Cross-Layer Discovery and Routing in Mobile Ad-hoc Networks

by

Meenakshi Venkataraman, B.E.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science in Engineering

THE UNIVERSITY OF TEXAS AT AUSTIN

May 2006

Cross-Layer Discovery and Routing in Mobile Ad-hoc Networks

APPROVED BY

SUPERVISING COMMITTEE:

Christine L. Julien, Supervisor

Scott Nettles

For Amma

Acknowledgments

Amma and Appa, without your constant encouragement this thesis would not have found its way to completion. Expressing my gratitude in words would be putting bounds on it. I hope I did you proud in finishing this piece of work. Thanks for being patient with me through this entire process, Vasan.

Christine, thank you for giving me an opportunity to explore what it feels like to conceive of an idea, dwell on it, live it for an extended period of time, and to take it to completion. Thanks also for making me feel free to talk to you any time I faced a problem. Your patience and liveliness have helped me on many an occasion. My sincere thanks to Scott, for consenting to read and critique this work.

If there is one person who gave me the courage to embark on a research project in an area I was not very familiar with, it is Sean. I am forever indebted to you for the long hours you spent helping me, for your constant motivation, cheer, support and companionship. Geethu, in many ways you have shown me that there are things in life beyond work, and that they are worth it too. Thanks for showing me the other side. Bala, special thanks for being there, just within cell phone's reach. Karen, Krithika, Ramya, Naresh, Arun Balaji, Lakshmi, and all of my family — thanks for bearing with me through all the irregular correspondence. Austinites — Pullur Vasan, Smrits, Sow, Karthik, Amit, Vishwas, Alex, Raquel, Shailesh, Brian, and Shweta — many thanks for making this an enjoyable and memorable experience.

A million thanks to all my labmates in the MPC group — Adam, Raghavan, Chang, Taesoo, Sanem, and Seth, without whose cooperation, I would not have been able to work on this thesis from Fort Worth. Adam, I owe you a huge bunch of thanks (and beverages ;)) for taking my call at 3:00 AM in the morning on a Sunday.

My thanks would not be complete without mentioning my colleagues at work. Ron, thanks for being a flexible and patient supervisor. Sumit, Ujjwal, Nathan, Aparna, Anne, Hong, Hari, Harsha, Chin-Wei, Po-Yu, and the rest of you — thanks for making me feel at home at Spirent from the day I started.

Cross-Layer Discovery and Routing in Mobile Ad-hoc Networks

Meenakshi Venkataraman, M.S.E.
The University of Texas at Austin, 2006

Supervisor: Christine L. Julien

Communication in ad hoc networks traditionally relies on network addresses known *a priori*. This work addresses the need for application-aware adaptive communication that creates network routes based on applications' dynamic resource requests. We motivate this need by examining the state of the art in mobile ad hoc network communication, the requirements of applications, and the impact of existing protocols on flexibility and efficiency. We introduce an intuitive generalization to source routing which facilitates discovery of a resource in an ad hoc network and the creation and maintenance of a route from the requesting host to the discovered destination. We thus eliminate the requirement that existing routing protocols be coupled with a name or resource resolution protocol, instead favoring an entirely reactive approach to accommodate significant degrees of mobility and uncertainty. We present an initial implementation, a performance evaluation, and a comparison to existing alternatives.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Mobile ad hoc networks are created when mobile devices communicate directly without using an infrastructure. Applications for such networks are common when an infrastructure is unavailable (e.g., in disaster recovery situations when the infrastructure has been destroyed) or unusable (e.g., in military applications where the infrastructure belongs to the enemy). Mobile ad hoc networks form opportunistically and change rapidly in response to the movement of the connected devices, or mobile hosts. Such an environment presents a network topology that is both dynamic and unpredictable, as nodes may join and leave the network at any time. In traditional multi-hop networks that are infrastructure driven, packets from a source to a destination are transported through designated nodes called "routers". This is made possible in such networks due to the relatively fixed network topology. However, this model of communication is ineffective in mobile ad hoc networks because of their dynamic nature. Hence, all nodes need to have the capability to route packets, as it is unknown *a priori* which nodes may need to serve as routers. In addition, because hosts may be constantly moving, their interactions are inherently transient.

