Dynamic Selection of Network Protocols for Group Communications in Mobile Ad-hoc Networks

A Thesis

Submitted to the Faculty

of

Drexel University

by

Aaron M. Rosenfeld

in partial fulfillment of the

requirements for the degree

of

Master of Science in Computer Science

April 2014

Dedications

This thesis is dedicated to my father, the late Dr. Stuart Alan Rosenfeld, who instilled in me an excitement for learning and discovery. Without his years of support, encouragement, and love, this thesis would not have been possible.

Acknowledgments

Foremost, I would like to thank my advisor, Professor William C. Regli, for introducing me to academic research and expanding my education outside of the classroom. I am forever grateful to the numerous opportunities and years of guidance he has provided.

Additionally, I want to express my appreciation to Joseph P. Macker for providing me the requisite experience to write a thesis on this topic and allowing me the freedom to investigate many interesting topics. His guidance and input have been invaluable.

My sincere thanks also goes to the friends and colleagues who have directly or indirectly contributed to this thesis including Robert N. Lass, Dustin S. Ingram, Duc N. Nguyen, and Thomas Wambold.

My mother Robin Kielkowski and step-father John Leighty deserve a special acknowledgement for their love and motivation throughout my education.

Lastly, I thank Stacey Wrazien for her technical input, but most of all for her unwaivering love, support, and encouragement during the writing of this thesis.

Table of Contents

List of Tables

List of Figures

Abstract

Dynamic Selection of Network Protocols for Group Communications in Mobile Ad-hoc Networks
Aaron M. Rosenfeld
William C. Regli, Ph.D.

This thesis addresses the topic of dynamically selecting protocols at various levels of network stacks in challenged environments, specifically those with message loss, long-term fragmentation, and high mobility, in an effort to meet the demands of group-based messaging applications.

Currently, developers select protocols based on a static set of assumptions about the underlying network and application requirements. This thesis introduces a method of sensing the network state, merging this with similar information from peers, and dynamically changing the underlying protocols. This alleviates the need for developers to select protocols and instead assert message requirements.

Further, since application instances are involved in group communications, they likely act as such from a mobility perspective, causing different portions of the network to have drastically different properties. For example, there may be clusters of nodes in certain locations, but minimal connectivity between them. The proposed solution allows systems to adapt to these situations as protocols may be interchanged at any time, allowing the best to be used in any given scenario.

The thesis first establishes a formal definition of the problem space, and then proposes a solution utilizing Markov Random Fields to classify the network. This classification is then used to dynamically select the protocols utilized by the network stack.

The Dynamic Protocol Selection Middleware (DPSM) is introduced as the implementation of this approach. Using this middelware, the effectiveness of the approach is tested in both random group environments and real-world scenarios. In general, DPSM delivered at least as many messages as any statically selected protocol, while delivering substantially more messages in many scenarios with only modest increases in overhead or latency.

Chapter 1: Introduction

This thesis addresses the topic of dynamically selecting an optimal set of communication protocols for applications participating in group communications in Mobile Ad-hoc Networks (MANETs). Group communications may be defined as the "many-to-many dissemination and reception of information in a network of two or more collaborating computer systems." Examples of such include chat rooms where participants communicate with all others, shared whiteboards with multiple contributors and viewers, and situational awareness applications where the location of each user is shared.

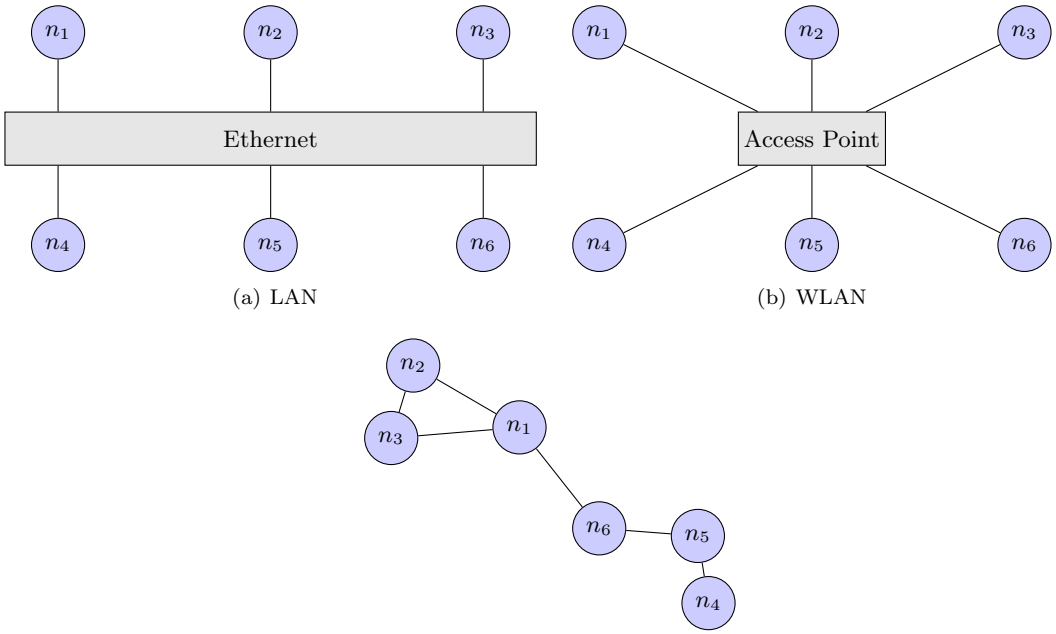
Applications requiring network access utilize a stack of protocols on each host which facilitate the transmission and reception of information. Each layer of the stack sequentially performs specific tasks when a message is sent or received as detailed in Section 2.2. The protocol for each of the stacks' layers is traditionally chosen during program creation by the developer.

This decision is made based on a set of messaging requirements which include reliability, ordering, and long-term delivery guarantees. Once these requirements are determined, a set of protocols best suited to provide the guarantees is selected for use.

For many "traditional" networks, such as small Local Area Networks (LANs), or even larger scale deployments such as enterprise server farms, this selection may be simple. For example, if messages *must* be delivered, TCP [29] may be used, while UDP [28] may be adequate if message loss is acceptable. Even in the presence of varying traffic load, or infrequent, minor changes to the network infrastructure, statically choosing protocols is likely adequate.

This also applies to most wireless LANs (WLANs) with an access point. The access point can be considered a hub where all traffic from network nodes is exchanged, whether to another node on the network or an external system.

On the contrary, as compared in Figures 1.1(a), 1.1(b), and 1.1(c), MANETs are self-organizing networks, without a centralized communication hub, and may be highly dynamic. Unlike traditional networks, messages may need to be forwarded by peers to their destination in the presence of



(a) LAN

(b) WLAN

(c) MANET

Figure 1.1: Comparison of a Local Area Network and a MANET

uncertain routing, possible data loss, and variable latency.

Another key difference of MANETs important to this thesis is the uncertainty of the network state (e.g. network graph, bandwidth availability) at any given point of time. That is, the network topology is unknown, peers may join or leave the network without notice, and messages may be lost due to a variety of factors. These uncertainties make selecting protocols during application creation very difficult as their performance may change drastically over a given deployment.

For example, consider a number of nodes using a very naïve protocol to guarantee message delivery by simply retransmitting *every* message it has ever sent. In a sparse network, so long as the destination(s) eventually establish connection to a sender, messages will likely be received, albeit at a high bandwidth cost. However, if the network becomes dense, messages will likely not be received because multiple instances of the protocol will cause contention.

1.1 Proposed Solution

This thesis proposes a new method of selecting protocols, not during program creation, but *dynamically* while an application is running, changing based on the underlying network conditions. Since application developers are generally concerned about messaging guarantees, it is sufficient for them to stipulate delivery requirements rather than specify protocols themselves.

The proposed solution formally defines the problem of protocol selection and then introduces a method of dynamically changing protocols, at various levels of the stack, based on locally- and remotely-sensed network state. Finally, a generic middleware which applications can utilize, instead of using the Operating System's sockets, is introduced and tested.

Each middleware instance senses local network information (e.g. number of neighbors), shares it with peer instances, receives peer information, and makes a logical choice based on the application's messaging needs and the underlying network state.

1.2 Contributions

Through formalizing the problem, implementing a solution, and evaluating its performance relative to existing techniques, this thesis addresses the following questions:

1. *How can a protocol stack be selected dynamically during runtime to provide application-specified message guarantees?*
2. *Can a middleware be used to abstract traditional sockets from applications, and provide a unified interface to a possibly dynamic network stack?*
3. *What components would the architecture of such a middleware require?*
4. *How effective is dynamic protocol selection versus the traditional static assignment?*

1.3 Motivation

1.3.1 Message Delivery in MANETs

With the proliferation of wireless devices, MANETs have become desirable in many environments. They have seen use in sensor networks due to their quick ability to deploy, share information with

peers, and adapt to disruption. The military makes extensive use of them in tactical deployments since they provide a robust means of communication without creating a central point of failure [1, 32, 33] and can partition easily.

As MANETs are applied in more extreme and challenging environments the number of factors affecting protocol performance will continue to rise.

Previous work [31] compared the performance of different Session and Transport layer protocols in challenged, mobile environments. The network connectivity and number of groups in the network were varied, and no one protocol achieves optimal performance for a given metric across all networks. Therefore, a developer could not select a static set of optimal protocols for any given deployment.

The rate at which protocols are developed for distributed environments like MANETs further complicates this selection problem.

This complicated, ever changing selection process motivates the need for an automated approach to dynamically switch protocols providing adequate levels of performance as the network evolves.

1.3.2 Whiteboard Example

One task an application may desire within a MANET is for its instances to share a common whiteboard. Imagine, for example, that there are three people, Alice, Bob, and Charlie, on different nodes drawing on the common whiteboard.

When they are fully connected, each time a person draws on the whiteboard, they flood pertinent information it to the network, sending it to each of their immediate neighbors, who in turn forward it exactly once to their neighbors. Assuming sufficient bandwidth and no packet loss, this will generally work well and provide a consistent view of the whiteboard.

At a later time, Alice and Bob move away from Charlie and the network fragments. Alice and Bob can still communicate, but Charlie will not receive any messages until he re-connects with them. Even so, any messages Charlie missed while he was separated will be lost from his whiteboard forever. Since the goal is to maintain a consistent whiteboard for everyone a process to rectify discrepancies is necessary.

One method could be to have the three participants frequently rebroadcast a list of all shapes

on their whiteboard. This way, Charlie will receive the ones he missed. When the network is sparse, this may work well since excess messaging is unlikely to interfere with new shapes being transmitted, or even other applications sending data.

Another method could be to occasionally have each of the participants talk to the others and determine which shapes are missing. This is potentially less taxing on the network since little information must be exchanged when there is no disparity in their whiteboards. However, this is a slower process.

These two methods demonstrate a need to adaptively select protocols during runtime; it seems optimal to use the first method if the network is sparse, or there is little other traffic with which to contend, but the second seems more appropriate in all other circumstances.

Chapter 2: Background

2.1 Mobile Ad-hoc Networks

A Mobile Ad-hoc Network (MANET) is a self-organizing network of *nodes*, each of which has at least one wireless *network device*. Two nodes are *neighbors* (sometimes called *one-hop neighbors* for clarity) and share a *link* if they are able to communicate directly without needing any intermediary forwarding. Two nodes are *n-hop neighbors* if they can communicate and the shortest path between them has exactly links n. Any two nodes which can communicate are called *connected*.

A *fragment* of the network is a subset of nodes which are connected, but not to every node in the network.

Two nodes which are not neighbors, but are connected must have their messages *routed* by intermediary nodes with the use of a routing algorithm.

Unlike wired networks where physical limitations prohibit mobility, MANET nodes are independent, sharing no physical connecting, and may move arbitrarily creating connections to some nodes, and becoming disconnected from others.

MANETs are also different from common Wireless Local Area Networks (WLANs) wherein one or more access points relay traffic between nodes on the network, or those beyond.

