slower data rate. This was done as a test to determine that the Q-routing algorithm can successfully choose the quicker path based the average end-to-end delays of packets

Source Node	**Destination Node**	**Delay (s)**	**Date Rate (bps)**
0	3	1 to 30	200 kbps
0	4	1	2 Mbps
0	5	1	200 kbps
1	3	1	2 Mbps
1	4	1 to 30	200 kbps
1	5	1 to 30	2 Mbps
2	3	1	200 kbps
2	4	1	200 kbps
2	5	1 to 30	200 kbps
3	6	1	200 kbps
3	7	1	2 Mbps
3	8	1 to 30	2 Mbps
4	6	1	200 kbps
4	7	1 to 30	2 Mbps
4	8	1	200 kbps
5	6	1	2 Mbps
5	7	1	200 kbps
5	8	1 to 30	2 Mbps

Table 7.7. Network Configuration for Simulation 5

sent on a particular route. Q-routing was found to react to the network load quite well and as expected, performed similarly to the shortest path algorithm under low network load but out-performed the shortest path algorithm as the network load increased. This shows that previous end-to-end delays may be a good indication of congestion or link unreliability along a particular path.

Figure 7.7 shows the average end-to-end delay versus the maximum link propagation delays that were incrementally increased over 4 simulations. It can be seen that the shortest path cannot account for the link delays and that Q-Routing is able to determine better paths based on an overall time estimate rather than the number of hops. Figure 7.8 shows the average end-to-end delay as the input data rate is increased (more traffic is generated by the nodes). As the network is more congested, the delay increases and Q-Routing is better able to determine routes which will lead to a shorter delay.

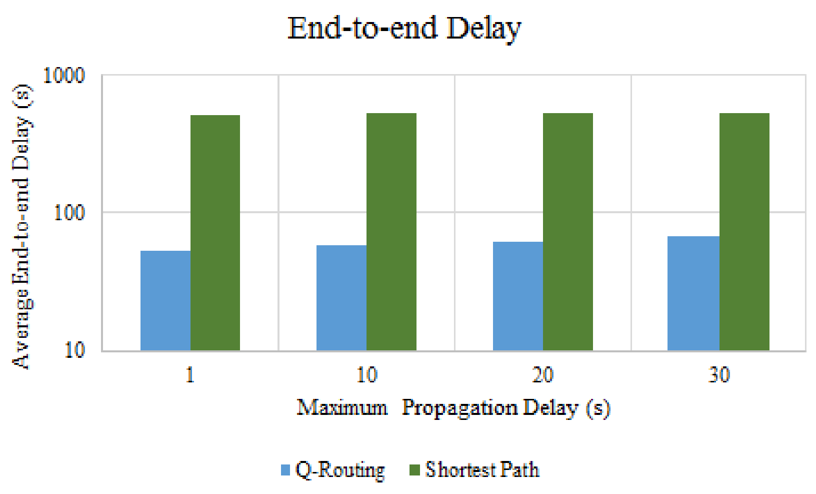


Figure 7.7. Average End-to-End Delay Based on Propagation Delay

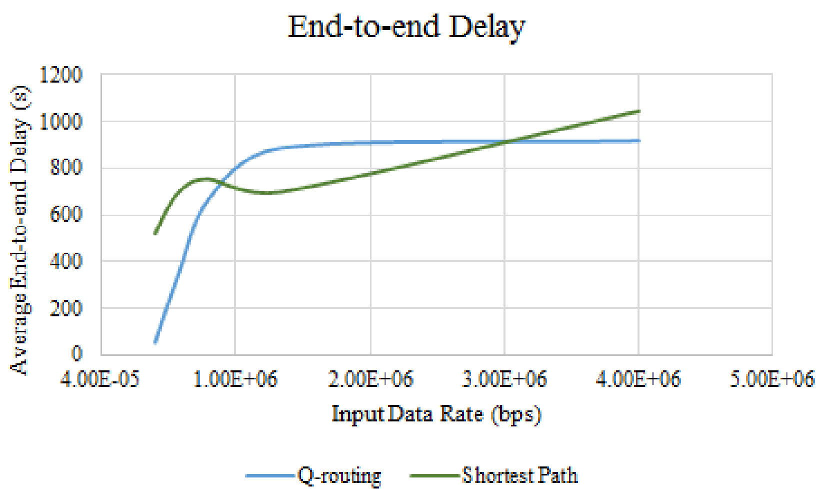


Figure 7.8. Average End-to-End Delay Versus Data Rate

This series of simulations studied Q-Routing with networks with brief link delays (on the order of seconds), slightly longer delays (10s of seconds), multiple nodes becoming unavailable and network congestion. From the results, it can be seen that Q-Routing does have some advantages for use in unpredictable network environments. However, the need for Q-Routing to propagate delay estimates between nodes could become problematic as delays and periods of unavailability become longer.

After this series of simulations were conducted, it was determined that further testing should be done in a more realistic DTN environment. The next section cover an emulations that were done using actual bundle protocol software and a more sophisticated test environment. The draw backs of OMNeT++ were that while it is very useful as a multi-purpose network simulator, there are several tools available such as CORE and the ONE that are more suited to DTNs.

7.4 Multi-label Classification Approach

The multi-label classification approach to bundle selection for replication routing was implemented using CORE as an emulation environment and a modified version of IBR-DTN as discussed in Chapters 5 and 6.

7.4.1 Classification Results

This section discusses the results of the multi-label classification independent of the routing results. This is the fourth step in the process of creating the machine learning model, following storing network statistics and training the classifier as discussed in section 6.3. During initial testing, several data set were used to determine the classifier performance with different mobility models. They are a mathematically generated RandomWalk mobility model from BonnMotion [81], and two datasets from the ZebraNet [80] archive from the CRAWDAD repository at Dartmouth. The ZebraNet dataset consists of the GPS coordinates from DTN experiment that traced the location of multiple zebras in the wild. The use of real world traces can help to provide a more realistic pattern of motion that what can be obtained from synthetic mobility models.

Figures 7.9 to 7.16 show the classification performance of several different base classifiers and multi-label approaches. Bundles are traced from the source node through each forwarding node to the destination and labeled as successfully delivered or not. The classifiers are trained to determine what nodes will results in successful delivery. Base classifiers selected were Decision Tree, Gaussian Naive Bayes and K-Nearest Neighbors. In addition, two different routing methods were used to obtain the initial bundle traces, Epidemic and PRoPHET. This was done to see if there was any impact on classification performance since PRoPHET uses a probabilistic method, whereas Epidemic simply uses a replication approach to all nodes as discussed in Chapter 3. Each color on the graphs in Figure 7.9 to 7.16 indicates a different multi-label approach to classify the set of nodes in the network as reliable or unreliable.

It can be seen that the classification method performs quite well for all combinations of initial routing , base classifier and multi-label method. Micro-Averaged F1 score, as defined in Eq. (7.11), ranges from 0.1 to 0.8 for the dataset, where a value of 1 indicates good performance. The lowest ranking classifier was the KNN base classifier and best performing classifier was Decision Tree.

The Jaccard Similarity Score is shown in Figures 7.11 and 7.12. Overall, all classifiers performed well by this metric, with again Decision Tree performing the best and KNN performing the worst, however with a much smaller differentiation in values. Again results based on the Hamming Loss were promising. In this case Naive Bayes performed the worst and Decision Tree performed the best. In the case of Hamming Loss, good performance is indicated by a value of zero and poor performance is indicated by 1. The values for the classifiers ranged from less than 0.05 for Decision Tree and 0.2 for Naive Bayes.

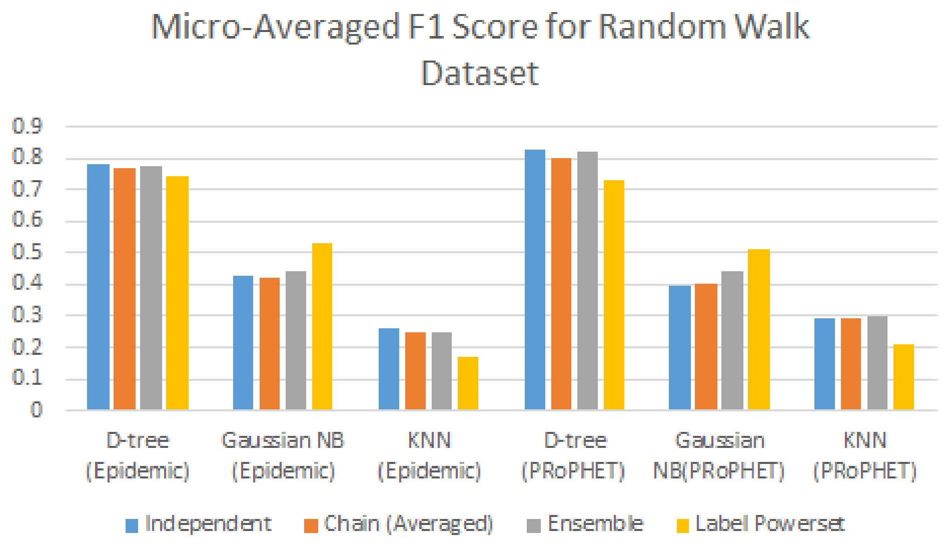


Figure 7.9. Micro-Averaged F1 Score (Random Walk)

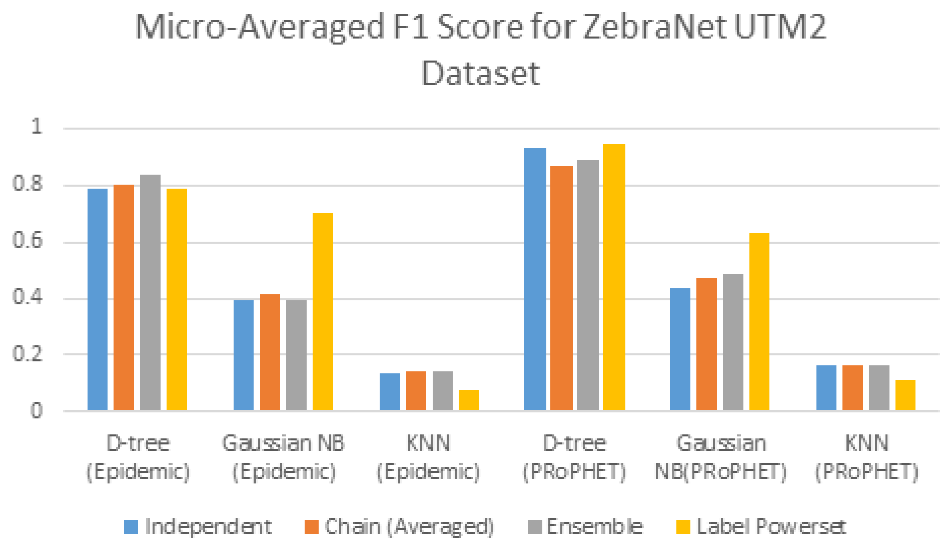


Figure 7.10. Micro-Averaged F1 Score (ZebraNet)

Finally, the Zero-One Loss is shown in Figures 7.15 and 7.16. The vertical axis represents loss, with values closer to zero indicating a higher percentage of correct predictions. The results are similar to the performance of other metrics in which Decision Tree performed the best and Naive Bayes and KNN performed the worst. Values range from 0.05 to 0.25.