While mobile devices are often connected to the Internet via wireless access points, at times they are completely disconnected from any wired infrastructure. In addition, it may be beneficial to some applications if they

elected to use only local interactions even when a connection to the Internet is available. Such scenarios abound in a wide variety of application domains. In military scenarios, troops and their vehicles are becoming increasingly capable of both sophisticated data collection and dynamic wireless communication. In the field, a soldier may wish to locate mapping information, mine locations, or other data collected by his fellow soldiers. First responder applications require people with differing tasks, e.g., emergency medical technicians (EMTs), firemen, policemen, search and rescue officers, etc., to converge on a confined area and perform concurrent tasks. They collect information about the site (e.g., hot spots, smoke density, location of survivors, etc.) and benefit from accessing data collected by others' devices. Construction sites are becoming increasingly intelligent as they contain a variety of sensors that provide information about the state of equipment, supplies, workers, etc. A super on the site may access information based on the site's quadrants or his immediate task. Cars that communicate sophisticated information to drivers have been made possible due to advances in user interfaces, paving the way for applications that coordinate automobiles to share weather, traffic, or mapping data, or even to exchange generic files on the roadway.

We characterize these applications and their needs according to the following generalization:

- *Dynamic network topology* - In mobile ad hoc networks link breakages are common. These breakages may be due to node failure, node motion or channel effects like multipath fading.

- *Data rich environment* - Nodes in certain networks are capable of sensing their environments. A large amount of user data is generated in other

networks. Such networks generate large amounts of dynamic and varying data. For example, sensors on a highway may be able to monitor atmospheric temperature and pressure (i.e., generate weather data); as well as measure traffic load.

- *Dynamic data generation* - The information available at any host is dynamic and unpredictable, i.e., hosts create and delete data according to their own processing, regardless of other devices.

- *Varying application needs* - Different devices and users require different data according to their instantaneous tasks and environments. In the highway example above, while one node may be interested in learning weather information in the vicinity, another node may want to know traffic conditions further along the highway in order to alter its route if necessary.

1.1 Communication in Mobile Ad Hoc Networks

Much work on supporting applications in mobile ad hoc networks builds on routing protocols that maintain communication between senders and receivers. As the topology of the network changes, these protocols adjust routes to maintain end-to-end connectivity. This style of interaction requires significant *a priori* knowledge to be shared among the mobile hosts. That is, a host must know in advance the unique addresses and resource capabilities of the other hosts in the network with which it desires to communicate. This assumes the existence of well-known and available servers that cache resource availability. A host wishing to communicate with another host must first contact the server to resolve the host's name, following which the node must additionally

employ a routing algorithm to discover and maintain a communication path to the desired destination. Figure 1.1 illustrates this *two phase* approach. In this figure (and the following figures), the hatched node (labeled **A**) is the source node (or requester). The node with the heavy border (labeled **B**) is the destination that can provide the requested data or service. The gray node (labeled **L**) provides the lookup service. Solid black arrows indicate requests, dashed arrows represent replies, and double-lined arrows indicate service registrations. The table beside node **L** indicates the resource information cached in the server's registry.

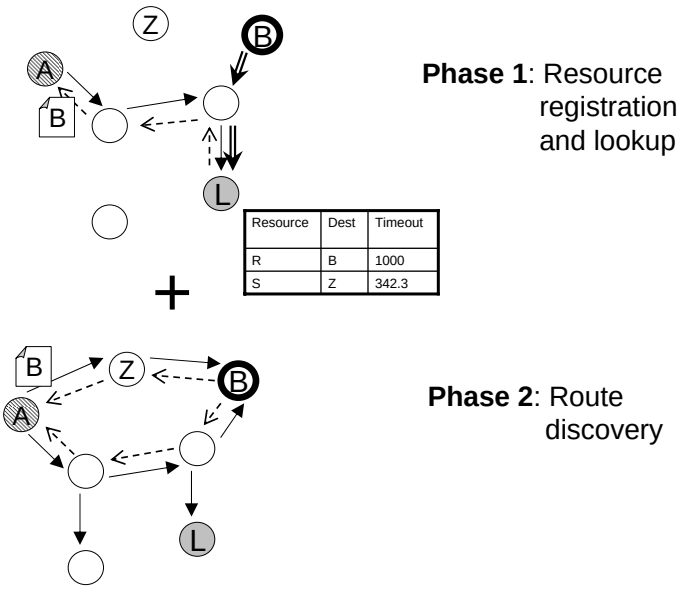

Figure 1.1: The two phase approach for communication in mobile ad hoc networks.