2.2 Protocols and the OSI Stack

A *protocol* is a set of rules and data formats used to exchange information between two network devices, generally on different nodes.

To abstract the responsibilities of different protocol classes, each protocol is placed into one of the seven OSI network stack categories as shown in Table 2.1.

When an application sends a message, it passes through each layer, down to the Physical. Once transmitted across the physical medium, after passing through the lowest three layers of intermediary nodes (if any), it travels up the stack at the destination as shown in Figure 2.1

Table 2.1: The standard OSI network stack's seven layers.

Layer	Example
Application	FTP, SSH
Presentation	MIME
Session	SSL, SOCKS
Transport	TCP, UDP
Network	IPv4, IPv6
Link	PPP, ATM
Physical	802.11, DSL

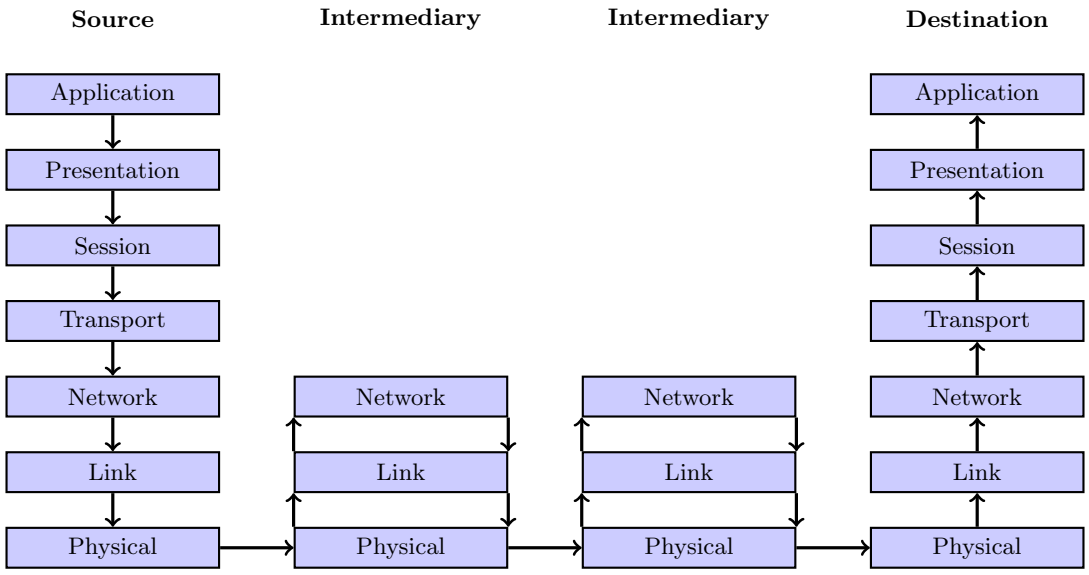


Figure 2.1: Data flow from source to destination with two intermediary nodes.

2.3 Transmission Methods

There are a number of paradigms used for transmitting messages across a network. The most basic is *unicast* which is used to send a message to a specific, individual, remote node. This thesis does not utilize any unicast messaging as it targets group communications.

Three other paradigms exist which are critical to group communications. The first is *broadcast*. When a message is broadcast, all nodes connected to the sender, by any number of links, receive the message. To facilitate this, a routing protocol must be used to forward the message from node to node. The most applicable for this thesis is *classical flooding*. In this protocol, every node will rebroadcast a given message exactly once.

The second is *link-local broadcast* where only one-hop neighbors receive the message and routing is not needed in wireless networks.

Finally, *multicast* which sends messages to a specific set of nodes on the network. Like broadcast, this requires a routing algorithm.

2.4 Group Communications

This thesis is limited to protocol selection for *group communications*, which can be defined as a set of network nodes which publish and subscribe to destinations (e.g. a chat room). Any node can transmit and receive from any destination.

This scoping limits a number of factors impacting protocol selection and the implementation of a generic middleware. First, it alleviates the need to know the recipients for a transmitted message since it is broadcast to all other nodes. If another node receives a message addressed to a destination to which it is not subscribed, the middleware can simply ignore the message.

It also reduces the number of protocols which could be used at each layer of the OSI stack since many protocols do not support one-to-many messaging. It is possible to emulate one-to-many messaging with a one-to-one protocol, by sending the same message to each recipient. However, this requires knowledge of other nodes on the network which is not generally possible in a MANET.

2.5 Reliable and Persistent Data Delivery

Reliability and persistence are tightly related protocol attributes. They provide a means of delivering messages even if the initial transmission is not received by the destination.

In the context of this thesis, *reliability* is defined as the ability to deliver a singular error-free message to neighbors within a bounded period of time. This is distinguished from *persistence* which delivers messages to neighbors in a possibly unbounded quantity of time. Generally, reliability protocols are short-term and persistence protocols are long-term, although they can both act similarly for medium-term delivery.

TCP [29] is an example of a reliable protocol which is not persistent. It guarantees a message will be delivered without error as long as a connection is maintained. However, it is not persistent because after connection termination the message may be lost.

There are many protocols which provide long-term persistence. These include protocols which simply rebroadcast messages (utilizing a great amount of bandwidth) and more clever approaches such which rectify differences between remote message stores and only exchange messages which are in conflict.

In this thesis two persistent protocols are frequently utilized: Rumor Mongering [11] and Trickle [21]. Both of these protocols are fully introduced in Section 5.3

2.6 Service Discovery

Service Discovery protocols allow network devices to automatically identify other devices and services. One of the most commonly used is Dynamic Host Configuration Protocol (DHCP) [13], which manages devices on the network primarily by issuing and revoking IP addresses.

There are many other service discovery protocols which vary in functionality. Some are used to simply discover other devices and the features available on each, such as the Link Layer Discovery Protocol (LLDP) [34]. Other times, custom information must be exchanged, and more robust discovery protocols, such as zeroconf's [16] DNS-Based Service Discovery protocol [9].

This thesis uses INDI [25], a service discovery protocol built on top of UDP. It allows for both

device and service discovery, the details of which are described in Section 4.5.1.

2.7 Overlay Networks

Systems such as Spines [4] instantiate virtual routers on physical network nodes to alleviate the need for expensive routing protocols and application-specific networking models. Instead, applications utilize an interface similar to that of the Unix API.

The overlay dynamically establishes efficient routes and can coordinate complicated actions such as handoffs between different network partitions [5].

Such overlays do not, however, abstract different transport protocols from the application. For example, in Spines the `send` command is used for TCP and the `sendto` is used for UDP. Further, destinations are specified by an IP address, so publish-subscribe systems must have a defined IP address over which messages are sent.

2.8 Messaging Frameworks

AMQP [26] is a standardized protocol for message-based communication. Many implementations exist including RabbitMQ [35], ØMQ [18], and Apache Qpid [15]. Unlike Spines, it provides full abstraction of underlying networking to applications. For example, an application may queue a message to a specified destination marking it for guaranteed delivery; it does not specify an underlying protocol.

One drawback of this approach is an underlying transport protocol is assumed. Generally TCP is used regardless of messaging needs, as it provides all the *possible* guarantees (e.g. ordered, assured delivery) that an application may choose to stipulate. This can result in excess overhead when no guarantees are necessary.

2.9 Protocol Selection

Current methods of protocol selection generally fall into one of the following categories:

1. *Statically*: The developer chooses a protocol during development. This has been the primary method of creating networked applications for many years. When delivery requirements are

flexible or the network is relatively static, this may be a suitable approach.

In MANETs, due to their dynamic nature, this many times fails as links constantly change.

2. *Automatically*: The protocol is selected at the start of deployment, but static throughout. For example, an application could initially detect if the network is experiencing high traffic load, and opt for a less verbose protocol to minimize congestion.

   This rectifies the issues of deploying a given application on multiple static networks, but not dynamic networks.

3. *Dynamically*: The protocol changes throughout the entire deployment, and adjusts to changing network conditions.

   Currently this is primarily done at only higher-layers of the OSI stack. For example the protocol in [30] adapts the interaction of applications and some Remote Procedure Call (RPC) frameworks [27, 23] can allow endpoints to negotiate a serialization method for each call.

This thesis extends the dynamic method by applying dynamic selection to multiple layers of the network stack using both local and remote information. It also removes the need for global, or even local, consensus and can thus function in heterogeneous, dynamic networks.

Chapter 3: System Architecture

This chapter introduces an architecture for creating dynamic protocol selection frameworks by providing high-level descriptions for each of its functional components. Chapter 4 discuses the specific implementations for each component used for the remainder of the thesis.

3.1 Overview

The architecture for dynamic protocol selection is a process instance which meets the following requirements:

1. Provide a means for applications to interact with the system. This includes the ability to send messages to a specific destination and stipulate certain messaging requirements.

2. Adequately abstract lower-layer protocols from applications while still providing flexibility as available when using system sockets.

3. Dynamically monitor network state and assign labels to different aspects of the state. For example, label the network as "densely" or "sparsely" connected.

4. Select protocols based on user-defined metrics on at least one level of the network stack.

An "instance" of the proposed system is a service running on a distinct network node. Multiple instances of the system communicate over one or more networks to deliver messages sent to specific destinations.

3.2 Messaging Features

The core of the system is comprised of the messages sent between applications, possibly on different instances of the system. The system must define classes of message requirements each with a (possibly infinite) set of values. For example, a class of requirements may be *guaranteed delivery*, with the values being *true* and *false*. Another class may be the *persist time* for which the number of

values are infinite. Applications may have no preference for some classes of requirements, in which case the system shall define which value to assign the message.

The system must fall into one of three categories for accepting messages based on an application's requirement assertions:

- *Strict*: The message is only accepted if the system believes all requirements will be met. Otherwise, it must notify the application that the message is rejected.
- *Lax*: The message is accepted regardless of requirements and the system attempts to satisfy as many as possible. Optionally, the system may notify the application of which requirements were met.
- *Alternate*: The message is only accepted if the system believes all requirements will be met. If they cannot be satisfied, the system will offer the application alternative values for the classes which are not able to be met. The application may either accept other values, or abandon sending the message.

3.2.1 Guarantee

In the *strict* and *alternate* categories the system can never provide a true guarantee. For example, even if a message is "guaranteed" to be delivered, if the node fails arbitrarily or fragments from the network indefinitely, that guarantee may not be fulfilled.

As such, requirements which are accepted are truly only valid at the time of issuance, but the system must attempt to satisfy all requirements indefinitely.

3.2.2 Addressing

In addition to delivery requirements, messages must have an intended destination. Since this thesis focuses on group communications, it is not necessary to maintain a list of all remote endpoints (nodes). Doing so may be useful in some circumstances, however, as in systems which implement custom routing instead of utilizing an underlying protocol.

Instead, it is sufficient to mimic lower-level group communication methods such as multicast and link-local broadcast where messages are published to a set of subscribed nodes by specifying a single

address. This removes the complexity of maintaining node lists, but unless established beforehand, may require the use of Service Discovery as detailed in Section 3.4.1.

3.3 Required Components

This section describes the components necessary to meet the requirements of a dynamic protocol selection system.

3.3.1 Client Application Programming Interface

From application developers' perspective, one of the most important features of the system is a robust Application Programming Interface (API). It must provide at least the features:

1. *Initialization*: A means of applications connecting to the system. If implemented as a daemon, this may be via a socket or pipe, or if as a library, a function instantiating the system.

2. *Subscriptions*: As mentioned in Section 3.2.2 addressing shall be a publish-subscribe model. This requires a means of subscribing to and unsubscribing from a given destination.

3. *Requirement Assertion*: The application must be able to assert desired requirements, from those available allowed by the system, on a per-message basis.

4. *Publication*: A method to send a message to a specified destination given the asserted requirements.

5. *Feedback*: A method of the system sending control messages back to the application. For example, if a message could not be sent due to its requirements.