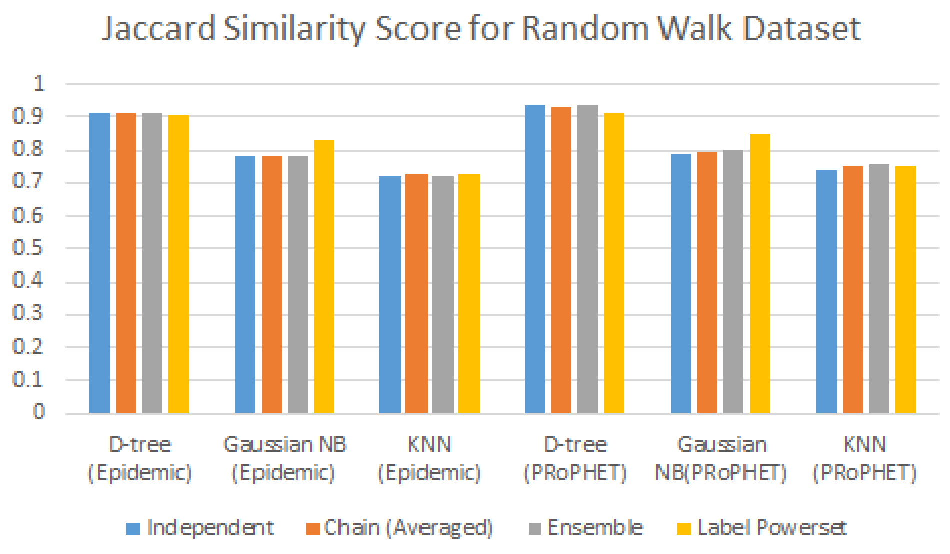


Figure 7.11. Jaccard Similarity Score (Random Walk)

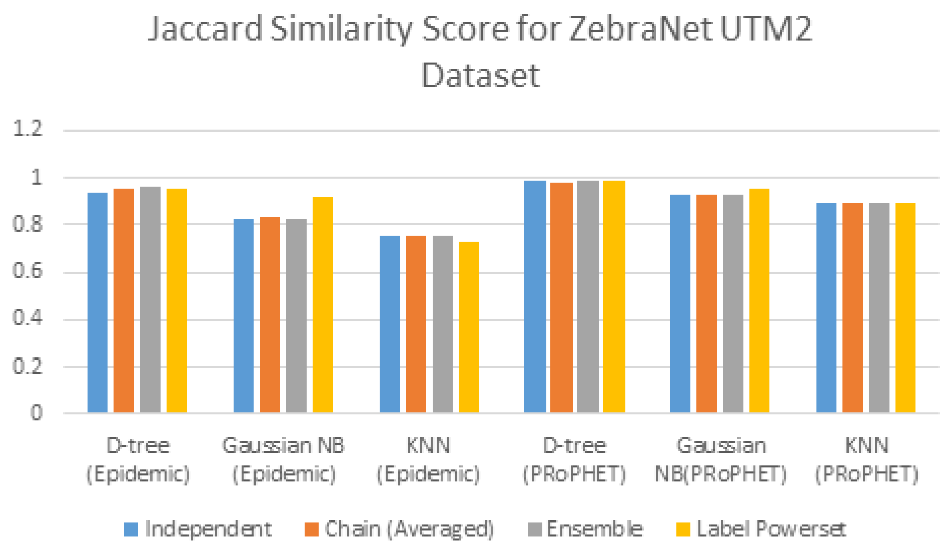


Figure 7.12. Jaccard Similarity Score (ZebraNet)

7.4.2 Routing Results

This section discusses the routing results obtained after the classification model has been trained and is now used to filter bundles to the most reliable nodes. The results of the classification approach are compared to Epidemic routing, where

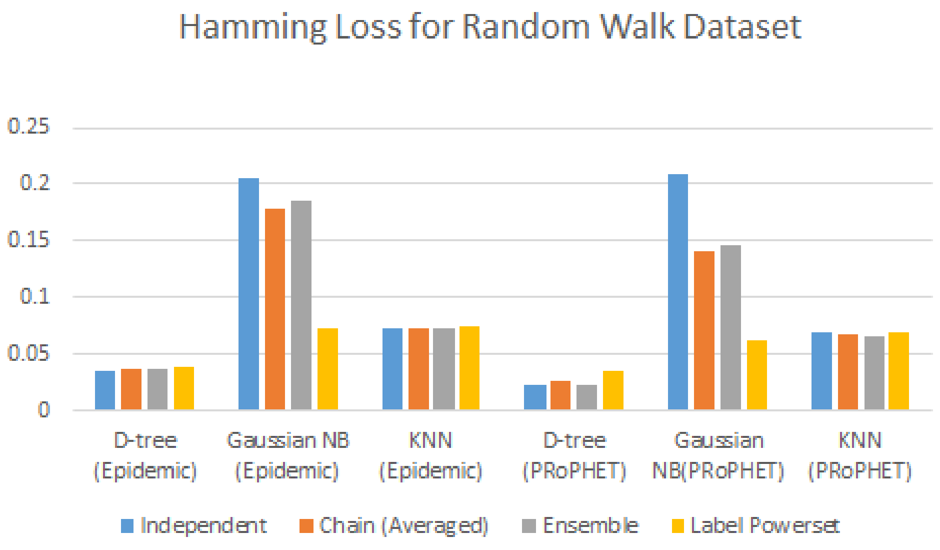


Figure 7.13. Hamming Loss (Random Walk)

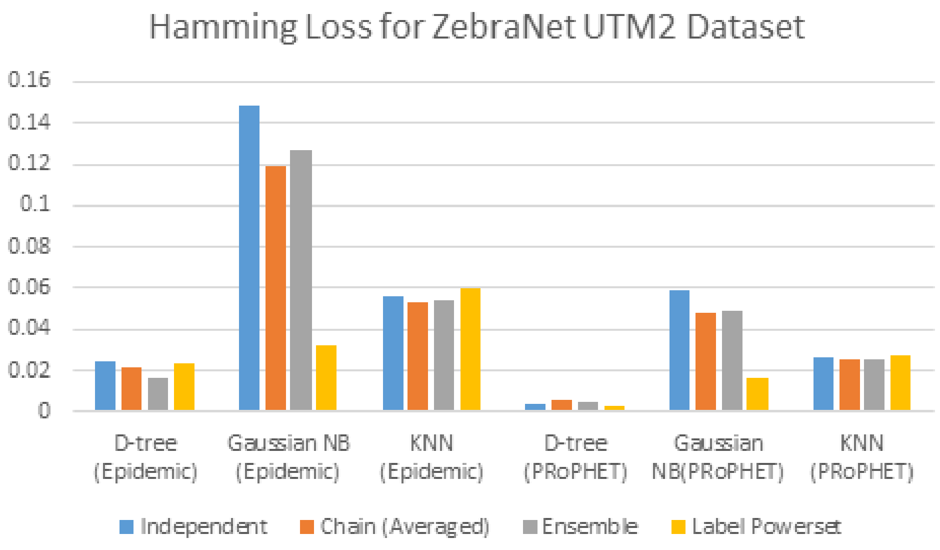


Figure 7.14. Hamming Loss (ZebraNet)

bundles are sent to all nodes that do not already have a copy of a given bundle. Several parameters are adjusted in addition to the routing method to see the results of additional traffic and queue lengths on the routing performance. To accomplish this, in each emulation scenario each of the 10 nodes will send 180 10 kB bundles. The bandwidth is changed from 10 kbs, to 100kbps, to 1 Mbps in

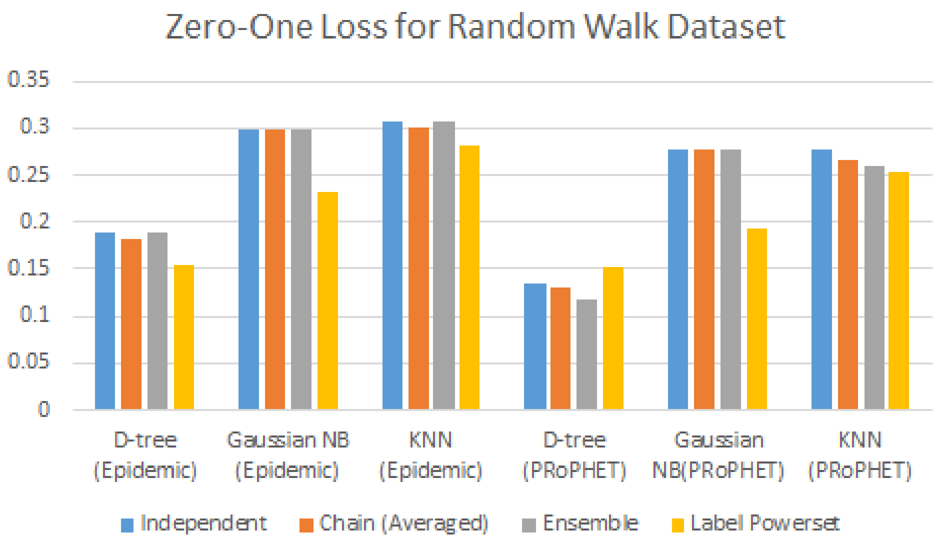


Figure 7.15. Zero-One Loss (Random Walk)