These two phase protocols are motivated by the desire to support the end-to-end model of communication common in Internet applications instead of emerging mobile applications like those characterized earlier. Such an ap-

proach, though appropriate in the wired Internet; has several drawbacks in mobile ad hoc networks, such as:

- Electing and maintaining a stable server set in an ad hoc network incurs a significant overhead in the highly dynamic scenarios targeted in this work [33]. The overhead incurred is primarily due to the reconfigurable nature of mobile ad hoc networks. In wired networks, such elections are less frequent, as these networks have a relatively static topology.

- The cost of advertisement in data rich environments becomes prohibitive as the number and variance of data sources increases.

- When the resources (or data) are highly dynamic, maintaining an accurate and consistent registry of resources requires significant volumes of control messages [18].

- In a purely ad hoc network, the servers may themselves be mobile and dynamic. In such a scenario, a node entering the network would require an initial discovery protocol targeted at finding the resolvers.

1.2 Motivation for Combined Discovery and Routing

In our evaluation of existing communication mechanisms and our examination of the needs of applications in mobile ad hoc networks, we identified a mismatch between the provisions of existing protocols and the needs of emerging applications. Specifically, to successfully utilize mobile ad hoc routing, a mechanism is required that resolves names, intentions, or service descriptions on behalf of the application.

Applications in mobile networks do not desire to contact each other based on unique addresses or even based on simple names. Instead, applications have intentions and goals, and they desire to find coordinating partners that satisfy them. For example, in a first responders application, triage workers may tag the injured with tiny vital sign sensors [19] capable of emitting signals regarding the patient's condition. An EMT's portable device (e.g., PDA or tablet PC) could search the network for vital sign data with certain properties indicating the criticality of the injury or the change in a patient's condition. Examining such query scenarios for a variety of application domains highlights the need for a communication protocol that adapts to varying query types, possibly even expressed in different resource description languages.

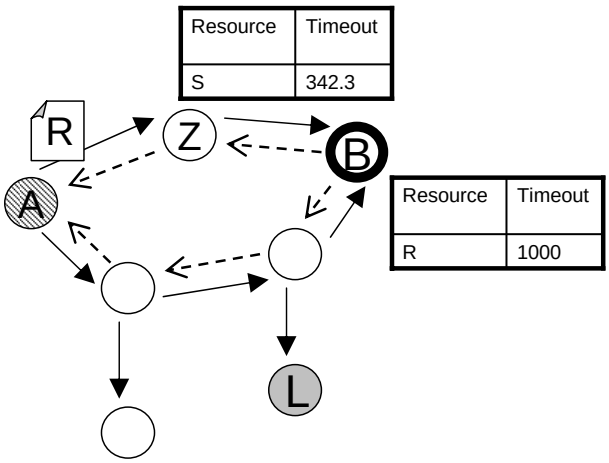

Figure 1.2: Resource discovery using a single phase approach.

In mobile ad hoc networks, network packet overhead and message delivery latency are of the utmost concern because interactions are opportunistic and hosts must take advantage of communication partners while they are

connected. This is the principal motivation for avoiding an approach which requires several phases of communication over the network. A single phase communication paradigm is proposed in this thesis, as shown in Figure1.2. The figure illustrates the network traffic generated by a combined discovery and routing protocol. Nodes now query the network with an "intention", instead of destination addresses, as is the case with conventional routing protocols. Each node maintains an internal registry of the resources it can provide. In the discovery phase, a source node queries for a particular resource directly. For example, **A** broadcasts a query for resource **R**, and because it can provide **R**, node **B** replies to the query as it can provide **R**, eliminating the second phase of communication.