6. *Shutdown*: A graceful method of disconnecting a client from the system.

3.3.2 Network Monitor

A means of monitoring the underlying network is necessary as it influences protocol selection. This may be as simple as querying the Operating System for information, or may entail more complex processes such as coming to a consensus with other nodes, or merging local and remote information.

There are benefits and drawbacks to each approach. Using the Operating System information alone is low-latency, but generally only takes into account local information (e.g. the number of neighbors). Using consensus algorithms is beneficial when nodes come to an agreement and all instances use the same protocols. On the contrary, guaranteeing consensus in error-prone environments is impossible [14] and does not allow nodes to adapt quickly to network changes.

3.3.3 Protocol Manager

A Protocol Manager component manages the startup and shutdown of protocols. Each protocol may run as a single instance which multiplexes messages to and from multiple destinations, or each protocol-destination pair may be its own instance.

In the former case, it is feasible, albeit possibly unnecessary to start all protocols at system startup. A less computationally intensive process is to start protocols as they are needed. This is also the only way to start protocols in the latter case, as all destinations may not be known at system initialization.

3.4 Optional Components

3.4.1 Service Discovery

A service discovery mechanism may be necessary in some implementations of the architecture. For example, it may be necessary for each instance to advertise which protocols it has available so peers can choose from them properly to guarantee certain messaging requirements. This component is optional for implementations which need not start or shutdown protocols which are not in use, for example in systems that only allow a finite set of destinations and can maintain a small set of open sockets indefinitely.

3.4.2 Data Persistence

In some implementations data persistence is necessary. If any of the messaging requirements available to applications allow for long-term delivery guarantees, messages must be stored locally until delivery. In many cases, it is unknown if a message has been delivered to all its intended recipients and the message may be stored indefinitely or until some user-defined timeout occurs.

Two primary types of persistence can be used, possibly together, to allow for message recovery. Volatile storage is quicker to access if stored in local memory, but as the number of messages grows, resource contention may occur. Also, if the instance or host node fails, all messages would be lost.

To combat this, non-volatile persistence of messages on a reliable storage (such as a hard-drive) may be used. Both approaches can be used in conjunction where the volatile storage can act as a cache for the non-volatile up to some bounded memory usage.

Chapter 4: Approach & Implementation

The Dynamic Protocol Selection Middleware (DPSM) was written as an implementation matching the architecture in Chapter 3. It provides a generic JSON based API to applications which assert the requirements for each message.

DPSM runs as a Java daemon and accepts incoming TCP connections, generally from clients on the same host. Applications using DPSM connect to it and register as a client. Instead of utilizing native socket calls, all messages from the client are sent to DPSM along with a set of metadata including if the message should be:

- *Reliable*: Guarantees that the message, if received, is correct and free of errors. Also assures that the message is delivered at least to any destination in the local network fragment as long as the topology remains static.

- *Persistent*: Guarantees that the message will eventually be delivered to its destination. This essentially assures that the message is stored in non-volatile storage and will be received by the intended destination when it is available.

  Applications can also delete a persistent message by marking it as stale. Note that this does not free any storage space, but the message data will no longer be transmitted across the network.

- *Ordered*: Guarantees that the message is received after the last sent message from the originating host. It does not guarantee a total ordering of messages across hosts.

Unlike traditional sockets, which are used by only one application, DPSM allows multiple applications to share resources since it runs as a daemon. This is beneficial for a number of reasons.

First, applications with common destinations and delivery requirements transparently utilize the same underlying sockets. Also, persistent messages which must be stored in non-volatile storage are

stored in a common SQLite[1] database, eliminating the redundancy of in-application persistence.

Another benefit of this approach is its modularity. Since applications are only aware of delivery requirements and not the underlying delivery mechanisms, developers can insert new or differently parameterized protocols into DPSM without changing anything in applications.

Finally, the network observations made by DPSM are shared for protocol selection for all applications; there is no need for each application to monitor the network independently.

4.1 Problem Formalization

This section formally defines the problem space for dynamically selecting protocols at multiple layers of the network stack based on underlying network conditions.

Let $\mathcal{F} = \{F_1, \ldots, F_{|\mathcal{F}|}\}$ be a finite set of *network factors* (e.g. connectivity, bandwidth utilization) which affect protocol selection. Each $F \in \mathcal{F}$ is a finite set of labels specific to that factor. For example, if F is the *network connectivity*, labels such as "dense" and "sparse" may be appropriate.

Let $\mathcal{M} = \{(d_1, v_1), \ldots, (d_{|\mathcal{M}|}, v_{|\mathcal{M}|})\}$, where all $v_i \in \mathbb{R}$, be a finite set of measured network conditions. The values of each d_i are labels for their associated value v_i. Examples of elements include the amount of data transmitted over some bounded time interval, estimated number of neighbors, and rate of link changes. It is assumed that $\mathcal{M}$ always contains current and accurate information. In practice this may be difficult due to the unpredictability of the network and response time to changes.

Given a network stack of size $c \in \mathbb{Z}^+$, there are sets $\mathcal{P}_1, \ldots, \mathcal{P}_c$, where $\mathcal{P}_i$ represents the set of protocols available at the i^{th} layer of the stack. It is important to note that the starting index of one is only for convenience, and does not imply that the protocol layers being selected start at the top or bottom of the stack. Further, the value of c may be less than the Operating System's stack size if only a subset of layers are to be selected dynamically.

The protocols available are determined by which are installed on the system as well as program assertions. For example, if both unreliable and reliable protocols are installed at a given layer, but an application stipulates it requires reliability, the unreliable protocol may be eliminated from that

[1] `www.sqlite.org`

layer's selections.

This is not required, however, as protocols at other layers may be able to fulfill this requirement in different ways (e.g. a persistence protocol). This is left to the implementation to define.

Given this, the problem is two-fold:

1. Functions must be established to select the labels which best represent their associated network factor based on the measured network conditions. Formally, this is to create labelling functions $\{L_F \mid L_F : \mathcal{M} \rightarrow F \wedge F \in \mathcal{F}\}$. This process is explained in Section 4.2.2.

2. Functions which map the selected labels to protocols at each of the c layers of the network stack. Formally, create selection functions $\{S_i \mid S_i : F_1 \times \cdots \times F_{|\mathcal{F}|} \rightarrow \mathcal{P}_i \wedge 1 \leq i \leq c\}$. This process is explained in Chapter 7.

4.2 Protocol Selection Algorithm

4.2.1 Network Features and Conditions

The first task in dynamic selection is to associate with each network factor a set of labels. For the purposes of this thesis, two factors, each with three labels were used. The factors are *traffic load* which is a measure of the bytes per second across the local wireless medium, and *connectivity* which is the number of one-hop neighbors. Each factor can have the label *high*, *medium*, or *low*.

DPSM has built-in features to measure the network conditions ($\mathcal{M}$) of bandwidth utilization and neighbor count, measured by the Operating System and unique peer exchanges respectively (as described in Section 4.2.2). Further, the total available bandwidth is known *a priori* and the total number of network nodes is estimated by inspecting the unique number of message sources.

Based on these measurements, each condition is independently labeled as *low*, *medium*, or *high* based on Equation 4.1 where x is the current value of the condition and $mv(x)$ is the maximum

value of that condition.

$$T(x) = \begin{cases} low, & 0 < \frac{x}{mv(x)} \leq \frac{1}{3} \\ medium, & \frac{1}{3} < \frac{x}{mv(x)} \leq \frac{2}{3} \\ high, & \frac{2}{3} < \frac{x}{mv(x)} \end{cases} \tag{4.1}$$

For example, if the maximum bandwidth is 5 Mbps and currently 3 Mbps is in use, $T(3) = medium$. If the total estimated nodes on the network is 50 and 8 are estimated to be current neighbors, $T(8) = low$. The *high* label does not have an upper bound since the estimate of the maximum value may be incorrect, and a measurement greater than m may occur.

4.2.2 Labelling

Label Probabilities

The most important contribution of DPSM is its method of labeling based on network conditions, which leads to the creation of labelling functions L_F for all network factors $F \in \mathcal{F}$.

DPSM adopts an approach similar to that presented in [12], which uses Markov Random Fields (MRFs) to estimate the probability that a given label is correct for each $F \in \mathcal{F}$. Nodes occasionally communicate their calculated probabilities to neighbors which incorporate this new information into their estimates.

The goal is to find the label $l \in F$, for each $F \in \mathcal{F}$, which has the highest probability of occurring given the measured network conditions $\mathcal{M}$.

To do so, for each $F \in \mathcal{F}$, every node maintains the following information in addition to a global timeout value t:

1. A probability that each label, $P_F(l)$, is correct for that network factor.

2. A count, $C_F(l)$, of how many remote nodes believe the label for F is l. For a given F, $\sum_{l \in F} C_F(l) = \mathcal{B}_F$ which is the total number of unique neighbors which have transmitted a belief for F in the last period t. After a duration t, beliefs older than t are purged.

3. M_F, the value of v from one measurable network metric $(d, v) \in \mathcal{M}$.

4. A normal distribution function $\mathcal{N}_F^l : \mathbb{R} \to (0, 1)$ based on the mean and variance of previous values of v when $T(v) = l$ in the last period t.

The nodes occasionally transmit a message b which specifies the labels with maximum probability for each network factor. Specifically $b = \langle (F_1, l_1), \ldots, (F_{|\mathcal{F}|}, l_{|\mathcal{F}|}) \rangle$ where each l_i is the label with maximum probability for network factor $F_i \in \mathcal{F}$.

When neighbors receive b, the local C_F values for all $F \in \mathcal{F}$ are updated and the local probability for the corresponding labels are recalculated to incorporate this data using Equation 4.2.

The balance of the system favoring local or remote observations is determined by the constant $\lambda \in [0, 1]$. A high value gives more weight to remote observations than local, and a low value gives more weight to local observations.

$$P_{F_i}(l_i) = \frac{1}{2}\left(\underbrace{(1-\lambda)\mathcal{N}_{F_i}^{l_i}(M_{F_i})}_{\text{Local measurement}} + \underbrace{\lambda\frac{C_{F_i}(l_i)}{\mathcal{B}_{F_i}}}_{\text{New broadcasts}}\right) \tag{4.2}$$

A normal distribution is applied as observations may temporarily fluctuate and determining their probability based on historical data can smooth short-term variations. It is possible that some network factors will follow different distributions.

Labelling Function

For DPSM the labelling functions are the same for all layers of the stack, given by Equation 4.3 which simply selects the label with maximal probability.

$$L_{F_i} = \arg\max_{l \in F_i} \; P_{F_i}(l) \tag{4.3}$$

It is worth noting that DPSM allows some layers of the stack to include a "null" protocol. For example, the Session and Presentation layers may be null when using traditional UDP. This does not change the formal definition as the appropriate $\mathcal{P}_i$ sets can simply include the additional "null protocol".

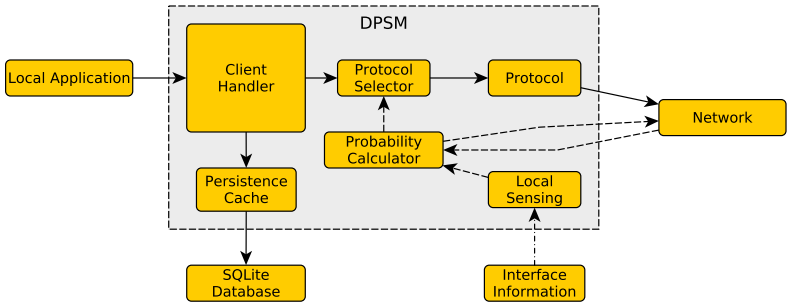


Figure 4.1: Information flow with DPSM. Solid lines indicate application messages and dashed indicate information for probability calculations.