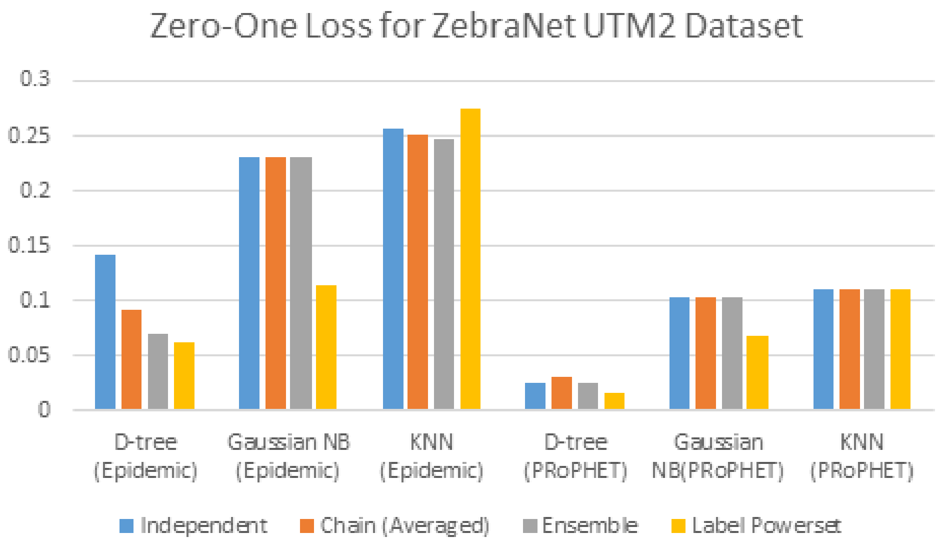


Figure 7.16. Zero-One Loss (ZebraNet)

3 scenarios, each using the same mobility script for node locations. In addition, this series is emulated 2 separate times, once with a bundle time-to-live of 90 seconds and once with 60 seconds. Figures 7.17 and 7.18 show the total number of bytes transmitted and received from the TCP convergence layer (CL). This includes all original bundles sent from a source node, replicated bundles that have

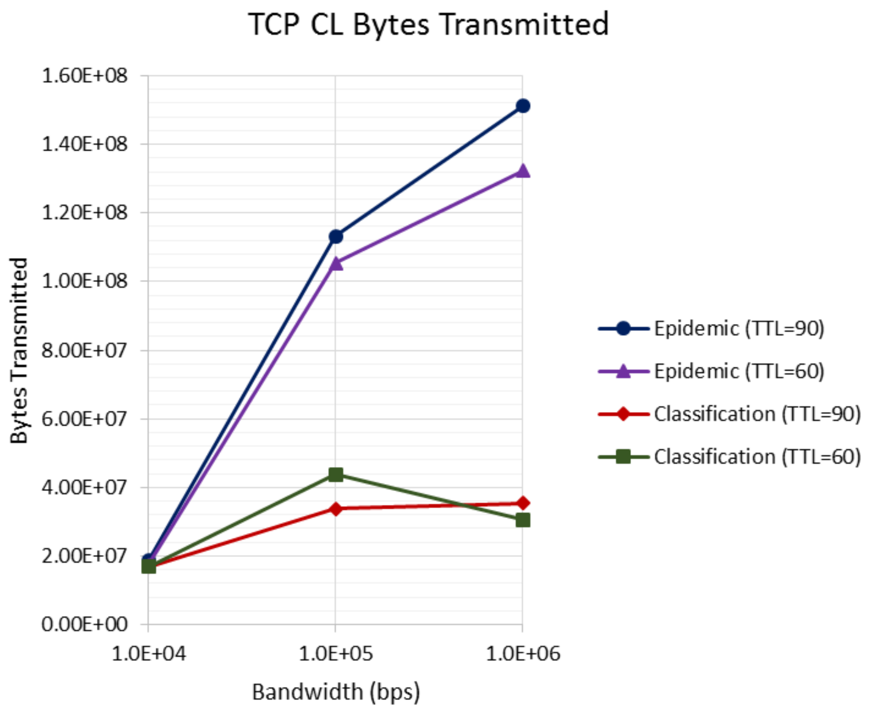


Figure 7.17. Epidemic versus Classification Filter TCP CL Bytes Transmitted

been forwarded by the routing module and bundles that are specific to the routing modules that contain handshake information and a list of the bundles that are already known to the neighboring node. The classification approach reduces the total number of bytes transmitted and received by replicating fewer bundles, since only neighbors which are deemed good candidates for bundle delivery are selected. This reduces wasted contact time, bundle processing, and the number of bundles in storage on each node. All these are resources which are desirable to conserve in resource constrained systems.

Figure 7.19 shows the percentage of bundles delivered versus the number of bundles sent for Epidemic and Classification based routing. The emulations were conducted with three different node bandwidth settings to determine the

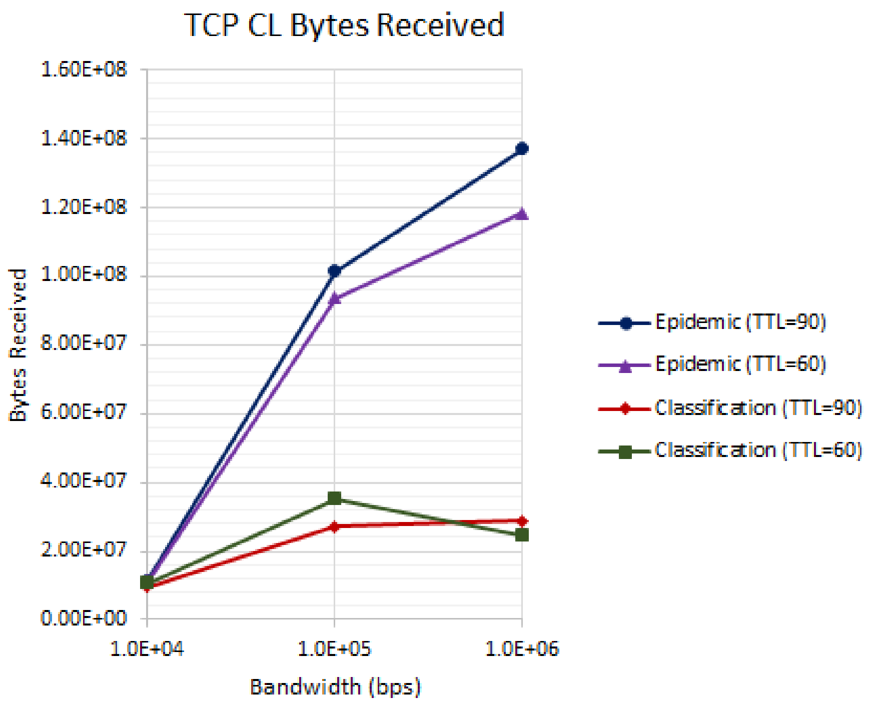


Figure 7.18. Epidemic versus Classification Filter TCP CL Bytes Received

effect of bundle expiration and queuing on each algorithm. In addition, the bundle expiration time is varied from 60 to 90 seconds for this same reason.

Figure 7.20 shows the bundle delivery cost for Epidemic versus Classification based routing. A value of zero would indicate that all bundles which were transmitted where delivered and values closer to zero indicate better performance. This metric in particular shows an advantage of classification based routing in that fewer unnecessary bundles (bundles that never get delivered to their destination) are transmitted. This is because fewer bundles are waiting in queues to be forwarded or expire before being delivered. Figure 7.21 shows that fewer bundles are replicated by the classification based routing and Figure 7.22 shows that fewer bundles expire before being delivered. Figure 7.23 shows fewer bundles are queued by the classification based routing as well.