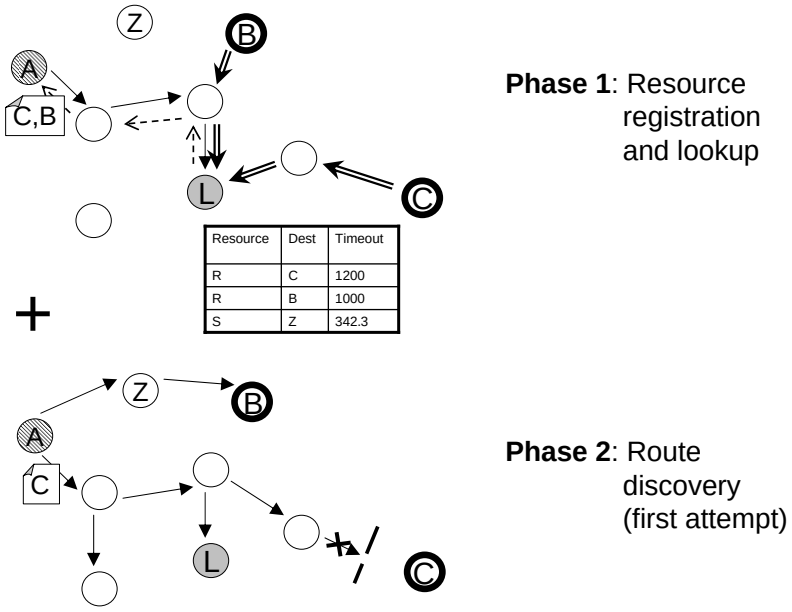

Figure 1.3: Impacts of stale registrations.

Service discovery approaches, though promising from an abstract per-

spective, require a lookup phase followed by a communication phase. In highly dynamic networks, these multi-phased interactions are more likely to cause failures as the services discovered may not actually be available when communication commences, e.g., as shown in Figure 1.3. The figure shows just one example of the negative impact of stale registrations. During the resource lookup phase, the lookup server **L** returns both nodes **C** and **B** as potential providers (listing **C** as preferred for some notion of "better"). Node **A** cannot tell the difference between the two, so during the route discovery phase, it attempts to contact **C**, but **C** may have disappeared from the network. Node **A** must wait for the route discovery attempt to time out before attempting to contact **B**. Some delay sensitive applications may have a low tolerance for such behaviour.

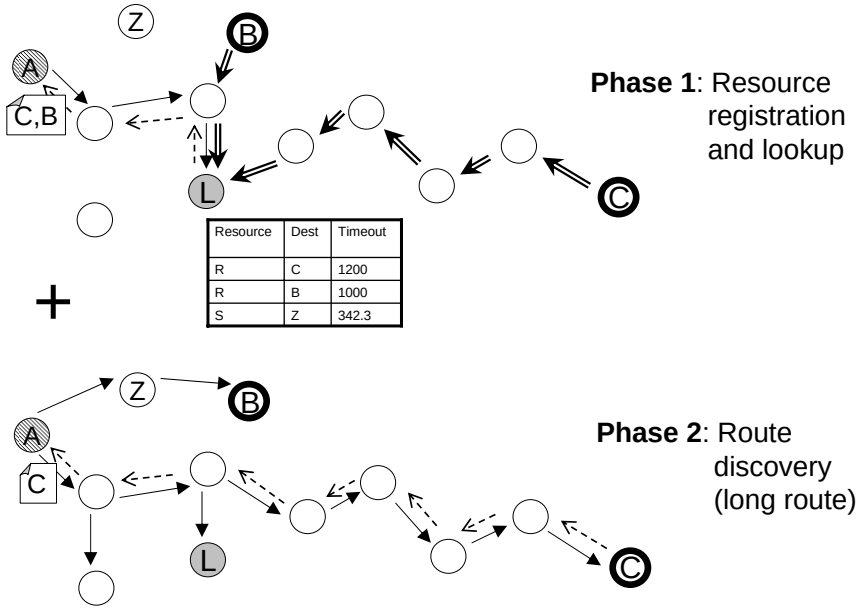

Figure 1.4: Illustrating route length effects of using the two phase approach.

Furthermore, in dynamic networks, the two phase approach cannot guarantee that the destination selected by the lookup service will be closest to the source for some application defined measure of "distance". Figure 1.4 illustrates this scenario. As in Figure 1.3, say the lookup service **L** returns both nodes **C** and **B** as providers of the service (listing **C** as the preferred resource). **A** attempts to contact **C**, but **C** may be further away from **A** as compared to **B**. Longer routes result in larger message delivery latencies and an increased probability of route breakage. In order for the protocol to provide more relevant information, the lookup server needs to maintain additional network topology information about resources that it has cached. The single phase cross layer approach on the other hand does not require such additional caching in its lookup server. By virtue of its design, this approach eliminates the need for a lookup server in the network. However, with the single phase approach, nodes need to cache information about resources they can themselves provide. The effects of the differences in protocol design will be discussed further in Chapter 4.