4.3 Network Sensing and Exchange

DPSM can incorporate any measurable network factor into its $\mathcal{M}$ set. The basic implementation monitors the Operating System's network statistics to determine how much traffic is on the network and exchanges heartbeat messages to determine the number of one-hop neighbors. Additional network factors could be monitored and taken into account during protocol selection without modifying any applications.

Network state is exchanged using DPSM's own messaging API which stipulates that the state be sent in an unreliable, non-persistent, non-ordered manner.

4.4 Architecture

Figure 4.1 shows a high-level overview of the DPSM architecture. Solid lines indicate application messages and dashed indicate local and sensed information used for protocol selection.

Messages originate at the application and enter DPSM with a JSON message which stipulates:

1. The delivery requirements (reliability, persistence, and ordered).

2. Destination for the message as a string. This string is hashed into an IP address for actual use on the network.

3. The message contents the application wants to transmit.

Upon receiving this, the Client Handler passes this information to both the Persistence Cache

and Protocol Selector. If the message is to be persisted, the Persistence Cache writes it to the non-volatile SQLite database and stores it in memory for faster retrieval.

The Protocol Selector queries the Probability Calculator which returns the current state of the network. This information and the message requirements are used to select a protocol or set of protocols to use for message delivery. The selector then creates a packet containing the:

1. The delivery requirements,
2. The destination string,
3. The source DPSM instance ID and local source application ID,
4. The application message.

The destination string is hashed using MD5 into a specified range of IPv4 addresses (e.g. 10.0.0.1 – 10.0.0.254) and sent to that address with the chosen protocol(s).

4.5 Protocol Maintenance

For the purposes of this implementation, it is assumed that all DPSM instances have the same set of available protocols. However, the middleware does not start all protocols at runtime as some may utilize a large amount of local resources or send messages unnecessarily until an application actually utilizes it.

For example, Trickle [21] sends messages every few moments querying for missed information. Until a persistent message is sent somewhere on the network, this is not useful. NORM [2] sends a number of messages to maintain an estimate of group membership and establishes buffers for incoming information. This is wasteful until reliable transport is needed.

As such, each DPSM instance must know when to start a protocol instance. There are two times this occurs: when the local Protocol Selector determines the protocol is needed or when a remote instance begins using that protocol. The first is trivial to implement, but the latter requires some method of service discovery.

4.5.1 Service Discovery

DPSM utilizes INDI [25] to determine the protocols and destinations other DPSM instances are utilizing. INDI provides robust service discovery methods built on top of mDNS [10] and allows DPSM instances to enable protocols for previously-unknown destinations. Messages indicating what services are available are called *advertisements*.

It has three possible modes of discovery:

1. *Proactive*: Services in use are broadcast at some interval to all neighbors.

2. *Reactive*: Instances occasionally query their neighbors to determine which services are available. Responses are sent via unicast.

3. *Opportunistic Caching*: Advertisements are cached within INDI and presented to the application only when queried.

For the purposes of DPSM, the Proactive method was chosen. Although it has the possibility of being the most verbose method, Reactive and Opportunistic Caching do not provide the necessary functionality. Reactive could be used, but since unicast is utilized, it breaks the paradigm of group communications and also incurs the overhead of maintaining a list of network nodes.

Every time an advertisement is received, the DPSM instance enables any protocols locally disabled that are listed as active in the advertisement. Some protocols may have multiple instances running and are advertised separately. For example, there may be multiple UDP sockets open for different addresses.

Each running protocol instance also maintains a timer. If the protocol is not used by a local application or advertised by a remote DPSM instance when the timer expires, it is shutdown to reduce overhead.

Chapter 5: Static Protocol Analysis

This chapter presents experiments that will determine the dynamic selection policies for DPSM in Section 7. Multiple combinations of protocols are statically selected at the beginning of each experiment in a variety of controlled group-oriented scenarios. Throughout each scenario, the Message Delivery Ratio (MDR), messaging overhead, and delivery delay is measured as a function of neighbor count for each node. Further, the traffic load on the network is varied and its impact on protocol performance is analyzed.

This information is then used in Section 7.1 to establish a qualitative mapping from network state to the best protocol in each situation which is applied in DPSM to dynamically select protocols.

5.1 Emulation Environment

The CORE [3] network emulator was used in all experiments to simulate MANETs.

CORE is a network emulation tool, which allows unmodified applications to be tested in a variety of network configurations. It was chosen over other network emulators, such as EMANE [20] due to its simplicity, and over other network simulators such as NS2 [7] and NS3 [17] as it allows all software to run on a live network, without needing changes or incurring time distortion.

CORE's primary radio model, the module that emulates wireless links, is "basic range." This means that any two nodes are either connected or entirely disconnected based on some distance threshold. Other models exist where connectedness is not a boolean value and provides a much more substantial set of parameters. However, for this thesis, basic range provides the necessary functionality without overcomplicating parameterization.

5.2 Mobility

This thesis focuses on group communications, so the mobility of nodes should mimic that of groups in different situations. The Reference Point Group Mobility (RPGM) pattern [8] was chosen due

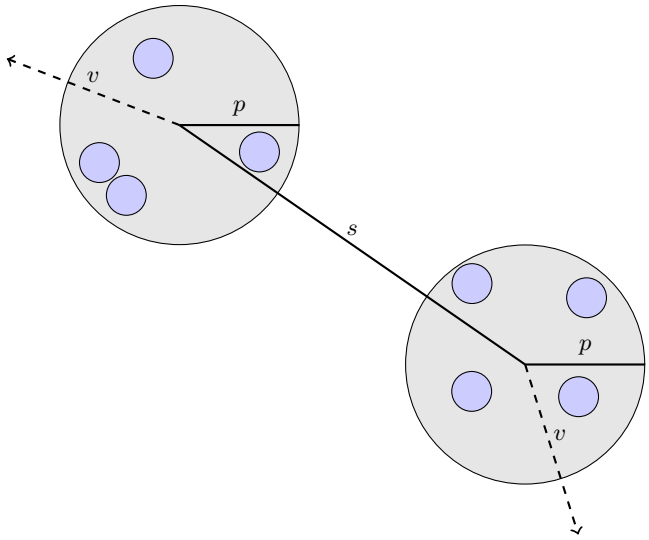


Figure 5.1: Example of the RPGM mobility model with eight nodes and two groups. s is the reference point spacing, p is the intra-group spread, and v are the velocity vectors.

to its wide acceptance for accurately modelling group movement and the availability of a reference implementation[1].

The primary parameters to RPGM are a number of groups n, a number of nodes m, reference point spacing s, intra-group spread p, and group speed v. RPGM creates n reference points which move randomly at speed v while maintaining a distance no less than roughly s between each pair.

The m nodes are divided into n roughly equal groups, each of which is associated with one of the reference points. As the reference point moves, the nodes associated with it move randomly while maintaining a distance roughly less than p from it. Figure 5.1 shows an static example of RPGM with two groups and eight nodes.

5.3 Protocols

The Session and Transport layers of the OSI stack were varied in each experiment. The former provides persistence in all experiments. As discussed in Section 2.5, persistence is a critical component to many distributed group communication systems; it provides long-term delivery capabilities that lower-layer protocols are too short-lived to handle, and the upper layer, the application, cannot or desires not to manage. The latter provides reliability guarantees when necessary.

[1] `toilers.mines.edu/Public/Code/mobilemodels.html`

5.3.1 Session Layer

Trickle [21] and Rumor Mongering [11] are the two protocols tested at the Session layer. In addition, when composing a Session and Transport layer, a "null" protocol may be used for the Session layer. This essentially means there is no messaging persistence, and the Session layer is simply a pass-through between the Transport and Application layers.

Trickle is a protocol for propagating data, which is possibly versioned, across a network in a manner known as "polite gossip." Nodes periodically broadcast a summary, sometimes known as a *manifest*, of their local information, but remain silent if they hear information identical to theirs. If a node hears a manifest with information which has missing or outdated information, it broadcasts an update to reconcile the differences. Trickle also has a backoff feature which automatically limits the send-rate, reducing unnecessary broadcasts.

Rumor Mongering in this context will be treated as the name of a protocol; however, it is truly a class of protocols introduced in [11] wherein every node treats new information as a "hot rumor" and randomly selects other nodes to tell. Further, every message generally has a maximum number of transmission before it is no longer considered "hot" and the transmissions slow or stop.

Since Rumor Mongering was defined, it has been used to identify a more broad scope of protocols, including those which utilize manifests, as Trickle does.

The implementation for this thesis falls into this more general category. Periodically, nodes broadcast a manifest of all messages they have received[2]. If a receiver hears of a new message (a "hot gossip"), it will be added to its next manifest broadcast and marked as "requested" which will induce other nodes to broadcast the reconciliation data.

5.3.2 Transport Layer

At the Transport layer, UDP [28] and NORM [2] are used.

UDP is a well know, unreliable, datagram protocol which is one of the oldest and most widely used in networking. Messages are generally sent either as unicast to another specific node, multicast to a specific group of nodes, or broadcast. In the latter two, a Time-To-Live (TTL) value stipulates

[2]The manifest is actually fragmented as to remain below the Maximum Transmission Unit (MTU).

Table 5.1: CORE configuration constants for all experiments.

Parameter	Value
Scenario Dimensions	1000 × 1000 meters
Range	20 meters
Bandwidth	5 Mbps
Delay	20 ± 5 milliseconds
Jitter	0 milliseconds

the maximum number of hops a packet may be routed to a destination before being dropped. If using broadcast with a TTL of one, it is considered link-local broadcast.

In this thesis all messages routed are sent using link-local broadcast. This minimizes the implementation and analysis complexity, as every packet goes at most one-hop, and is frequently used in deployments where routing is infeasible.

NORM, NACK Oriented Reliable Multicast, is a robust and highly configurable multicast protocol which provides various levels of delivery guarantees depending on configuration. It uses *NACK*s, Negative ACKnowledgements, to indicate to senders that a message was lost. Many techniques may be used to reconcile missed information including simple rebroadcast and more advanced Forward Error Correction (FEC) codes. Finally, NORM may act in a datagram or stream mode.

To limit complexity, the NORM configuration for all experiments use datagrams, since these best compare to UDP, implement rebroadcasting as well as FEC codes, and otherwise use default parameters.

5.4 Experimental Procedure and Parameters

5.4.1 CORE and RPGM Configuration

All experiments in this chapter utilize the same set of scenario and mobility parameters, with the exception of the number of RPGM groups. The CORE configuration for all scenarios is shown in Table 5.1 and the RPGM parameters are shown in Table 5.2.

5.4.2 Traffic Load

The traffic generator MGEN [19] was used to simulate the applications utilizing each protocol being tested. MEGN was configured to send messages of various sizes and at various intervals to simulate

Table 5.2: RPGM parameters for static scenarios.

Parameter	Value
Number of Nodes	50
Number of Groups	1, 2, 5, 10
Reference Point Separation	100 meters
Node Separation	25 meters
Total Time	5 minutes
Speed	2 ± 5 meters per second
Pause Time	2 ± 2 seconds

Table 5.3: Message frequency and size for various traffic loads used in the static protocol comparison.

Load	Frequency	Size
High	1 seconds	1024 ± 50 bytes
Medium	5 seconds	512 ± 20 bytes
Low	10 seconds	256 ± 10 bytes

low, medium, and high traffic loads. Table 5.3 shows both the message frequency and size for each load level.

All messages are addressed for delivery to all nodes on the network. Therefore, Message Delivery Percentage is measured as the percentage of total network nodes, of which there are 50, that receive a given message. Messages are given the requirements of both reliable and persistent, as this is the most taxing both on DPSM's selection and the network due to overhead.

5.4.3 Group Size and Repetition

As indicated in Table 5.2, the number of groups is varied between 1, 2, 5, and 10. This is to allow for a variety of connectivity levels.

In total, sixty experiments were run, five for each group count in each of the three traffic loads. The data for these experiments are classified by traffic-level, each of which was averaged over all four of the group counts.