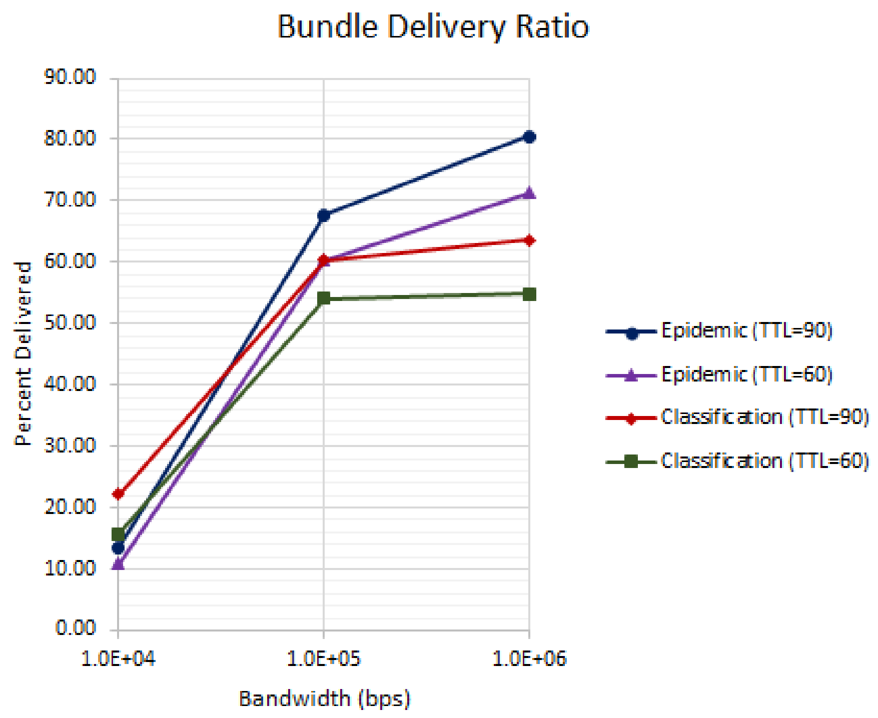


Figure 7.19. Epidemic versus Classification Filter Bundle Delivery Ratio

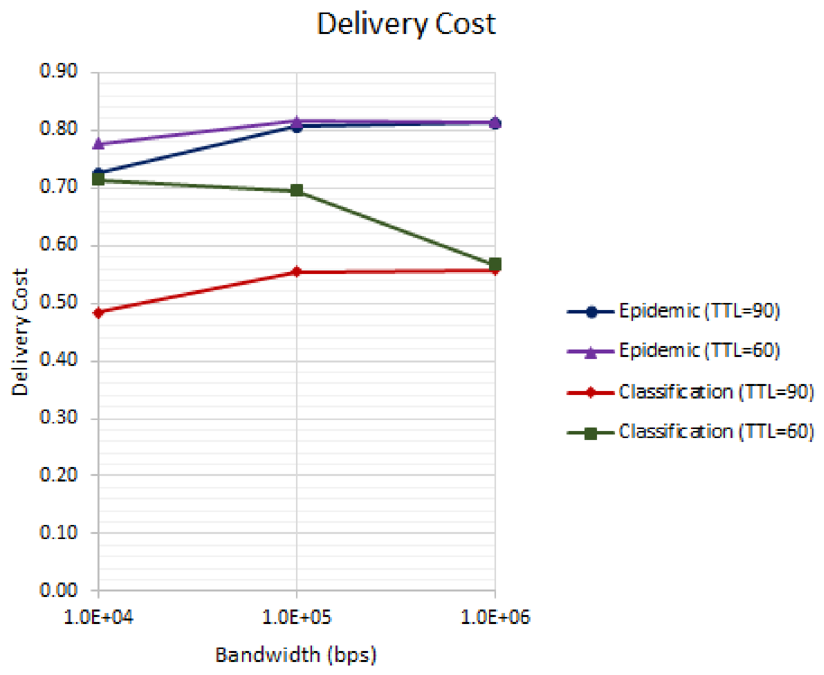


Figure 7.20. Epidemic versus Classification Filter Bundle Delivery Cost

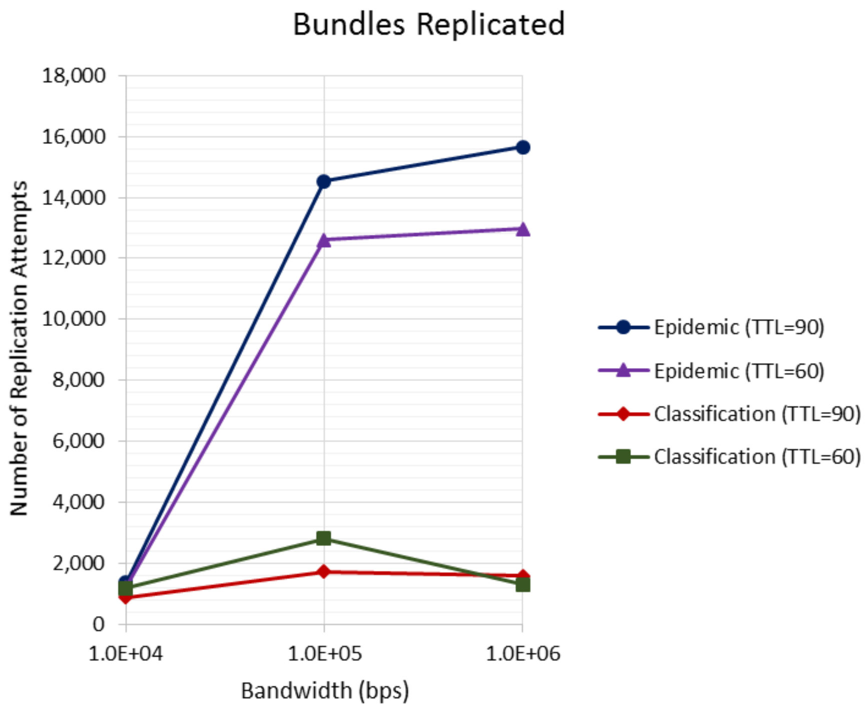


Figure 7.21. Epidemic versus Classification Bundles Replicated

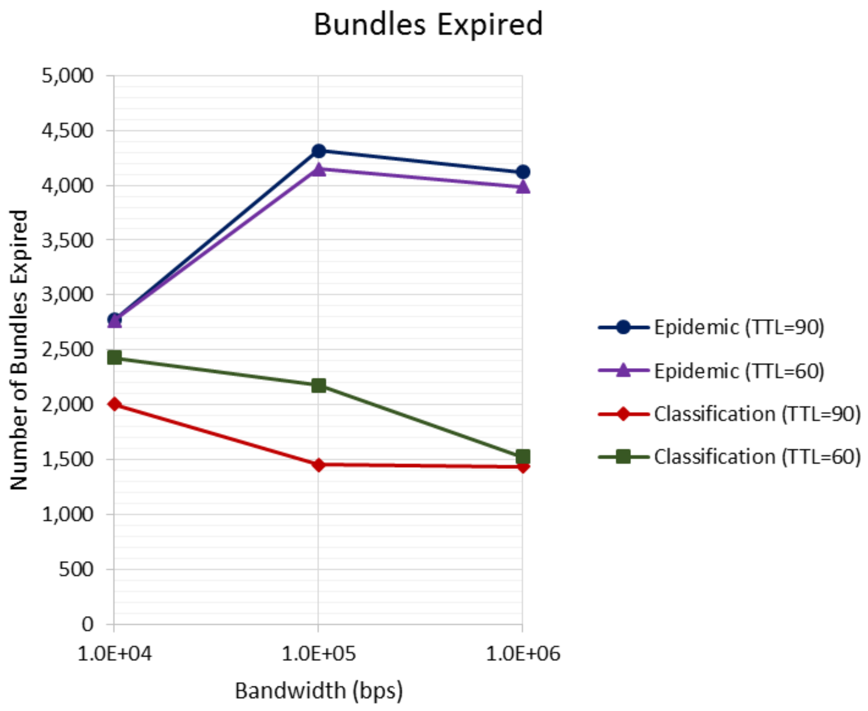


Figure 7.22. Epidemic versus Classification Bundles Expired

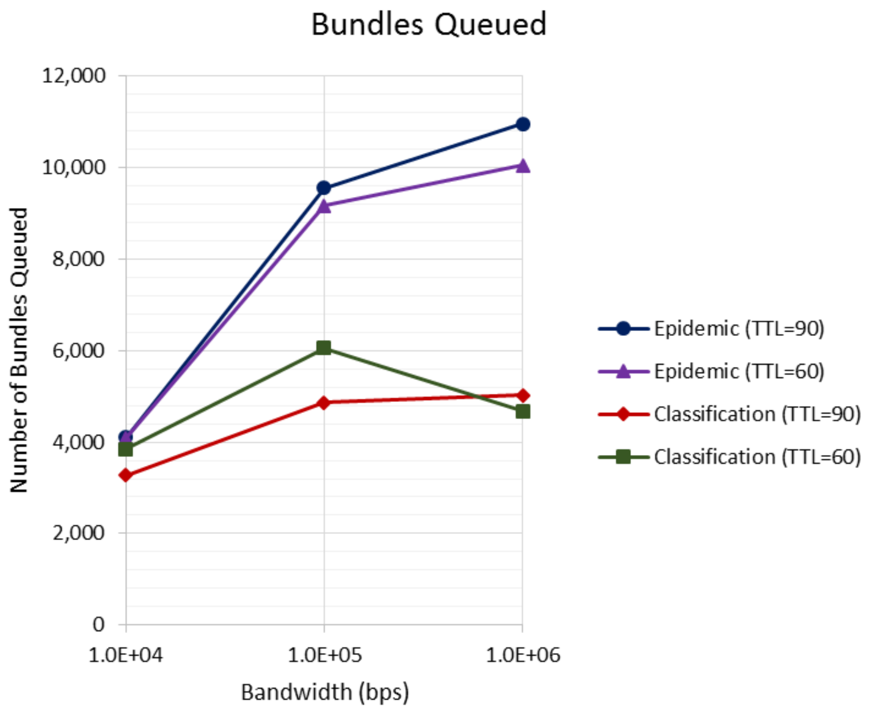


Figure 7.23. Epidemic versus Classification Bundles Queued

Figure 7.24 shows the delivery delay associated with Epidemic and Classification based routing. There is less delivery delay associated with the Classification based routing since fewer bundles need to be transmitted and queued. It can be seen that the use of selecting bundles to forward based on the classification approach discussed in Section 4.3 is able to reduce the total amount of traffic in the network while still delivering bundles at a satisfactory rate. The overall number of bundles replicated is reduced, which reduces the number of bundles waiting in the transmission queue as well as the number of bundles that expire before being delivered.

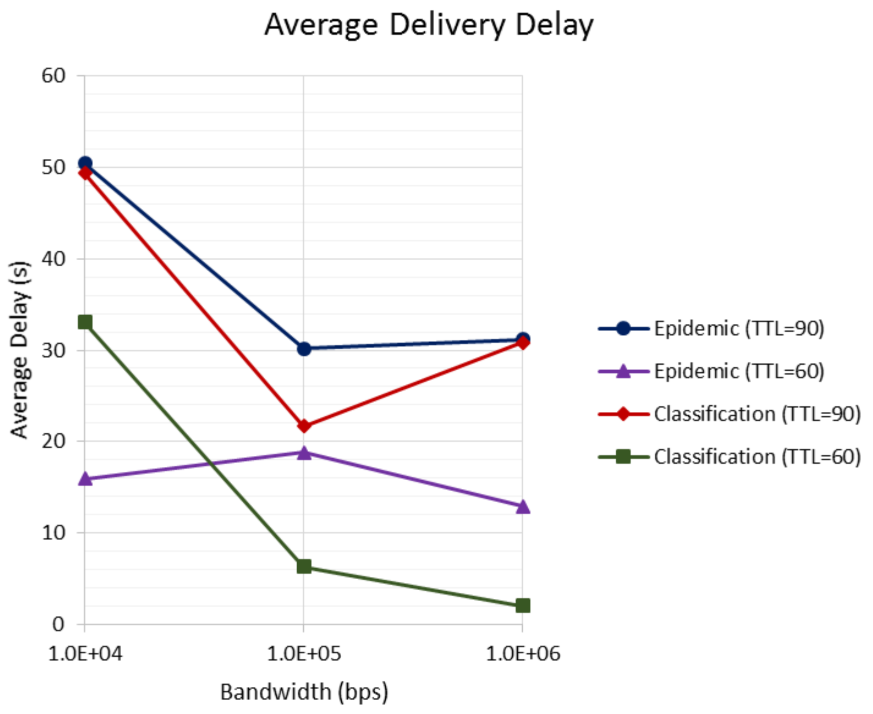


Figure 7.24. Epidemic versus Classification Bundle Delivery Delay

7.5 Clustering Results

This section discusses the metrics used for determining the best number of clusters, given the number of nodes and location points. Figure 7.25 shows a graphical representation of how node locations are divided into cluster regions. The location coordinates shown are based on the UTM (Universal Transverse Mercator) coordinate systems as discussed in Chapter 6. Initial emulations were performed to determine the best number of clusters to use for routing bundle filter. As discussed in section 7.1, the metrics of Silhouette Coefficient and Calinski-Harabaz Index indicate how well defined clusters are. This is loosely an indication of performance , since there are no true or correct ways to cluster the location points. Based on these results from varying the number of clusters, 9 clusters are chosen for the rest of the emulation.