The approach taken in this work concerns itself with the performance implications of our design decisions. Chapter 3 provides the details of a protocol that combines resource resolution and routing into a single step. This protocol employs a route discovery mechanism that functions *without* the source host having to know the unique address of the destination. Instead, route discovery is based solely on properties of the destination. To achieve this behaviour, we introduce a level of indirection into a source routing protocol (e.g., Dynamic Source Routing (DSR) [30]). Source routing was selected as a foundation because performance comparisons of DSR (a source routing protocol) and AODV (a distance vector routing protocol) have shown that, although

the distance vector protocol achieves better performance on application level metrics like delay and throughput, the source routing protocol achieves a lower overhead in highly dynamic situations like those that pervade our target applications [6, 12]. In addition to its description, a formal behavioural characterization using I/O Automata [35] is included to provide a clear description of the protocol.

1.3 Thesis contribution

This thesis introduces a reactive communication protocol that removes the drawbacks of name resolution, as applied to mobile ad hoc networks. The principal goal of this work is not to provide a highly-tuned protocol but to evaluate the feasibility and desirability of incorporating application-level information directly into a routing protocol. In Chapter 4, this baseline protocol is evaluated with respect to several routing metrics and compared with existing alternatives. It is our position that an application-centered approach is essential to supporting real-world applications and therefore the cost of such an approach must be made acceptable. While others have proposed similar mechanisms, their protocols make assumptions that limit the solutions' flexibility and expressiveness [18, 23]. The novel contributions of this work are as follows.

- Identification of a set of assumptions made by existing mobile ad hoc network communication mechanisms that are limiting to the protocols' applicability to real-world applications.
- Development of a protocol for communication that overcomes these assumptions.

- A formal model of the protocol's behaviour (using I/O Automata).
- Performance evaluation of the new protocol that serves to not only compare our protocol to alternatives but to demonstrate the feasibility, advantages, and disadvantages of incorporating non-fixed length addressing into a reactive mobile ad hoc routing protocol.

1.4 Outline of the thesis

This thesis is organized as follows. Chapter 2 evaluates existing communication solutions in mobile ad hoc networks. A wide range of protocols have been explored and contrasted in relation to our work in this chapter. In Chapter 3, the combined discovery and routing protocol is described in detail. This chapter also provides a formal description of the protocol, validating the protocol design. Chapter 4 analyzes protocol performance through simulation and compares it to existing alternatives. Chapter 5 takes a critical look at the expressiveness and flexibility of our protocol and discusses potential enhancements. Finally, Chapter 6 provides a brief conclusion of this work.

Chapter 2

Related Work

Chapter 1 gave a brief overview of applications in mobile ad hoc networks, and the motivation for the work done in this thesis. In this chapter, we take a brief glimpse at the presently existing approaches to making communication possible in mobile ad hoc networks.

2.1 Routing in Mobile Ad Hoc Networks

Routing protocols for mobile ad hoc networks can generally be divided into two categories: *proactive* and *reactive*. Proactive protocols [8, 9, 27, 29, 37, 39] maintain routes between each pair of hosts in the network. Reactive, or on-demand protocols [28, 30, 38, 40], create routes only when requested by a particular source and maintain them only until they are no longer used. Performance studies across these two broad categories have been widely performed [4, 6, 12]. In general, while proactive protocols have a lower latency for the initial use of a route, the extensive overhead incurred in maintaining routing information that is never used makes reactive protocols a better choice in most dynamic scenarios. The Zone Routing Protocol [22] is a hybrid that leverages proactive behaviour within a local "zone" surrounding a node and switches to reactive behaviour outside of that zone. A protocol that adaptively modifies the degree of proactiveness is discussed in [41].

These routing protocols require the application to provide the unique address of the destination in order to create a route. This is analogous to routing in traditional wired networks, where applications take advantage of the static nature of the network topology to gather these unique addresses. Applications contact Domain Name System (DNS) servers to resolve the unique IP address of a host, given a higher-level description (the host name). This independence is feasible because the set of hosts a user wants to contact is relatively static, and the information needed to resolve names can be cached for long periods of time.