5.5 Performance Metrics

These experiments must answer the following questions:

- *What percentage of messages are delivered with each combination of protocols?*

- *What is the delivery latency with each combination in various connectivity and traffic levels?*
- *What is the average total amount of data that must be sent to deliver one message?*

To do so, three metrics are used. *Message delivery percentage* (sometimes referred to as Message Delivery Ratio, or MDR) indicates the percentage of messages which reach their in destination. Since all messages are to be destined for all applications, 100% delivery means a message was delivered to all applications. *Overhead percentage* is the percentage of additional information that must be sent to deliver a message. This includes header information, control messages, and other data transmitted by the selected protocols. Finally, *latency* is the average amount of time it takes for a message to be delivered to a single destination.

All of these are measured as a function of *sender connectivity*. This is the number of neighbors the sending node has, as measured by the simulation. This gives an estimate of how connected the sending node is to the rest of the network.

5.6 Experimental Results

This section provides results showing how well various protocols, selected statically at the start of an experiment, perform at the Session and Transport layers. This information from is used to both determine optimal protocol selection policies for the dynamic approach, and for comparison thereof.

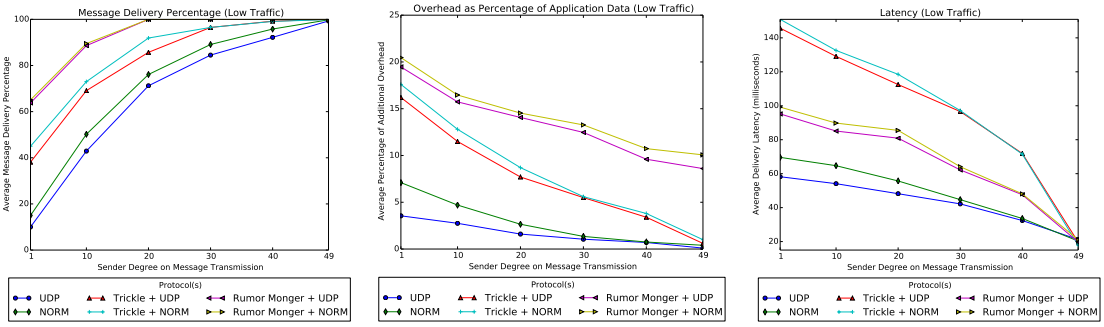
For the Session layer, which provides persistence in these experiments, three protocols are available. Trickle, Rumor Mongering, and a "null" protocol. As described in Section 5.3, Trickle transmits only a small amount of information which gives a summary of all messages available in its datastore, and is quiet when other instances are communicating. The latter ignores other transmissions and broadcasts more verbose information at a fixed rate. Further, there is the option of not using a Session layer (e.g. a "null" protocol), adding a third choice to the layer.

The Transport layer has two possible options. Traditional UDP link-local broadcast, which is unreliable, or NORM, a reliable multicast protocol. Since a Transport layer is required, the "null" protocol is *not* available.

5.7 Results

5.7.1 Low Traffic

Figures 5.2(a), 5.2(b), and 5.2(c) show the message delivery percentage, average additional overhead, and latency for all protocols in networks under low traffic.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 5.2: Performance of statically selected protocols in low-traffic networks.

In general, protocols that deliver more messages also require more overhead. Specifically, Rumor Mongering, regardless of the underlying Transport layer, delivers the most messages, but also requires the transmission of much more data. Because Rumor Mongering does not stifle manifest broadcasts like Trickle, even in fully connected networks, there is some overhead utilization.

Further, protocols which do not attempt retransmission, such as UDP, have fast delivery. This is because a message which is sent will be received by nodes connected to the sender quickly or not at all. Longer-lived protocols like Rumor Mongering and Trickle may take longer as they must reattempt delivery if at first it fails.

Because there is minimal message loss, NORM does not increase the MDR of any protocol more than few percent, while increasing latency and overhead utilization.

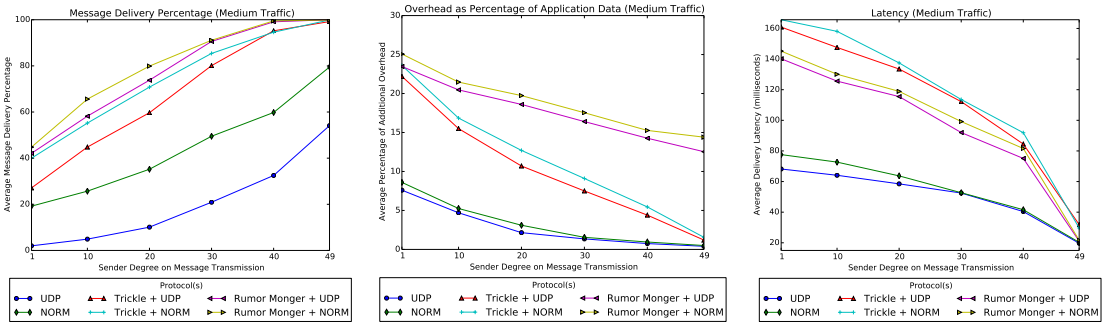
5.7.2 Medium Traffic

Figures 5.3(a), 5.3(b), and 5.3(c) show metrics for medium-traffic scenarios. Compared to low-traffic, two primary trends stand out.

First, the MDR of a given protocol increases significantly in some cases when NORM is used

as the transport protocol. Most drastically, Trickle's MDR increases by nearly 15% in the lowest connectivity scenarios when utilizing NORM. Further, this increase in MDR appears to cause minimal extra overhead as compared to the low-traffic scenarios. This suggests that NORM has some minimum amount of overhead it utilizes, which only affects MDR with a specific amount of loss.

The second primary trend is that the MDR of Trickle (especially with NORM) approaches that of Rumor Mongering. This is likely because Rumor Mongering causes increased load on the network, further congesting it, while Trickle is a relatively quiet protocol.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

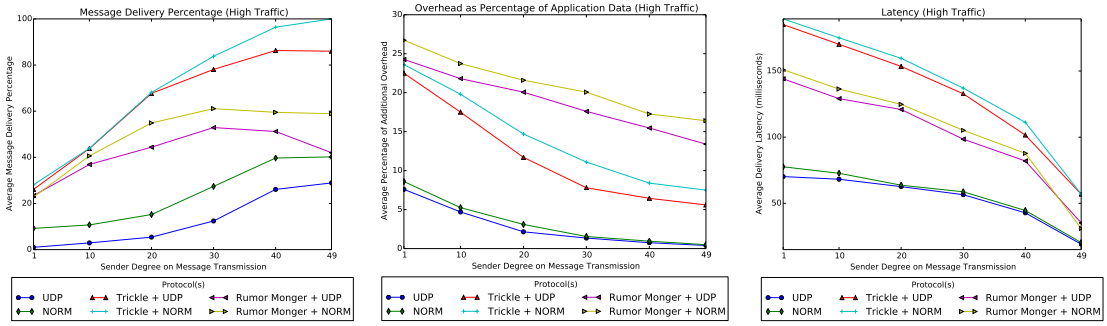
Figure 5.3: Performance of statically selected protocols in medium-traffic networks.

5.7.3 High Traffic

The last set of experiments compares the protocols in networks under high traffic. Figures 5.4(a), 5.4(b), and 5.4(c) show the MDR, overhead, and latency of each set of protocols.

This set of experiments show very different results than low- and medium-traffic, specifically the performance of Rumor Mongering. In the previous experiments, all protocols continued to deliver more messages as the network became more dense. However, due to its verbosity, Rumor Mongering in high-traffic, high-connectivity scenarios causes the MDR to *decrease*. Figure 5.4(a) shows a peak MDR for Rumor Mongering at 30 neighbors, for both UDP and NORM transports.

Additionally, NORM and UDP alone send similar amounts of traffic and have similar latencies to the medium-traffic scenarios, albeit delivering fewer messages.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 5.4: Performance of statically selected protocols in high-traffic networks.

5.8 Analysis

From the baseline experiments, a few conclusions can be made:

1. In low- and medium-traffic scenarios, as the connectivity of the network increases Trickle and Rumor Mongering, regardless of underlying transport, deliver approximately the same percentage of messages.

2. In high-traffic scenarios, Rumor Mongering fails to perform as well as Trickle in terms of both MDR and overhead.

3. Trickle takes longer to deliver messages than Rumor Mongering at all traffic levels.

4. UDP and NORM deliver drastically fewer messages in all scenarios than when utilizing a persistence protocol at the Session layer.

Chapter 6: State Estimation Analysis

This set of experiments determines how accurately DPSM estimates the local network state based on the approach in Section 4.2. Specifically, they answer the questions:

- *How close is the estimated level of connectivity to the actual connectivity level?*
- *How close is the estimate of traffic load to the actual traffic load?*
- *Given the estimates, does DPSM accurately classify the network?*

Fifty nodes were placed in emulation using the RPGM model. The parameters of the RPGM model were then varied. The number of groups was set to 2, 4, 5, 10, or 25, and the reference point separation was set to 150, 250, or 800 meters. Since RPGM has a random factor to it, namely the starting positions and reference point movement, five different mobility instances of each pattern were generated. Thus, every data point is the average of five separate trials.

6.1 Connectedness Estimate

DPSM monitored the network and the connectedness it measured versus the actual connectedness of the local fragment is compared in Figure 6.1. This shows that as the number of groups increases, the accuracy of the measurement increases, but is always under 16% (or 8 of the 50 nodes) error. Also, as the reference point separation increases the error decreases.

6.2 Traffic Estimate

During the emulation, all nodes broadcast packets every second between the size of 128 and 512 bytes using the MGEN [19] traffic generator . DPSM measures this, and estimates the amount of local total traffic. The performance of this estimation is shown in Figure 6.2.

Similar to the connectedness error, as the number of groups and reference point separation increases, the error decreases. For both network factors, these trends are due to the fact that the network is becoming more sparse, and local measurements are a better judge of the network state.

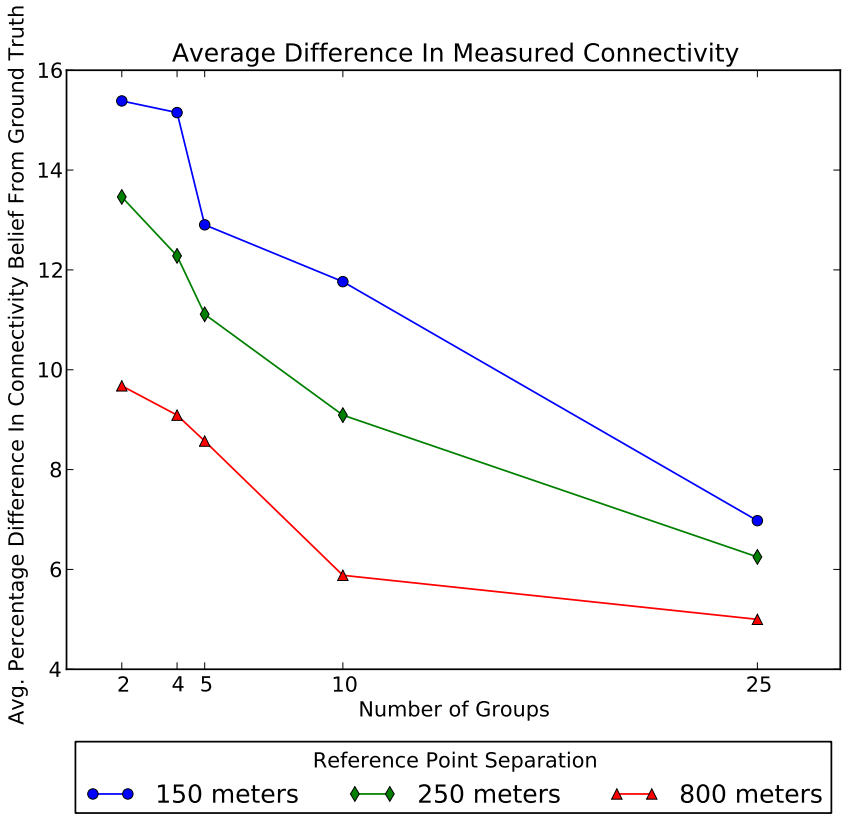


Figure 6.1: Average difference between using DPSM to estimate connectedness and ground truth.