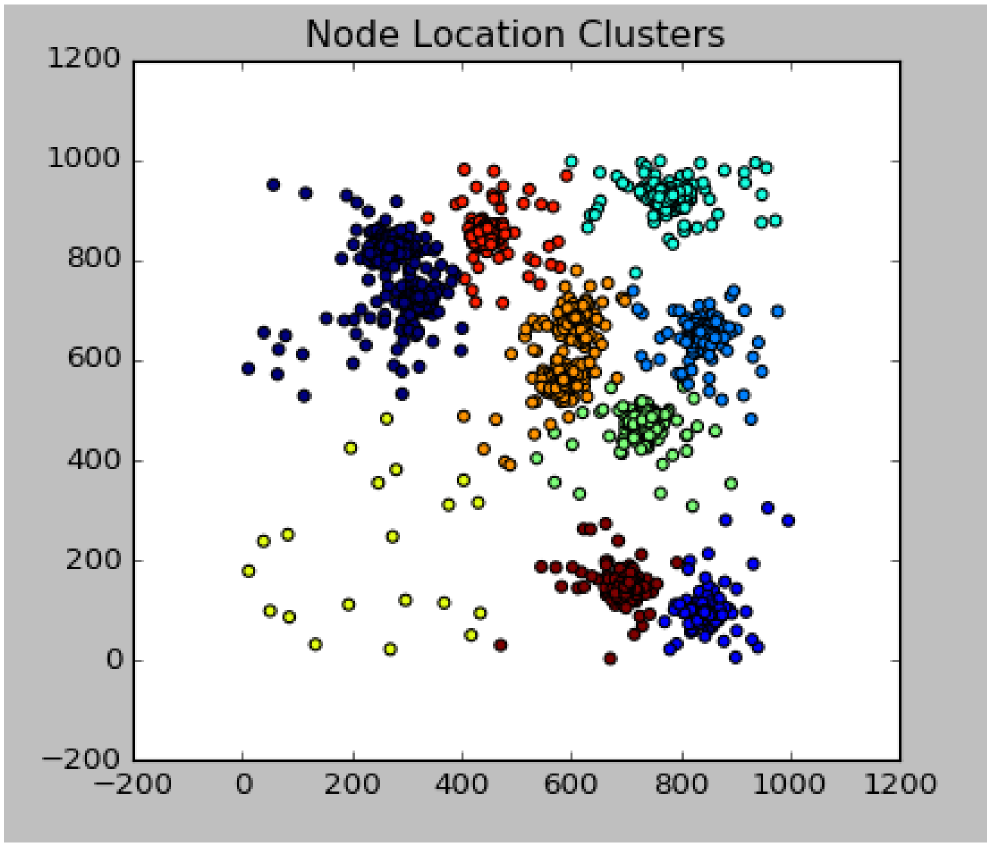


Figure 7.25. Graphical Representation of Location Clusters

Clusters	Silhouette Coefficient	Calinski-Harabaz Index
3	0.57	4343.4
4	0.62	5307.6
5	0.63	5635.2
6	0.63	5783.9
7	0.68	6398.3
8	0.68	7481.3
9	0.69	8854.8

Table 7.8. Cluster Metrics for Number of Clusters

7.5.1 Routing Results

Emulations were performed similarly to tests with the classification based routing. Data rates are varied from 10 kbps to 1 Mbps over three emulations and repeated with bundle expiration times of 60 and 90 seconds. Again, the total

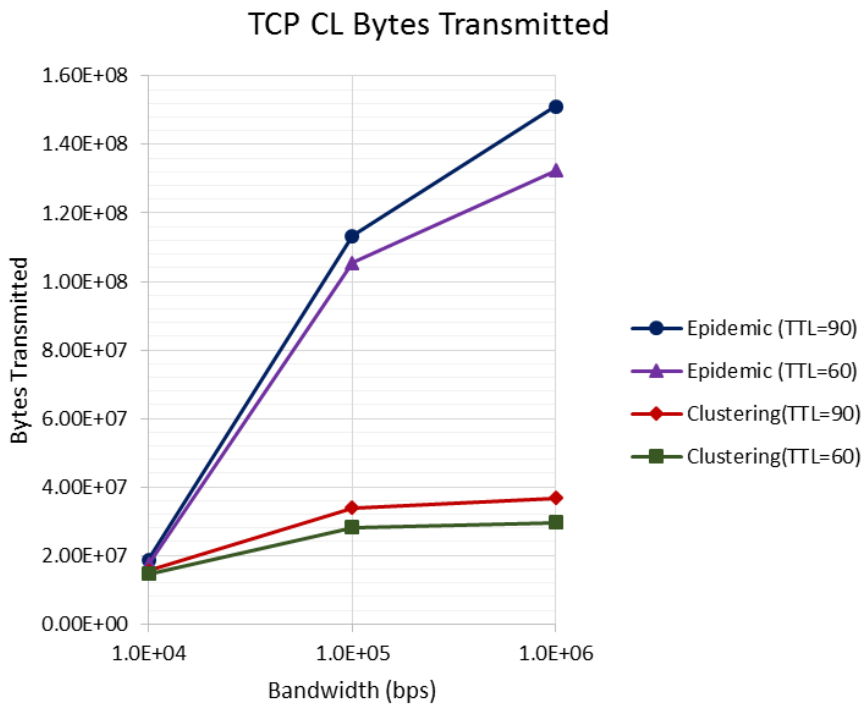


Figure 7.26. Epidemic versus Cluster Filter TCP CL Bytes Transmitted

number of bytes sent and received is reduced while still delivering a sufficient number of bundles.

Figures 7.26 and 7.27 show the total number of bytes transmitted and received by the TCP convergence layer. Figure 7.28 shows the bundle delivery ratio and Figure 7.29 shows the bundle delivery cost. Again, it can be seen that the cluster based approach is an improvement over Epidemic routing. Using the same concept as the classification based routing, which is to use historical data to predict the behavior of the nodes and subsequently reduce the number of unnecessary replications as shown in Figure 7.30.

Reducing the number of replications reduces the number of bundles that must be queued as shown in Figure 7.32. This will also reduce the number of