2.2 Service Discovery Mechanisms

While the protocols mentioned in Section 2.1 address the need for finding a path to the destination once the unique address is discovered, this section deals with protocols that operate on top of the routing protocols. The protocols mentioned here identify nodes that potentially satisfy the resource/service requirements of a source.

Recent work has focused on building DNS equivalents for mobile ad hoc networks [14, 15] that use reactive routing and multicast to create a name resolution phase that occurs before the routing phase. *Service discovery* approaches [2, 3, 11, 13, 16, 20, 21, 33], add a level of indirection by allowing applications to query resolvers based on descriptions instead of names. *Publish-subscribe* systems disseminate information in systems with multiple receivers desiring the same messages. Senders *publish* messages with topic labels. Each message is then sent to all receivers that have *subscribed* to receive messages matching the content or topic. Publish-subscribe systems provide a service similar to our goals, and the concept has been applied successfully in infras-

tructure mobile networks [7] and even in mobile ad hoc networks [45]. The philosophical bases of name resolution, service discovery, and publish-subscribe approaches assume that multiple subscribers will be simultaneously interested in the same publication. As a result, the architectures use varying degrees of proactive behaviour for resources or data to announce or advertise their presence. In highly dynamic networks, this generates significant overhead that is often not necessary given the expected behaviour of applications.

Our work is not the first to propose application-level or content-directed communication. Content Based Multicast (CBM) [46] pushes messages to receivers based on the message's content. This approach is complementary to ours in that they support applications which require strictly push interactions. Network Abstractions [42] uses a multicast to collect and maintain a set of the identities of hosts that satisfy an application level property. Messages are subsequently sent only to the collected set of nodes. Application-oriented routing [36] extends TORA [38] to create a service provision architecture. This approach maintains a combination of proactive and reactive behaviour, requiring hosts to perform topologically limited advertisements for services they wish to offer to other hosts. Such an approach is targeted towards scenarios in which applications share common interests and are therefore often looking for similar things. Finally, Person-Level Routing [43] maintains person-to-person connections in the face of mobility. It is not designed specifically for ad hoc networks and therefore relies on a centralized infrastructure. In addition it supports applications that need to contact specific *people* and not necessarily generically specified resources.

Work more closely aligned with our goals also integrates resource discovery and route construction in mobile ad hoc networks [18], providing an

implementation of the architecture requirements first elucidated in [32]. This work enhances AODV [40] to simultaneously discover services and routes to them, but the approach assumes a predefined and well-known mapping of service descriptions to fixed length integers. This significantly limits the flexibility, dynamics, and expressiveness attainable. Directed diffusion [26] is an attribute based routing scheme targeted directly for sensor networks. The communication occurs in two "phases;" the exploratory phase creates a network of gradients (and floods responses back to the requester). The "best" gradients are subsequently selected through reinforcement. This protocol operates in environments where sensor nodes commonly coordinate to perform a specific sensing task and can take advantage of this cooperation to aggregate messages destined for a sink node. In addition, the directed diffusion implementation makes strong assumptions regarding an agreed-upon naming representation. The target communication environment for this thesis are networks where the traffic flows are neither predictable, persistent, nor deterministic; and many nodes serve as "sinks." A key contribution to the already existing body of work is an evaluation of the impact of routing with non-fixed length addressing on the overhead of communication in highly mobile environments.

Chapter 3

A Cross-Layer Protocol

This chapter describes in detail a protocol that performs resource discovery in ad hoc networks across many layers. The novelty of our approach lies in the fact that we achieve support for realistic applications, while having an acceptable impact on system performance, especially when compared with viable alternatives.

3.1 Request Specification Language

Our protocol's routing packets carry an application level specification of the destination host instead of its fixed length network address. This is the most important aspect both in terms of providing increased expressiveness and flexibility and in terms of negative impacts on performance.

A host may provide a number of capabilities, store different types of data, or satisfy varying requirements (e.g., it may be connected to a printer or display, it may function as an FTP server or a database server, or it may collect local traffic or weather information). In the first responders application, a vital sign monitoring device provides information about an injured individual. In the case of traffic monitoring on highways, both automobiles and kiosks on the sides of the road may monitor traffic density. In general, a host that wishes to communicate in a mobile ad hoc network does not know *a priori* which other

host(s) will satisfy its needs.