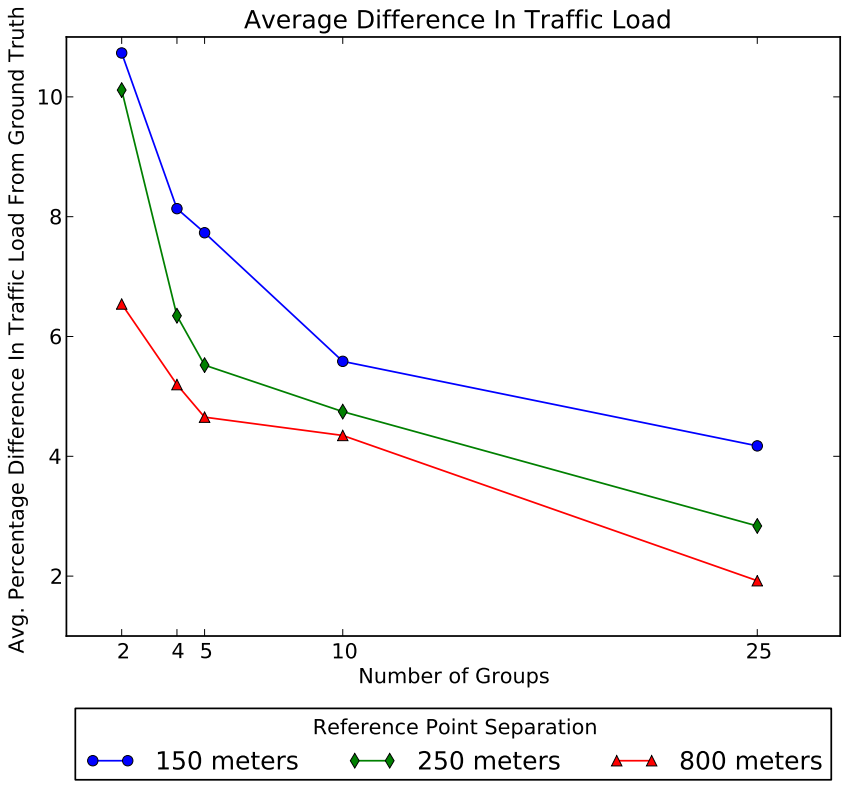


Figure 6.2: Average difference between using DPSM to estimate local traffic load and ground truth.

6.3 Classification

This section presents results indicating how well DPSM classifies the local network. Recall the network is to be classified in terms of connectedness and traffic. For each, there are three labels, *high*, *medium*, and *low*.

For connectedness, low represents 33% or less of the network being connected to a node, medium as between 33% and 66%, and high as over 66%.

Similarly, for traffic load, low represents 33% or less of the 5 Mbps available bandwidth being used, medium as between 33% and 66%, and high as more than 66% bandwidth utilization.

Figure 6.3 shows the percentage of time each connectivity classification was selected versus the actual number of connected nodes, and Figure 6.4 shows the percentage of time each bandwidth classification was selected versus the total bandwidth usage in the neighborhood (e.g. one-hop neighbors).

The dotted lines in each plot show the threshold values for which the state classification should change (33% and 66% of the maximum network factor value). Both plots show nearly the same trend: that DPSM classifies networks qualitatively well, and that the assigned classification nearly matches the underlying network state.

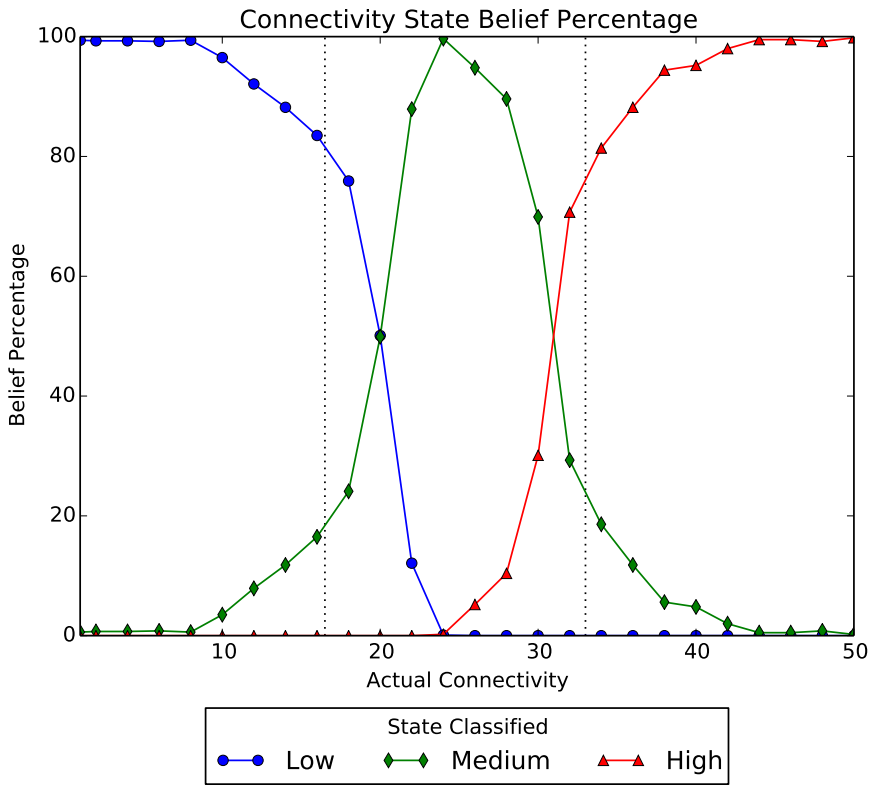


Figure 6.3: Percentage of time each connectedness classification was applied versus the actual number of connected nodes.

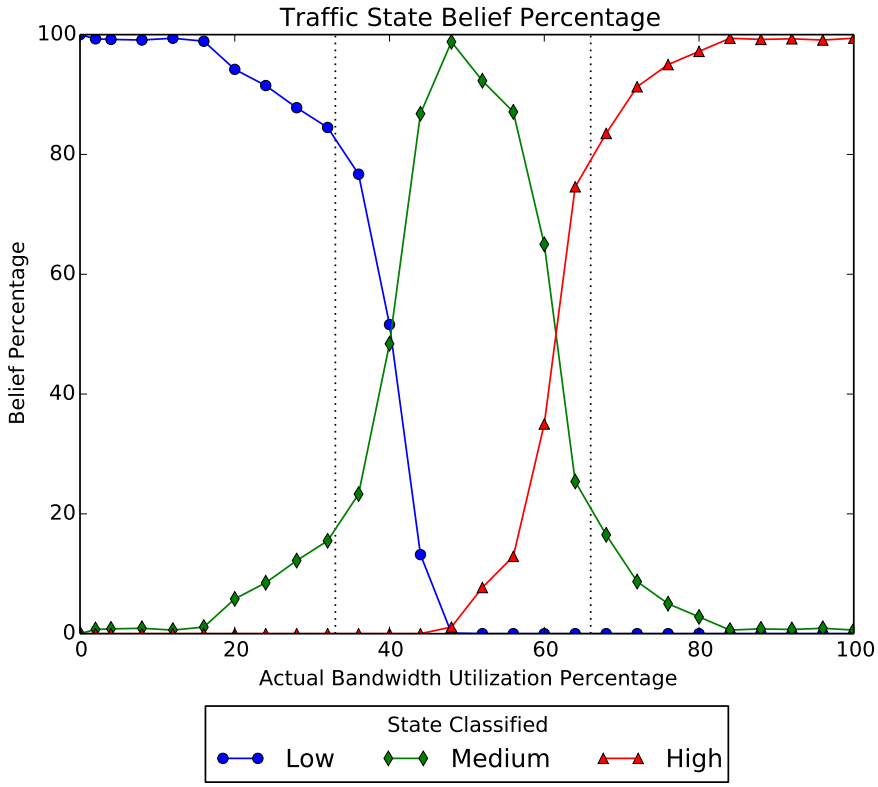


Figure 6.4: Percentage of time each traffic classification was applied versus the total bandwidth utilization percentage in the neighborhood.

Chapter 7: Dynamic Comparison

In this chapter, dynamic protocol selection is compared to the static selection of protocols. First, the mapping of sensed network state to protocols is established. Then, the dynamic and static protocol selection approaches are applied to a variety of MANETs and compared on the basis of MDR, overhead, and latency.

7.1 Mapping State to Protocols

Using the results from Section 5.6, one can select the "best" protocols for each combination of network connectivity and traffic load. What defines "best" is subjective, and deployment-specific. For these experiments, the protocols are selected qualitatively, focusing on maximizing MDR while sending a reasonable amount of data. This can be thought of as an approximation of maximizing the number of messages delivered per byte sent.

Table 7.1 shows the mapping of network state, as determined by DPSM, to protocol which will be used in further experiments. Note "RM" stands for Rumor Mongering and "TR" stands for Trickle.

7.2 Static Comparison

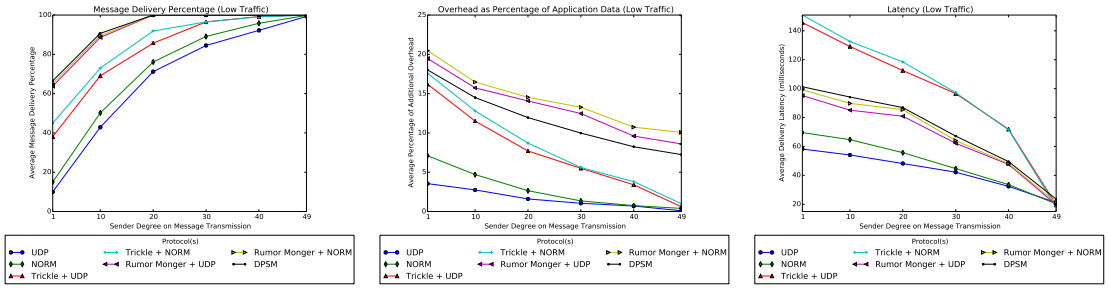
In the first set of experiments, DPSM was run in the same scenarios as Section 5.4 and compared on the basis of message delivery, overhead, and latency.

Table 7.1: Protocols selected for every network state, as measured by DPSM.

		Connectivity		
		Low	*Medium*	*High*
Traffic	*Low*	RM + UDP	TR + NORM	TR + UDP
	Medium	RM + NORM	TR + NORM	TR + UDP
	High	TR + UDP	TR + NORM	TR + NORM

7.2.1 Low Traffic

Figures 7.1(a), 7.1(b), and 7.1(c) show DPSM as compared to static protocol selection in low-traffic scenarios.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

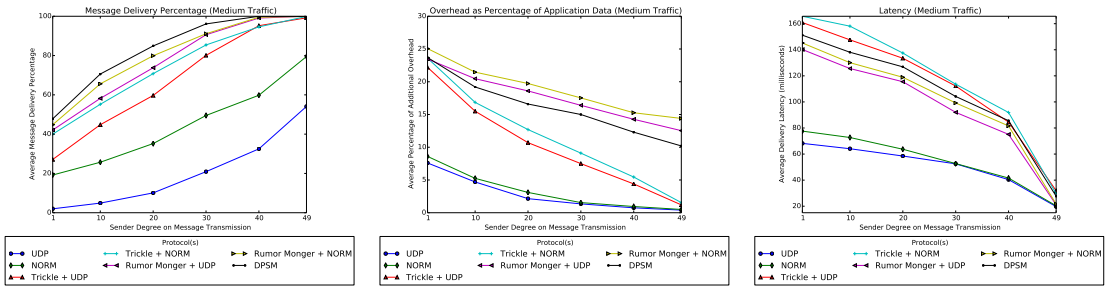
Figure 7.1: Performance of DPSM and statically selected protocols in low-traffic networks.