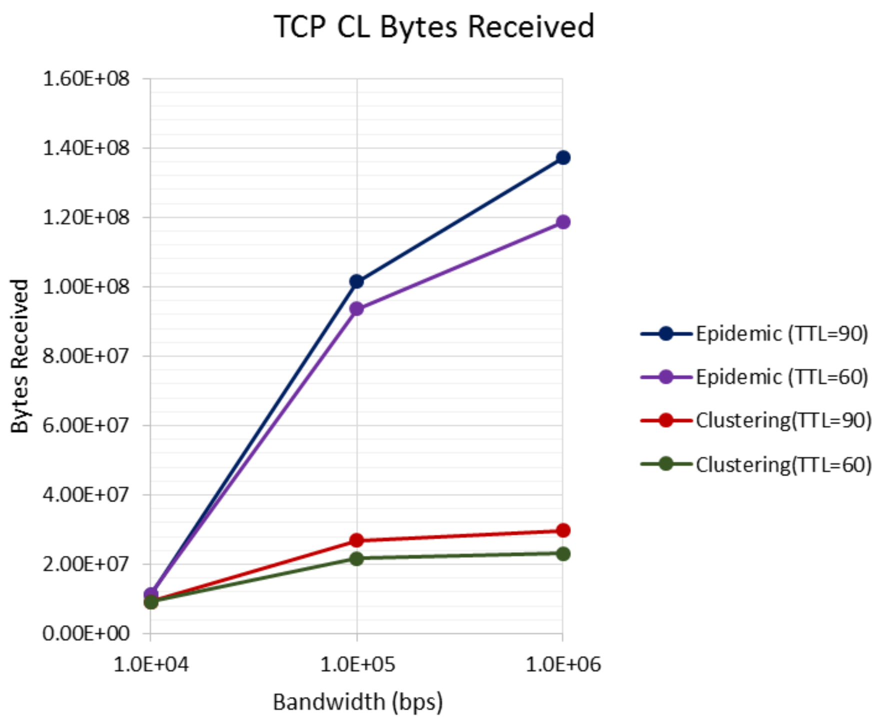


Figure 7.27. Epidemic versus Cluster Filter TCP CL Bytes Received

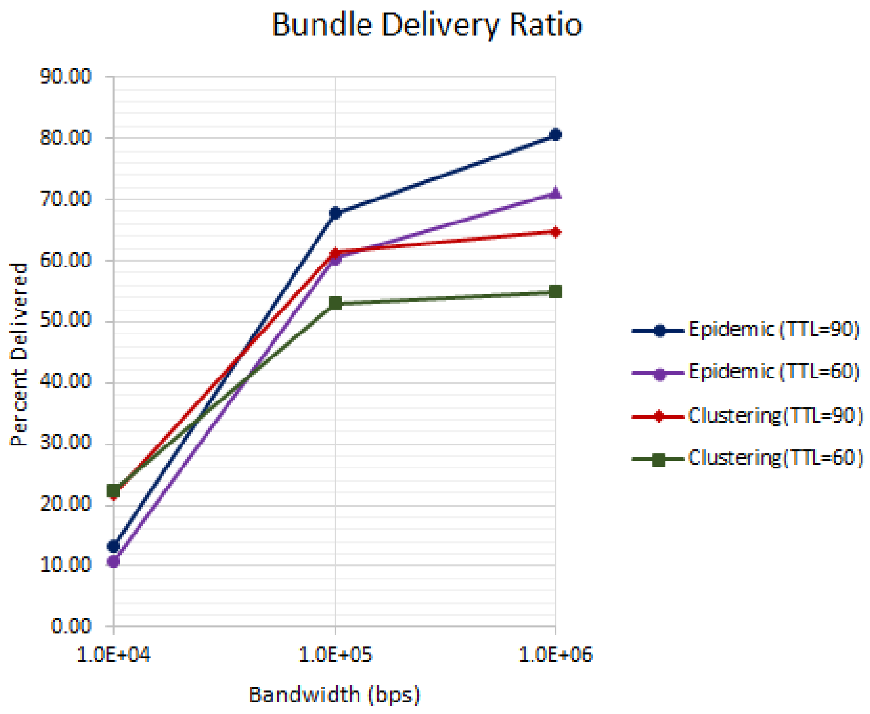


Figure 7.28. Epidemic versus Cluster Filter Bundle Delivery Ratio

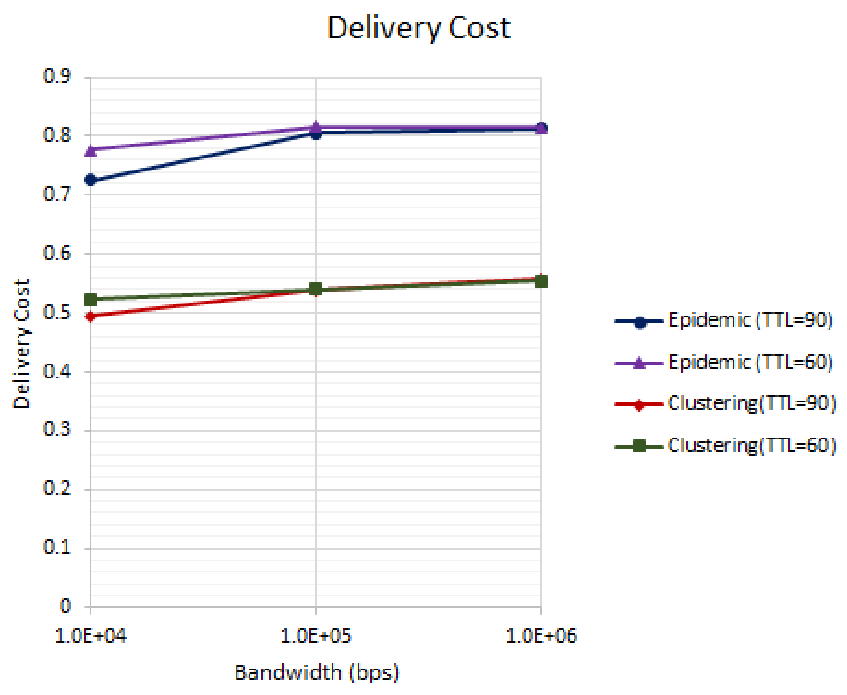


Figure 7.29. Epidemic versus Cluster Filter Bundle Delivery Cost

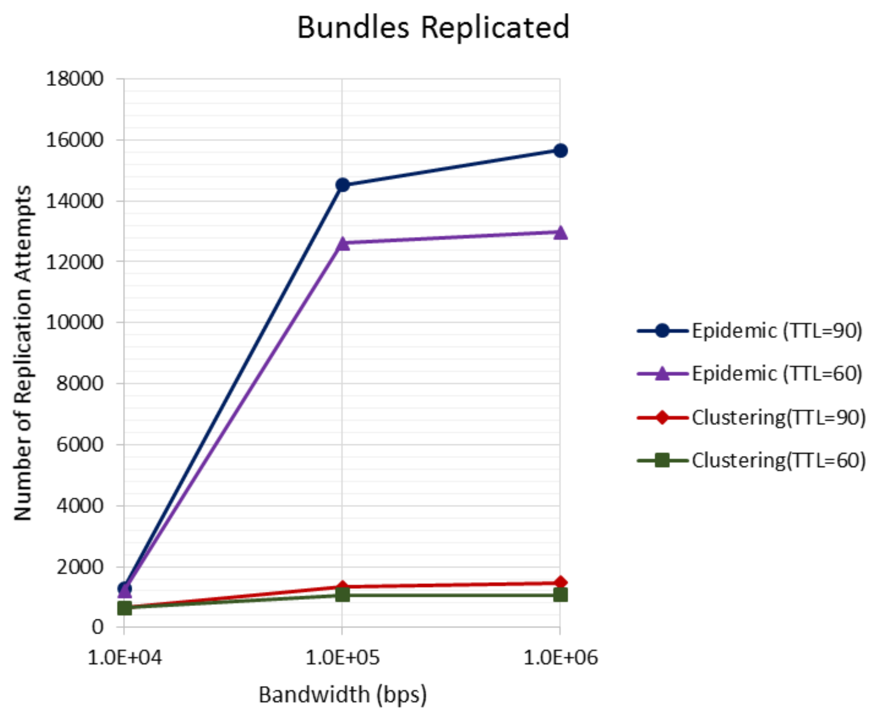


Figure 7.30. Epidemic versus Cluster Bundles Replicated

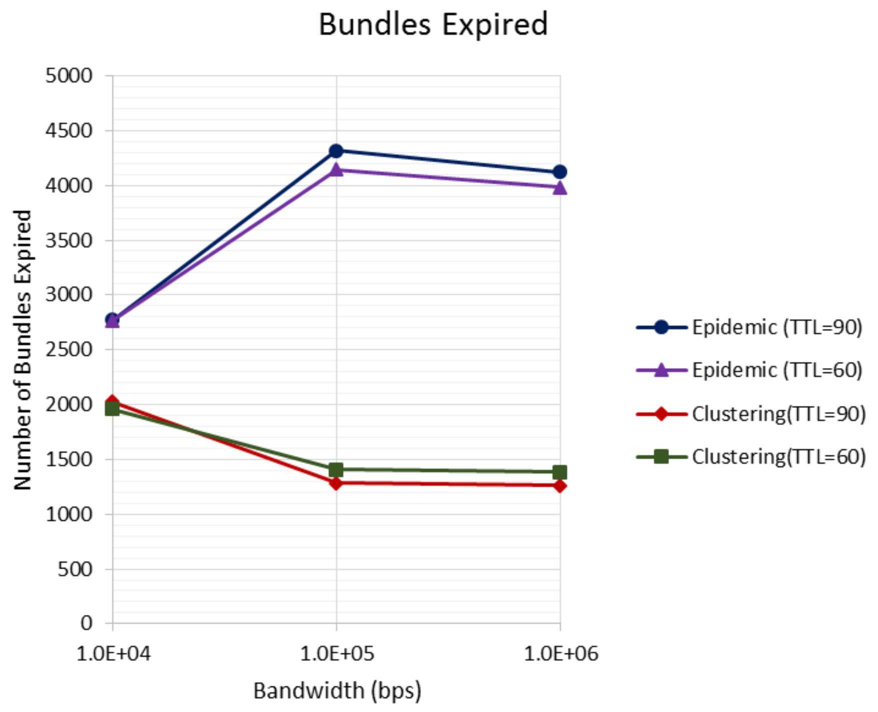


Figure 7.31. Epidemic versus Cluster Bundles Expired

bundles that expire before delivery as shown in Figure 7.31 and also reduce the overall delivery delay as shown in Figure 7.33.

Both the cluster and classification based bundle filters reduce the number of replicated bundles which in turn reduce the number of bundles that must be queued , transmitted and that may expire before delivery. This allows the entire network to perform more efficiently, saving system resources and well as reducing delivery times.

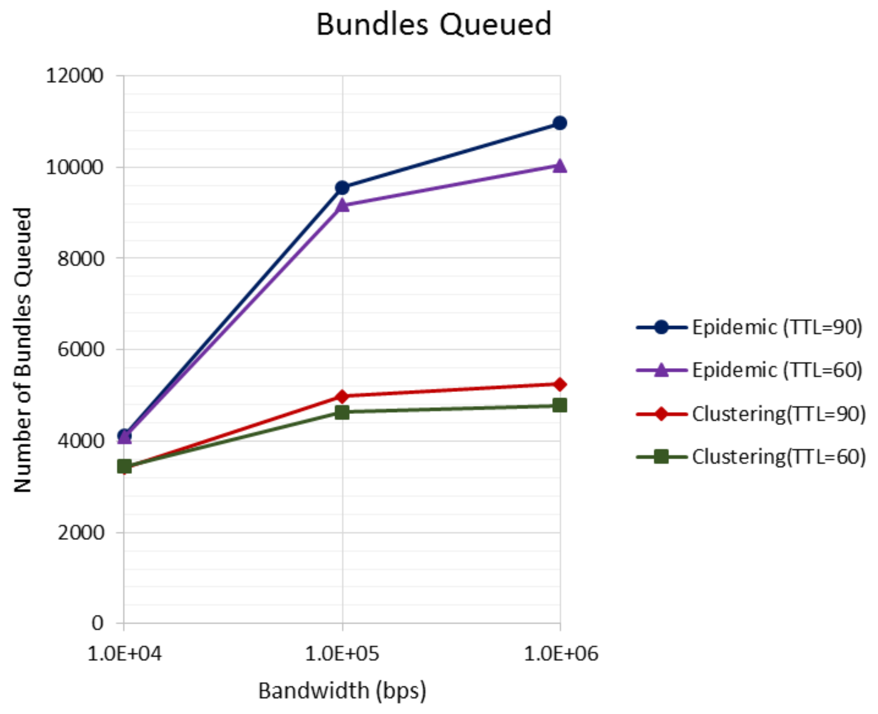


Figure 7.32. Epidemic versus Cluster Bundles Queued

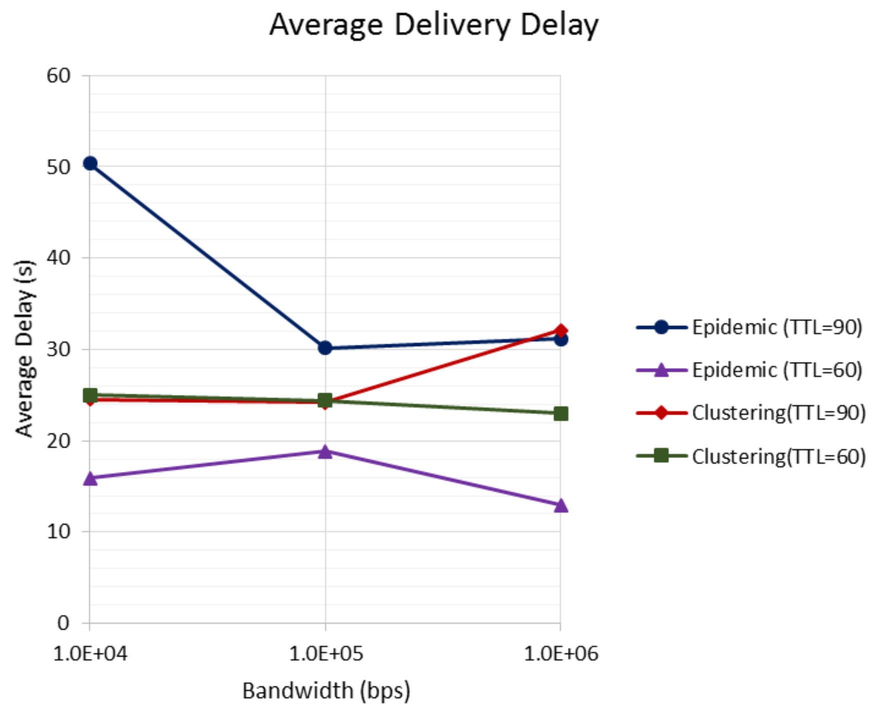


Figure 7.33. Epidemic versus Cluster Bundle Delivery Delay

8 Conclusions

This chapter covers a summary and conclusion of this dissertation as well as suggestions for future work. In this dissertation several well known machine learning techniques were applied to opportunistic DTN routing, including the reinforcement Q-Learning approach, an innovative multi-label classification bundle filter and a clustering based bundle filter for epidemic routing.