Due to the popularity of service provision systems, many possible solutions for providing descriptions and specifications exist [5, 10, 13, 21, 44]. In general, successful, decentralized approaches use *semi-structured data* [1] where attributes are related hierarchically. The matching capabilities can be described as selections of descriptions according to restrictions on the semi-structured data. We assume that not only are the capabilities (i.e., resource, services, etc.) a host provides described in such a manner, but that application data is also structured so that it can be matched by similarly structured queries. The determination of such structures, especially in the case of data items, is likely to be application dependent and this is one of the major motivations for a *cross-layer* design, i.e., a design that performs actions not only at the network layer but also at the application layer. The specifics of the description scheme used are not important, and a particular application or network deployment may choose to swap out one specification language for another (we use a simple example scheme in Chapter 4. As such, our protocol must function independently of *how* matching decisions are made. Therefore, for now, we forego any use of optimizations at the matching level; we assume the only knowledge shared between the hosts *a priori* is that of the structure of the specification scheme.

A cross-layer design increases the overhead of our protocol in two ways. First, routing decisions are made at the application level, which requires passing the routing packets up the network stack, thus increasing processing overhead. Second, the packets are no longer of a fixed length, which has the potential to impact the end-to-end latency of the protocol. Optimizations targeted to mitigating these negative effects are examined in Chapter 5.

3.2 CDR Protocol Fundamentals

Our protocol, *Cross-layer Discovery and Routing* (CDR) enables route discovery between two hosts in a mobile ad hoc network based solely on attributes of the destination host, its resources, or its data, as desired by a particular source. As part of discovery, a source route is generated that contains a list of the hosts connecting the source to each potential service provider. Throughout our description of the protocol, we provide an abstract model of the behavior in addition to the standard textual description. This eases understanding of the protocol, makes explicit the state maintenance needs at each host, and guides the careful design of our implementation.

3.2.1 Packet Types

CDR in its basic form, uses the following packet types to interact with peer protocol entities residing on other nodes in the network. Figure 3.1 shows in detail the information contained in each of these packets. Details of how CDR uses these packets and the information in each packet is given in Section 3.3.

1. *Resource Discovery* - A *Resource Discovery* (RD) is sent when a source host requires a service or data that it does not itself have, thereby necessitating the discovery of another host. An RD carries with it the host's desire to create a route to a destination as specified by a restriction on semi-structured data.

2. *Route Reply* - A *Route Reply* (RR) is generated in response to an RD reaching a node that satisfies the request (according to the matching algorithm in use).

3. *Application Packet* - An *Application Packet* (P) contains the application payload handed down to CDR from the application(s) above it in the protocol stack.

4. *Route Error* - A *Route Error* (RE) is generated in response to an application packet encountering an error (i.e., a broken link) within its specified source route.

Resource Discovery (RD):	$\langle seq\_num, source\_id, spec, route\_record \rangle$ contains a sequence number (used to distinguish different discoveries from the same host), the source's id, the resource specification, and the route record built so far.
Route Reply (RR):	$\langle seq\_num, source\_id, route\_record \rangle$ contains the same sequence number and source id, but contains the complete path.
Route Error (RE):	$\langle link\_end1, link\_end2, reverse\_route \rangle$ contains the two hosts between which the error occurred and the reverse of the original route (to deliver the error back to the source).
Application Packet (P):	$\langle packet\_num, source\_id, spec, route\_record, application\_data \rangle$ contains the data, a unique packet number, the source's unique id, the semi-structured specification of the destination, and the route record.

Table 3.1: CDR Packet Types

3.2.2 State Variables

To participate in CDR, each host stores several pieces of state information, relating to its previous and pending requests, existing routes, and a minimal amount of information stored about requests made by other sources. Some of the key state variables are described below.

- *RouteCache(spec)* - A set of source routes that originate from this host for a resource described by *spec*. This cache enables a node to select an alternate route in the event of a link failure on the route currently being used. An important advantage of using abstract attributes (*spec*) to identify a destination (rather than using an "address"), is that many nodes can possibly provide the service/resource requested for. Hence, in applications where stateless transactions are acceptable, CDR reduces the network packet overhead by using a destination that it has in its *RouteCache(spec)*, as opposed to sending a new RD when it detects that all routes to a given destination have failed.