For all sender connectivity levels, DPSM delivered at least as many messages as the best static protocol, Rumor Mongering with NORM, taking only marginally longer. It also utilized less overhead to deliver each message than Rumor Mongering regardless of underlying transport. This indicates that in some cases Trickle was utilized due to a higher connectivity level in the network.

Overall, DPSM performs well in the low-traffic scenarios incurring a 1-3% latency penalty to deliver marginally more messages with less overhead.

7.2.2 Medium Traffic

Figures 7.2(a), 7.2(b), and 7.2(c) show DPSM as compared to static protocol selection in medium-traffic scenarios.



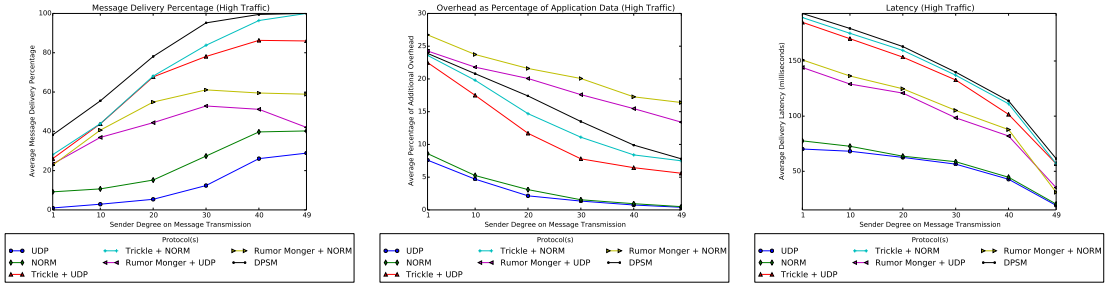
(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 7.2: Performance of DPSM and statically selected protocols in medium-traffic networks.

The difference in MDR for DPSM compared to the best statically selected protocol is more drastic when the traffic load is increased. For most connectivity levels, DPSM delivered 5-10% more messages while utilizing an average amount of overhead. Further, messages were delivered only a few milliseconds slower than Rumor Mongering with NORM, the protocol delivering the second-most messages.

7.2.3 High Traffic

Figures 7.3(a), 7.3(b), and 7.3(c) show DPSM as compared to static protocol selection in high-traffic scenarios.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 7.3: Performance of DPSM and statically selected protocols in high-traffic networks.

When the network has a high-traffic load, DPSM drastically outperforms the static protocol in terms of message delivery. This is because it may select more verbose protocols which utilize more bandwidth but deliver more messages in periods of relatively low traffic. This does, however, incur

an increase in both overhead and latency. Relative to the increase in message delivery, however, these increases are small.

7.3 Comparative Scenarios

The second set of experiments compare DPSM to static protocols in real-world scenarios. Unlike previous experiments which use local connectivity as a comparison basis, these experiments use scenario-specific features.

7.3.1 Message Ferry

When two or more groups of nodes remain out of communication range for a long time, message "ferries" are sometimes used. A small number of nodes can visit each group, gathering messages to deliver to the other groups.

To simulate this, 48 of the 50 nodes are split evenly into four groups in a square, each group out of communication range of the others. The nodes in each group are free to roam randomly within two-communication ranges of the group's original center point, while still remaining fragmented from the other groups.

The two remaining nodes, starting at groups diagonal from each other, constantly move in a circle, "ferrying" messages between sequential groups. Figure 7.4 shows this configuration.

For these experiments, measuring effectiveness as a function of local connectivity is not necessarily useful. Since the four groups of nodes generally remain connected but never connect with other groups, the number of neighbors for a given node rarely varies far from 25%. As such, the effectiveness was compared only to the traffic load as shown in Figures 7.5(a), 7.5(b), and 7.5(c).

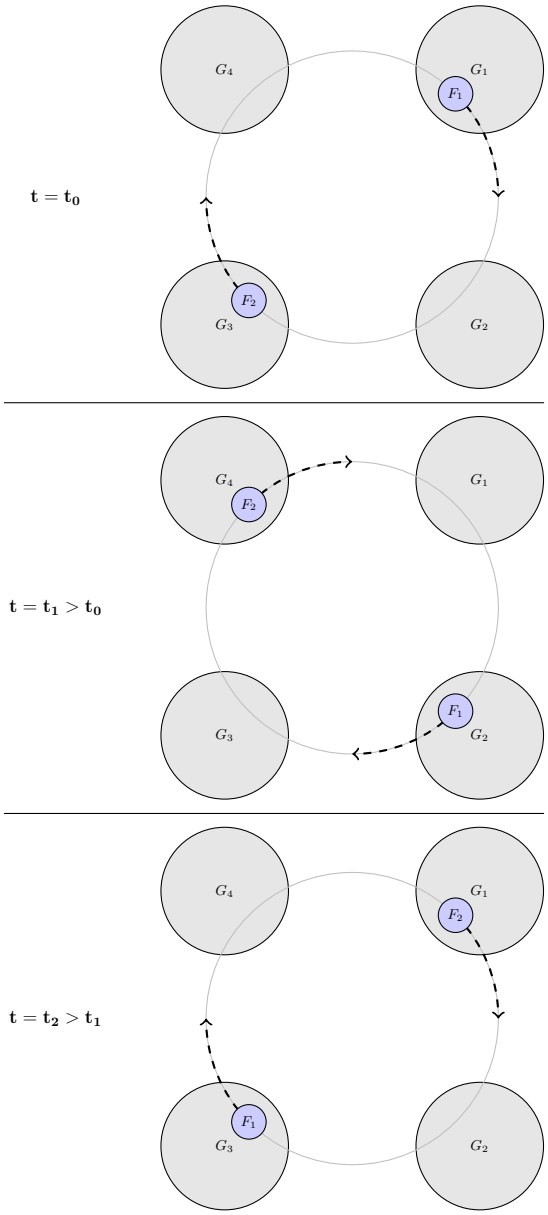
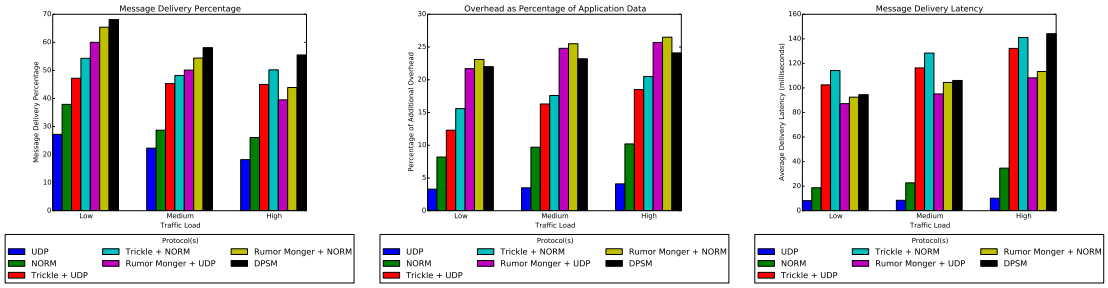


Figure 7.4: Example message ferrying scenario. The two ferries, represented as F_1 and F_2 constantly circle the four groups, G_1, G_2, G_3, and G_4, delivering messages from other groups.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 7.5: Performance of DPSM and statically selected protocols in message ferrying scenarios with various traffic load.

DPSM always delivers at least as many messages as the best statically selected protocol. As the traffic load increases, DPSM delivers increasingly more messages than any other protocol while sending less data. For this, there is a latency penalty, in the worst case, of about 1% over the static protocol delivering the most messages.

7.3.2 Group Following

Groups advancing sequentially from waypoint to waypoint is a common military-domain mobility scenario. For example, advancing groups of soldiers to a location far away may be achieved by breaking the trip into smaller movements. It is assumed that each group knows the location of the preceding group from a pre-determined plan or via communication.

To simulate this, 50 nodes are divided into five equally sized groups, starting at approximately the same location. At the start, the first group moves to a random location and then stops. Once stopped, the second group moves to the same location. Upon reaching the first group, the first group moves to another position. All other groups follow in suit.

Figure 7.6 shows an example. Each cross represents the waypoint for the groups as determined by the first group. A group reaching a given waypoint is indicated by shading it the associated color.

The same data was collected as the message ferry in Section 7.3.1 as shown in Figures 7.7(a), 7.7(b), and 7.7(c).

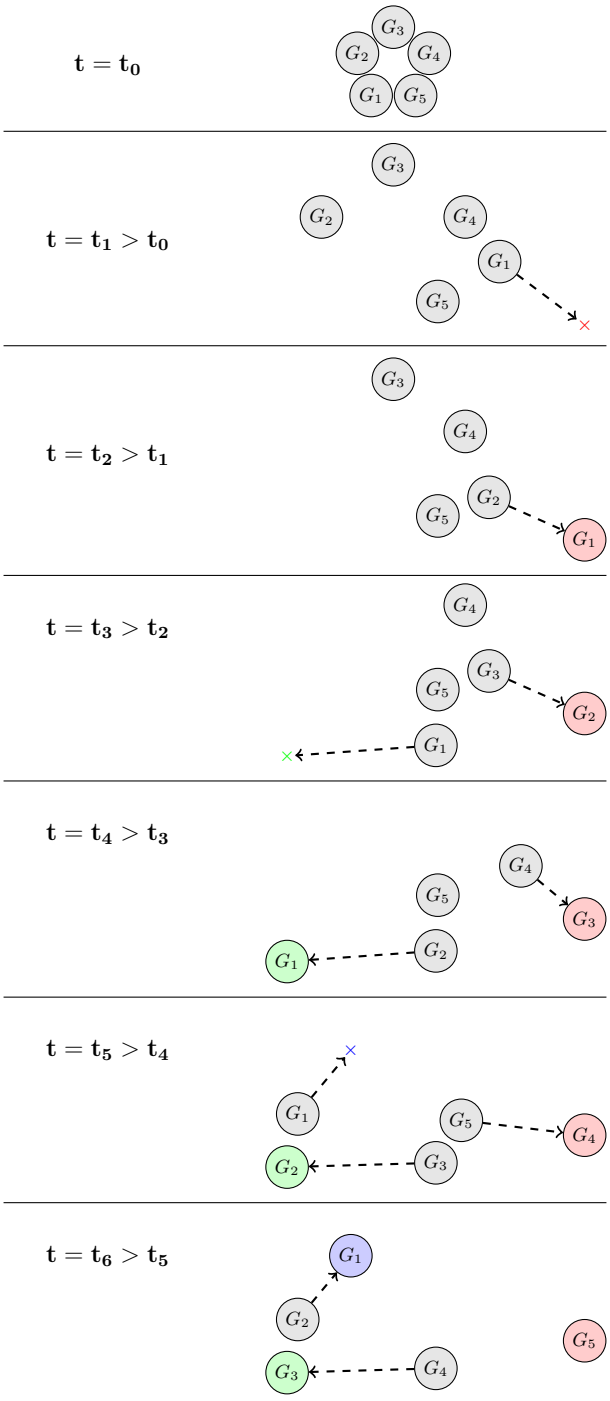
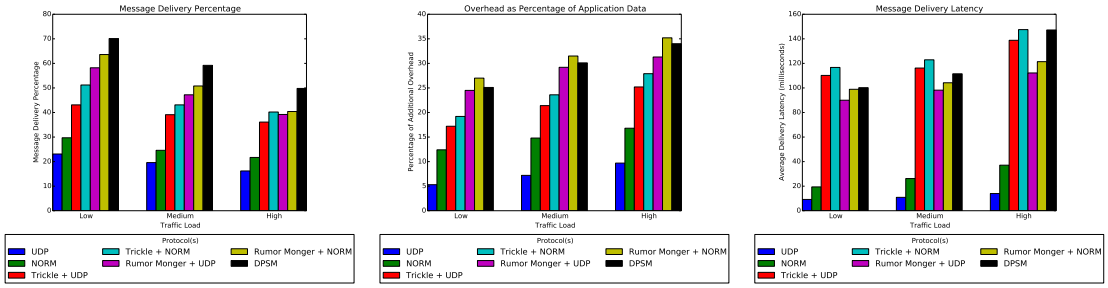


Figure 7.6: Example of group following progression. Each cross represents the waypoint for the groups as determined by the first group. A group reaching a given waypoint is indicated by shading it the associated color.