8.1 Summary

Chapter 1 discusses the introduction and motivations for cognitive and machine learning based networking and how these relate to the thesis contributions. Chapter 2 covered background material on the NASA SCaN Networks, the main concepts of DTNs and an introduction to the machine learning technique used in this work. Next, selected related works in DTN routing are discussed as well as other works that have applied related techniques such as Naive Bayes classifiers to DTN routing.

Chapter 4 discusses the approach taken to the contributions of this thesis, such as the development of a network and software architecture, and the pros and cons of distributed versus centralized learning architecture. From here, the majority of the work in this thesis used a centralized architecture, other than Q-Routing which is a distributed approach. Each technique is then discussed in detail. The development of the classification approach including the determination of suitable attributes, labels and problem formulation are discussed. Next, the development of the cluster based approach are covered.

In Chapter 5, the actual details of how these algorithms where implemented and tested are discussed. A variety of DTN bundle protocol implementations were investigated for the suitability of this work and IBR-DTN was selected as the best choice. The details of the modifications to IBR-DTN to integrate the machine learning models are covered. Chapter 6 discusses the development of a DTN testbed and the tradeoff between a simulation or emulation test environment. Several popular network simulators were explored and the final selection of the CORE emulator is discussed with details on how the emulation is performed using Linux containers.

Finally Chapter 7 covers metrics used to evaluate the learning and routing performance and a summary of results are given. It is shown that these route selection techniques are promising way to reduce overhead associated with epidemic routing while still providing a comparable delivery ratio.

8.2 Conclusion

The techniques of clustering and classification are used to reduce the consumption of resources such as bandwidth, processing time and data storage. Using historical data from the network, both techniques are able to predict which neighbors are the most likely to deliver bundles to their destination. For systems with less data storage capacity, limited bandwidth or limited processing capabilities, the overhead associated with epidemic routing can be burdensome. Both machine learning techniques reduce the number of bundles that must be replicated, while still delivering a satisfactory number of bundles.

While Q-Routing is an interesting approach for many network scenarios, the long delays associated with some types of DTNs may make estimation of the reward function (end-to-end delay) and propagation of the estimate difficult. For this reason, alternative algorithms such as supervised learning (classification) where considered. It is still very promising to consider Q-routing and other reinforcement learning techniques, however this will require additional consideration for DTNs. The use of reinforcement learning and Q-Routing with a cross-layer technique for obtaining information from lower protocol layers in particular may be a promising approach.

8.3 Future Work

This section covers several interesting avenues that could be pursued for future work.

8.3.1 Cross Layer Approach

A cross layer approach as discussed in [93] and [32] could be used to incorporate information from the lower protocol layers to add addition features to each reliability classifier for the nodes in the network. This information could also be used in a reinforcement learning scheme to reward routing decisions that produce less errors or retransmission in the underlying protocol layers.

8.3.2 Evolutionary Clustering

Evolutionary clustering [94],[95] is a type of clustering which takes into account points changing in time and updates clusters accordingly. This could potentially improve the performance of the cluster based filter which simply used a naive clustering method where all points within the period are clustered once and then matched back to the location and point in time.

8.3.3 Deterministic Routing

This work focused on an opportunistic DTN environment using replication routing techniques. Additional work could be done to apply machine learning techniques to networks that perform in a more deterministic manner with additional constraints.

8.3.4 Radio/Optical Link Models

The emulations conducted used a very basic model of the wireless communication links based on distance and data rate. However, EMANE has the ability to use custom defined radio models. The development of a higher fidelity radio model or development of an optical link model would be interesting. This was left for future work due to the scope of this dissertation being more focused on DTN software and routing algorithms.

References

[1] The Consultative Committee for Space Data Systems . Cislunar space internetworking architecture. Technical report, The Consultative Committee for Space Data Systems , 2006.

[2] The Consultative Committee for Space Data Systems. *Rationale, Scenarios, and Requirements for DTN in Space*. 2010.

[3] W. Ivancic, P. Paulsen, K. Vaden, and D. Ponchak. Cognitive Networking With Regards to NASA's Space Communication and Navigation Program. Technical report, NASA Glenn Research Center, 2013.

[4] Space Communications and Navigation (SCaN) Network Architecture Definition Document (ADD). Technical report, National Aeronautics and Space Administration, 2014.

[5] V. Cerf, S. Burleigh, and K. Fall. Delay-tolerant networking architecture. `https://tools.ietf.org/html/rfc4838`, 04 2007.

[6] K. Scott and S. Burleigh. Bundle Protocol Specification. `https://tools.ietf.org/html/rfc5050`, 2007.

[7] S. Burleigh. *Interplanetary Overlay Network (ION) Design and Operation*. JPL, 03 2016.

[8] D. Ellard, R. Altmann, and A. Gladd. DTN IP Neighbor Discovery (IPND). `https://tools.ietf.org/html/draft-irtf-dtnrg-ipnd-02`, 11 2012.

[9] P. Papadimitratos, M. Poturalski, P. Schaller, P. Lafourcade, D. Basin, S. Capkun, and J. P. Hubaux. Secure neighborhood discovery: a fundamental element for mobile ad hoc networking. *IEEE Communications Magazine*, 46(2):132–139, February 2008.

[10] G. Ansa, E. Johnson, H. Cruickshank, and Z. Sun. Mitigating Denial of Service Attacks in Delay-and Disruption-Tolerant Networks. In *Proceedings of the International Conference of Personal Satellite Services*, pages 221–234, 2010.

[11] W. D. Ivancic. Security analysis of dtn architecture and bundle protocol specification for space-based networks. In *2010 IEEE Aerospace Conference*, pages 1–12, March 2010.

[12] T. Mitchell. *Machine Learning*. McGraw-Hill, 1997.

[13] A. Barto and R. Sutton. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.

[14] S. Boshy. Packet Routing Using Reinforcement Learning: Estimating Shortest Paths in Dynamically Changing Networks. 1999.

[15] Jesse Read, Bernhard Pfahringer, Geoff Holmes, and Eibe Frank. Classifier Chains for Multi-label Classification. *Mach. Learn.*, 85(3):333–359, December 2011.

[16] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics*, pages 281–297, Berkeley, Calif., 1967. University of California Press.

[17] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[18] Aruna Balasubramanian, Brian Neil Levine, and Arun Venkataramani. Replication Routing in DTNs: A Resource Allocation Approach. *IEEE/ACM Trans. Netw.*, 18(2):596–609, April 2010.

[19] T. Spyropoulos, A. Valisakos, and Y. Zhang, editors. *Delay Tolerant Networks :Protocols and Applications*. CRC Press, 2012.

[20] S. Ahmed and S. Kanhere. A Bayesian Routing Framework for Delay Tolerant Networks. In *In Proceedings of the 2010 IEEE Wireless Communications and Networking Conference*, 2010.

[21] L. Portugal-Poma, C. Marcondes, H. Senger, and L. Arantes. Applying Machine Learning to Reduce Overhead in DTN Vehicular Networks. In *Simposio Brasileiro de Redes de Computadores e Sistemas Distribuidos – SBRC 2014*, 2014.

[22] P. Mundur and M. Seligman. Delay tolerant network routing: Beyond epidemic routing. In *2008 3rd International Symposium on Wireless Pervasive Computing*, pages 550–553, May 2008.

[23] T. Spyropoulos, K. Psounis, and C. Raghavendra. Spray and Wait: An Efficient Routing Scheme for Intermittently Connected Mobile Networks. In *SIGCOMM'05 Workshops*, 2005.

[24] M. Demmer and K. Fall. DTLSR: Delay Tolerant Routing for Developing Regions, booktitle = Proceedings of the 2007 Workshop on Networked Systems for Developing Regions. pages 5–1 – 5–6, 2007.

[25] M. Demmer. *A Delay Tolerant Networking and System Architecture for Developing Regions*. PhD thesis, University of California at Berkeley, 2008.

[26] A. Lindgren and A. Doria. Probabilistic Routing Protocol for Intermittently Connected Networks. https://tools.ietf.org/html/draft-lindgren-dtnrg-prophet-02, 2006.

[27] S. Grasic, E. Davies, A. Lindgren, and A. Doria. The Evolution of a DTN Routing Protocol - PRoPHETv2. In *Proceedings of the 6th ACM Workshop on Challenged Networks*.

[28] A. Balasubramanian, B. Levine, and A. Venkataramani. Replication Routing in DTNs: A Resource Allocation Approach. *IEEE/ACM Transactions on Networking*, 18:596–609, 2010.

[29] S. Burleigh. Contact Graph Routing. https://tools.ietf.org/html/draft-burleigh-dtnrg-cgr-00, 2009.

[30] N. Bezirgiannidis. *Accurate Estimation of End-to-End Delivery Delay in Space Internets: Protocol Design and Implementation*. PhD thesis, 2015.

[31] N. Bezirgiannidis, C. Caini, D. Padalino, M. Ruggieri, and V. Tsaoussidis. Contact Graph Routing Enhancements for Delay Tolerant Space Communications. In *Proceedings of the 7th Advanced Satellite Multimedia Systems Conference and the 13th Signal Processing for Space Communications Workshop (ASMS/SPSC)*, 2014.

[32] R. Dudukovich and A. Hylton. A Machine Learning Concept for DTN Routing. In *Proceeding of the 2017 IEEE International Conference on Wireless for Space and Extreme Environments*, 2017.

[33] M. D. Colagrosso. A classification approach to broadcasting in mobile ad hoc network. In *IEEE International Conference on Communications, 2005. ICC 2005. 2005*, volume 2, pages 1112–1117 Vol. 2, May 2005.

[34] M. Littman and J. Boyan. Packet Routing in Dynamically Changing Networks: A Reinforcement Learning Approach. In *Proceedings of the 6th International Conference on Neural Information Processing Systems*, pages 671–678, 1993.

[35] Asaf Valadarsky, Michael Schapira, Dafna Shahaf, and Aviv Tamar. A machine learning approach to routing. *CoRR*, abs/1708.03074, 2017.

[36] Grigorios Tsoumakas and Ioannis Katakis. Multi-label classification: An overview. *Int J Data Warehousing and Mining*, 2007:1–13, 2007.

[37] Michael Doering, Sven Lahde, Johannes Morgenroth, and Lars Wolf. IBR-DTN: An Efficient Implementation for Embedded Systems. pages 117–120, 01 2008.

[38] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[39] P. Szymański and T. Kajdanowicz. A scikit-based Python environment for performing multi-label classification. *ArXiv e-prints*, February 2017.

[40] *Advances in Delay Tolerant Networks (DTNs) Architecture and Enhanced Performance*. Woodhead Publishing, 2015.

[41] *DTN2 Manual*.

[42] A. Krifa, C. Barakat, and T. Spyropoulos. An optimal joint scheduling and drop policy for Delay Tolerant Networks. In *2008 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, pages 1–6, June 2008.

[43] S. Burleigh. Interplanetary Overlay Network: An Implementation of the DTN Bundle Protocol. In *2007 4th IEEE Consumer Communications and Networking Conference*, pages 222–226, Jan 2007.

[44] J. Morgenroth. IBR-DTN - A Modular and Lightweight Implementation of the Bundle Protocol. https://github.com/ibrdtn/ibrdtn.

[45] Wolf-Bastian Pöttner, Johannes Morgenroth, Sebastian Schildt, and Lars Wolf. Performance Comparison of DTN Bundle Protocol Implementations. In *Proceedings of the 6th ACM Workshop on Challenged Networks*, CHANTS '11, pages 61–64, New York, NY, USA, 2011. ACM.

[46] S. Schildt, J. Morgenroth, W. Pöttner, and L. Wolf. IBR-DTN: A Lightweight, Modular and Highly Portable Bundle Protocol Implementation. *Electronic Communications of the EASST*, 37, 2011.

[47] M. Feldmann and F. Walter. μPCN-A Bundle Protocol Implementation for Microcontrollers. In *2015 International Conference on Wireless Communications Signal Processing (WCSP)*, pages 1–5, Oct 2015.

[48] S. Burleigh and E. Birrane. Toward a Communications Satellite Network for Humanitarian Relief. In *Proceedings of the 1st International Conference on Wireless Technologies for Humanitarian Relief*.

[49] J. Morgenroth. *Event-driven Software-Architecture for Delay- and Disruption-Tolerant Networking*. PhD thesis, Technical University of Braunschweig, 2015.

[50] David Abrahams. Boost.python. `https://www.boost.org/doc/libs/1_68_0/libs/python/doc/html/index.html`.

[51] OMNet++ Discrete Event Simulator. `https://omnetpp.org/`.

[52] W. Ivancic. DTN Network Management. `https://datatracker.ietf.org/meeting/77/materials/slides-77-dtnrg-3`.

[53] J. Ahrenholz, T. Goff, and B. Adamson. Integration of the CORE and EMANE Network Emulators. In *Proceedings of the 2011 IEEE Military Communications Conference*, pages 1870–1875. IEEE, 2011.

[54] A. Keranen, J. Ott, and T. Karkkainen. The ONE Simulator for DTN Protocol Evaluation. In *Proceedings of the 2nd International Conference on Simulation Tools and Techniques*. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2009.

[55] ns-3. `https://www.nsnam.org/`.

[56] B. Niehoefer, S. Šubik, and C. Wietfeld. The CNI Open Source Satellite Simulator Based on OMNeT++. In *Proceedings of the 6th International ICST Conference on Simulation Tools and Techniques*, pages 314–321, 2013.

[57] S. L. F. Maia, E. R. Silva, , and P. R. Guardieiro. A Proposal of a Simulator Based on OMNeT++ for Delay/Disruption Tolerrant Networks Comprising Population of Nodes with High Level of Heterogeneity. In *Proceedings of the 2013 International Workshop on Telecommunications (IWT)*, 2013.

[58] J. A. Fraire, P. Madoery, S. Burleigh, and et al. Assessing Contact Graph Routing Performance and Reliability in Distributed Satellite Constellations. *Journal of Computer Networks and Communications*, 2017.

[59] M. Rodolfi. DTN Discovery and Routing: From Space Applications to Terrestrial Networks. Master's thesis, 2014.

[60] M. Feldmann, F. Walter, and R. Bohm. Routing in Ring Road Networks: Leveraging a Spot of Maximum Knowledge. In *Proceeding of the 2017 IEEE International Conference on Wireless for Space and Extreme Environments*, 2017.

[61] S. Burleigh, C. Caini, J. J. Messina, and M. Rodolfi. Toward a Unified Routing Framework for Delay-Tolerant Networking. In *Proceeding of the 2016 IEEE*

International Conference on Wireless for Space and Extreme Environments, 2016.

[62] K. Massri, A. Vitaletti, A. Vernata, and I. Chatzigiannakis. Routing Protocols for Delay Tolerant Networks:A Reference Architecture and a Thorough Quantitative Evaluation. *Journal of Sensor and Actuator Networks*, 2016.

[63] S. Endres. Simulation And Emulation of the Space Networking Environment. Master's thesis, Case Western Reserve University, 2005.

[64] B. Barritt. *The Modeling, Simulation, and Operational Control of Aerospace Communication Networks*. PhD thesis, Case Western Reserve University, 2017.

[65] R. Martinez-Vidal, T. Henderson, and J. Borrell. Implementation and Evaluation of Licklider Transmission Protocol (LTP) in ns-3. In *Workshop on ns-3*, 2015.

[66] S. Yang, J. Jiang, and P. Chen. Using ns2 for a new routing method to note the packet drop problem in dtns. In *2012 6th International Conference on New Trends in Information Science, Service Science and Data Mining (ISSDM2012)*, pages 167–172, Oct 2012.

[67] D. Bittencourt, E. Mota, E. N. Silva, and C. B. Souza. Towards realism in dtn performance evaluation using virtualization. In *2013 IFIP Wireless Days (WD)*, pages 1–7, Nov 2013.

[68] D. Bittencourt et al. Towards Realism in DTN Performance Evaluation Using Virtualization. In *2013 IFIP Wireless Days (WD)*. IEEE, 2013.

[69] J. Ahrenholz. Comparison of core network emulation platforms. In *2010 - MILCOM 2010 MILITARY COMMUNICATIONS CONFERENCE*, pages 166–171, Oct 2010.

[70] S. Endres, M. Griffith, B. Malakooti, K. Bhasin, and A. Holtz. Space Based Internet Network Emulation for Deep Space Mission Applications. AIAA, 2004.

[71] Analytical Graphics, Inc. Systems Tool Kit. https://www.agi.com/products/engineering-tools.

[72] U.S. Naval Research Laboratory. Common Open Research Emulator. https://www.nrl.navy.mil/itd/ncs/products/core.

[73] U.S. Naval Research Laboratory. Extendable Mobile Ad-hoc Network Emulator. https://www.nrl.navy.mil/itd/ncs/products/emane.

[74] U.S. Naval Research Laboratory. Scripted Display Tools. `https://www.nrl.navy.mil/itd/ncs/products/sdt`.

[75] J. Ahrenholz. Comparison of CORE Network Emulation Platforms. In *Proceedings of the 2010 IEEE Military Communications Conference*, pages 166–171. IEEE, 2010.

[76] *CORE Documentation Release 4.8*.

[77] K. Scott. MITRE Publishes Updated DTN Development Kit. `http://ipnsig.org/2017/04/14/mitre-publishes-updated-dtn-development-kit/`, 4 2017.

[78] ArchWiki. Linux Containers. `https://wiki.archlinux.org/index.php/Linux_Containers`.

[79] M. Krasnyansky. Universal TUN/TAP Device Driver. `https://www.kernel.org/doc/Documentation/networking/tuntap.txt`, 2000.

[80] Yong Wang, Pei Zhang, Ting Liu, Chris Sadler, and Margaret Martonosi. Movement data traces from princeton zebranet deployments. CRAWDAD Database. http://crawdad.cs.dartmouth.edu/, 2007.

[81] BonnMotion A Mobility Scenario Generation and Analysis Tool. `http://sys.cs.uos.de/bonnmotion/`.

[82] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[83] `http://www.numpy.org/`.

[84] S. Ahmed and S. S. Kanhere. A bayesian routing framework for delay tolerant networks. In *2010 IEEE Wireless Communication and Networking Conference*, pages 1–6, April 2010.

[85] Peter Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *J. Comput. Appl. Math.*, 20(1):53–65, November 1987.

[86] Marcin Kozak. “a dendrite method for cluster analysis” by caliński and harabasz: A classical work that is far too often incorrectly cited. *Communications in Statistics - Theory and Methods*, 41(12):2279–2280, 2012.

[87] Alice Zheng. *Evaluating Machine Learning Models*. O’Reilly Media, 2015.

[88] Krzysztof Dembczyński, Willem Waegeman, Weiwei Cheng, and Eyke Hüllermeier. Regret analysis for performance metrics in multi-label classification: The case of hamming and subset zero-one loss. In José Luis Balcázar, Francesco Bonchi, Aristides Gionis, and Michèle Sebag, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 280–295, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.

[89] A. Santos, A. Canuto, and A. Feitosa Neto. A Comparative Analysis of Classification Methods to Multi-label Tasks in Different Application Domains. *International Journal of Computer Information Systems and Industrial Management Applications*, 3:218–227, 2011.

[90] M. Zhang and Z. Zhou. A Review on Multi-Label Learning Algorithms. *IEEE Transactions on Knowledge and Data Engineering*, 26(8), August 2014.

[91] D. Raible and A. Hylton. Networked Operations of Hybrid Radio Optical Communications Satellites. In *AIAA International Communications Satellite Systems Conference*, 2014.

[92] V. Almeida, A. Oliveira, D. F. Macedo, and J.M.S. Nogueira. Performance Evaluation of MANET and DTN Routing Protocols. In *IFIP Wireless Days*, pages 1–6, November 2012.

[93] D. Kliazovich, F. Granelli, and Nelson L. S. Da Fonseca. *Handbook on Sensor Networks*, chapter Architectures and Cross-Layer Design for Cognitive Networks. World Scientific Publishing Co Inc, 2009.

[94] R. Langone, R. Mall, and J. A. K. Suykens. Clustering data over time using kernel spectral clustering with memory. In *2014 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 1–8, Dec 2014.

[95] Kevin S. Xu, Mark Kliger, and Alfred O. Hero, Iii. Adaptive evolutionary clustering. *Data Min. Knowl. Discov.*, 28(2):304–336, March 2014.