(a) Delivery Percentage (b) Overhead Percentage (c) Delivery Latency

Figure 7.7: Performance of DPSM and statically selected protocols in group following scenarios with various traffic load.

The trends for this scenario are quite similar to those of Section 7.3.1. For all traffic loads, DPSM delivered more messages than any static protocol, while sending an amount of data similar to the other protocols. Unlike the message ferry scenarios, however, DPSM also consistently delivered messages faster than Trickle.

Chapter 8: Conclusions

8.1 Contributions

This thesis has introduced a method of dynamically selecting group communication protocols for use in MANETs. The following specific contributions were made:

1. A formal definition of the problem space for protocol selection.
2. A formal approach to solving the protocol selection problem.
3. A generic architecture for systems to automatically select protocols and abstract their details from applications.
4. The implementation of a dynamic protocol selection middleware, DPSM, which increases the effectiveness of messaging in MANETs.
5. Quantitative results indicating dynamic protocol selection delivers at least as many messages as static protocols in various environments.

Chapter 1 introduced the problems with static protocol selection in dynamic networks, specifically that as the network changes, the initial assumptions may not hold and protocols' expected performance will change.

In Chapter 2, a general overview of MANETs was provided and existing approaches to dynamic selection were introduced. These existing approaches were shown to have limited applicability to general communication systems as they are scoped only to one deployment, or only function at a single layer of the network stack.

A generic architecture to generalize dynamic selection was introduced in Chapter 3 which identifies the necessary functional components. Additionally, it introduced topics such as addressing and how implementations should handle application requests. An example implementation using this architecture was detailed in Chapter 4.

8.2 Effectiveness of Approach

The results gathered and presented in Chapter 7 show that dynamic selection is an effective method to eliminate some of the problems with choosing application protocols prior to deployment. In all scenarios, DPSM delivers at least as many messages at the best statically selected protocol, while sending less data.

In low- and medium-traffic scenarios, DPSM delivered more messages than of any protocol to which it was compared. It sent more data than Trickle but less than Rumor Mongering. Similarly, its delivery latency was generally between these two other protocols, indicating that DPSM oscillated between the two protocols as the network changed. When message delivery is the primary metric to maximize, DPSM appears to be the best choice. Further, although there are some overhead and latency trade-offs, DPSM seems to be a good compromise for applications which must balance message delivery, overhead, and latency.

In high-traffic scenarios, DPSM requires additional time to deliver messages which can be partially attributed to delay incurred by switching protocols. In these scenarios DPSM took 3% longer to deliver messages than the protocol with the next highest message delivery ratio. Nonetheless, this small increase in latency allows for an average increase in message delivery of approximately 10% for high traffic scenarios. Therefore, in high-traffic scenarios, DPSM again seems to be the best choice unless a slight increase in latency cannot be tolerated.

In the real-world scenarios of Section 7.3, DPSM shows very similar trends — as traffic load increases DPSM begins to deliver significantly more messages than other protocols at a small latency expense.

Interestingly, dynamically switching between protocols allowed for a message delivery ratio higher than statically selecting any single protocol. This is due to DPSM using the highly-verbose Rumor Mongering protocol in areas of the network that are not experiencing high traffic, and the less verbose Trickle protocol in others, alleviating congestion.

Overall, these results demonstrate that selecting protocols a priori for dynamic networks is difficult — as the network changes, each protocol performs differently. Automatically and dynamically

selecting protocols at runtime alleviates the burden on developers, and allows communication to continue through unforeseen changes of network topology. DPSM may incur small increases in overhead and latency, but for most group communication applications it would likely perform better than any statically assigned protocol.

8.3 Future Work

There are a number of areas which need additional work. Foremost is the selection algorithm presented in Section 4.2. This is only one of many ways the network can be labelled based on measurable information. There are other approaches to fusing local and remote information, especially in the wireless sensor community [24, 36, 22, 6]. These and other approaches could be used on a deployment-specific basis based on the needs of the applications.

Further, there are assumptions in the presented selection algorithm that may not hold in all networks, namely knowing the total number of nodes and total available bandwidth. Both of these values may be unknown or change over time. Additional research is needed to determine how a replacement function to Equation 4.1, could be created without having bounds on these values.

Another useful topic to investigate is the protocol selection of nodes in the fringe between two disparate portions of the network. For example, many nodes experience extended periods of high-connectivity and low-connectivity. There are some nodes, though, that may be in between a high-density and low-density network region. This could potentially cause that node to quickly oscillate between multiple protocols, resulting in unknown behavior. Investigating first if this is indeed a possibility, and creating a solution to smooth the transition between protocol changes is an important area to research.

Finally, an extremely important area of research which could drastically improve this approach is adding the ability of learning to the middleware. This thesis required extensive background experimentation as documented in Chapter 5 to determine which protocol was most effective in various environments. This could prove a labor-intensive process if done with more protocols, at more layers of the stack, or in the real-world rather than emulation. The ability for a middleware to learn which protocol is best would be an invaluable and extremely challenging undertaking.

Bibliography

[1] *MANETs in Military Communications - Strategic Insights and the Road Ahead.* Frost & Sullivan, 1 edition, September 2009.

[2] B. Adamson, C. Bormann, M. Handley, and J. Macker. NACK-Oriented Reliable Multicast (NORM) Transport Protocol. RFC 5740 (Proposed Standard), November 2009.

[3] Jeff Ahrenholz, Claudiu Danilov, Thomas R. Henderson, and Jae H. Kim. CORE: A real-time network emulator. In *IEEE Military Communications Conference*, pages 1–7, 2008.

[4] Yair Amir, Claudiu Danilov, Raluca Musuăloiu-Elefteri, and Nilo Rivera. The spines overlay network. `http://www.spines.org`, 2003.

[5] Yair Amir, Claudiu Danilov, Raluca Musuăloiu-Elefteri, and Nilo Rivera. The smesh wireless mesh network. *ACM Trans. Comput. Syst.*, 28(3):6:1–6:49, September 2008.

[6] T. Banerjee, K. Chowdhury, and D.P. Agrawal. Tree based data aggregation in sensor networks using polynomial regression. In *Information Fusion, 2005 8th International Conference on*, volume 2, pages 8 pp.–, July 2005.

[7] L. Breslau, D. Estrin, K. Fall, Sally Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Ya Xu, and H. Yu. Advances in network simulation. *Computer*, 33(5):59–67, May 2000.

[8] Tracy Camp, Jeff Boleng, and Vanessa Davies. A survey of mobility models for ad hoc network research. *Wireless Communications and Mobile Computing: Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications*, 2(5):483–502, 2002.

[9] S. Cheshire and M. Krochmal. DNS-Based Service Discovery. RFC 6763 (Proposed Standard), February 2013.

[10] S. Cheshire and M. Krochmal. Multicast DNS. RFC 6762 (Proposed Standard), February 2013.

[11] Alan Demers, Dan Greene, Carl Houser, Wes Irish, John Larson, Scott Shenker, Howard Sturgis, Dan Swinehart, and Doug Terry. Epidemic algorithms for replicated database maintenance. *SIGOPS Oper. Syst. Rev.*, 22(1):8–32, January 1988.

[12] Linda E. Doyle, Anil C. Kokaram, Senan J. Doyle, and Timothy K. Forde. Ad hoc networking, markov random fields, and decision making. *IEEE Signal Processing Magazine*, 23(5):63–73, 2006.

[13] R. Droms. Dynamic Host Configuration Protocol. RFC 2131 (Draft Standard), March 1997. Updated by RFCs 3396, 4361, 5494, 6842.

[14] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, April 1985.

[15] The Apache Software Foundation. Qpid. `http://qpid.apache.org`, 2005.

[16] IETF Zeroconf Working Group. Zero configuration networking. http://www.zeroconf.org/.

[17] Thomas R. Henderson, Sumit Roy, Sally Floyd, and George F. Riley. Ns-3 project goals. In *Proceeding from the 2006 Workshop on Ns-2: The IP Network Simulator*, WNS2 '06, New York, NY, USA, 2006. ACM.

[18] iMatix Corporation. ØMQ. `http://zeromq.org`, 2007.

[19] Naval Research Laboratory. Multi-generator (MGEN). `http://www.nrl.navy.mil/itd/ncs/products/mgen`, 2005.

[20] Naval Research Laboratory. Extendable mobile ad-hoc network emulator (EMANE). `http://www.nrl.navy.mil/itd/ncs/products/emane`, 2009.

[21] Philip Levis, Neil Patel, David Culler, and Scott Shenker. Trickle: A self-regulating algorithm for code propagation and maintenance in wireless sensor networks. In *In Proceedings of the*

First USENIX/ACM Symposium on Networked Systems Design and Implementation (NSDI), pages 15–28, 2004.

[22] Tsung-Han Lin and Polly Huang. Sensor data aggregation for resource inventory applications. In *Wireless Communications and Networking Conference, 2005 IEEE*, volume 4, pages 2369–2374 Vol. 4, March 2005.

[23] Arskom Ltd. Spyne. http://spyne.io/, 2013.

[24] Hong Luo, Huixiang Tao, Huadong Ma, and S.K. Das. Data fusion with desired reliability in wireless sensor networks. *Parallel and Distributed Systems, IEEE Transactions on*, 22(3):501–513, March 2011.

[25] J. Macker and I. Taylor. Indi: Adapting the multicast dns service discovery infrastructure in mobile wireless networks. In *MILITARY COMMUNICATIONS CONFERENCE, 2011 - MILCOM 2011*, pages 1616–1621, 2011.

[26] OASIS. AMQP: Advanced message queueing protocol. `http://amqp.org`, 2003.

[27] Oracle. Remote method invocation. http://www.oracle.com/technetwork/java/javase/tech/index-jsp-136424.html, 2011.

[28] J. Postel. User Datagram Protocol. RFC 768 (INTERNET STANDARD), August 1980.

[29] J. Postel. Transmission Control Protocol. RFC 793 (INTERNET STANDARD), September 1981. Updated by RFCs 1122, 3168, 6093, 6528.

[30] J.G. Quenum, S. Aknine, O. Shehory, and S. Honiden. Dynamic protocol selection in open and heterogeneous systems. In *Intelligent Agent Technology*, pages 333 –341, December 2006.

[31] Aaron M. Rosenfeld, Robert N. Lass, Dustin S. Ingram, William C. Regli, and Joseph P. Macker. A comparison of group-based data persistence techniques in manets. In *Military Communications Conference*, 2012.

[32] R. Sivakami and G.M.K. Nawaz. Reliable communication for manets in military through identification and removal of byzantine faults. In *Electronics Computer Technology (ICECT), 2011 3rd International Conference on*, volume 5, pages 377–381, April 2011.

[33] R. Sivakami and G.M.K. Nawaz. Secured communication for manets in military. In *Computer, Communication and Electrical Technology (ICCCET), 2011 International Conference on*, pages 146–151, March 2011.

[34] IEEE Computer Society. Station and media access control connectivity discovery. `http://standards.ieee.org/getieee802/download/802.1AB-2009.pdf`, 2009.

[35] Pivotal Software. RabbitMQ. `http://rabbitmq.com`, 2013.

[36] W. Yuan, S.V. Krishnamurthy, and S.K. Tripathi. Synchronization of multiple levels of data fusion in wireless sensor networks. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 1, pages 221–225 Vol.1, Dec 2003.