- *SentPackets*(*packet_num*) - A buffer containing packets that have been sent recently by this node. Like other routing protocols, CDR provides a limited guarantee against packet loss due to link failures. Hence, when a node sees a link failure on the route currently being used, CDR tries to recover from the link failure by retransmitting the packet that created the RE packet on an alternate route selected from *RouteCache(spec)*

- *ResourceTable* - A table of resources/services this node can provide. When this node sees an RD, it searches this table to find out if it can service the request. If it can, it sends out an RR to the source of the RD.

A summary of all state variables is shown in Figure 3.2. The figure shows the state held by a single host; every host has its own set of these variables.

3.3 Protocol Actions

This section describes the behaviour of CDR in response to events generated by other protocol entities — applications on top of CDR, and peer CDR protocol entities residing on other hosts in the network.

We use I/O Automaton notation [35] to describe the protocol's behavior. We show the behaviors of an individual host A, indicated by the subscript A on every action. Each *action* (e.g., SENDAPPLICATIONPACKET$_A$) has an effect guarded by a (possibly empty) precondition. Actions without listed preconditions are *input actions* triggered by another host. In the model, each action is executed in a single atomic step.

To abbreviate the formal description of the protocol, we make two assumptions. We assume each host only attempts to send one application packet at a time and waits until a send succeeds before any subsequent attempts. We also assume that a satisfactory destination exists and will be discovered. Both of these assumptions are particular to our abstract description and are removed in the actual implementation. In the latter case, we use a time-to-live (TTL) flag to indicate that we should stop propagating a resource discovery. We abuse standard I/O Automata notation slightly by using, for example "send *ResourceDiscovery*(RD) to *Neighbors*" to indicate a sequence of actions that ultimately triggers RESOURCEDISCOVERYRECEIVED (Figure 3.3) on each neighbor.

Neighbours	the set of neighbouring hosts (i.e., hosts to which this host is directly connected); used in our abstract description to demonstrate the reactive behavior of the hosts. In the implementation, we use broadcast and therefore do not require this table.
KnownRequests	a record of the Resource Discovery (RD) packets this host has seen. This ensures that flooding is marginally controlled.
RouteCache(spec)	the routes that satisfy *spec*. The routes are sorted in order from lowest to highest latency.
PendingPacket	the packet waiting for a route discovery
seq_num	the sequence number for this host's resource discoveries; it is incremented for every new discovery to ensure that the discovery is forwarded only once by each host.
packet_num	application packet sequence number sent by this host; may be used in resending packets that experience errors.
SentPackets(*packet_num*)	application packets sent by this host.
ResourceTable	the semi-structured descriptions of this source's resources
Resends	application packets queued to be resent due to transmission failures.

Table 3.2: CDR State Information

3.3.1 Application Interaction

To use the protocol, an application sends packets to destinations designated by restrictions on semi-structured data. Figure 3.1 shows the send

action triggered by the application. As the figure shows, an application triggers SendApplicationPacket when it has data to send. The data and the resource description are encapsulated in p. If no satisfactory route exists in the route cache, the action initiates resource discovery by creating a *Resource Discovery* (rd) and sending it to each neighbour (or, in the implementation, simply broadcasting). When the discovery process completes, the *RouteCache* will contain at least one route to a satisfactory destination. This enables the second action in Figure 3.1.

```
SendApplicationPacket_A(p)
  Precondition:
    PendingPacket = null
  Effect:
    PendingPacket := p
    if RouteCache(p.spec) = ∅ then
      rd := ⟨ seq_num++, A, p.spec, {A} ⟩
      send ResourceDiscovery(rd)
          to Neighbours
    end

TransmitApplicationPacket_A(P)
  Precondition:
    PendingPacket = p
    RouteCache(p.spec) ≠ ∅
  Effect:
    route := RouteCache(PendingPacket.spec).head
    p.route_record := route
    p.packet_num := ++packet_num
    send ApplicationPacket(p)
        to p.route_record.successor(A)
    SentPackets(packet_num) := p
    PendingPacket := null
```

Figure 3.1: Sending an Application Packet

To send the application packet, the host selects the first available route for the specification ((*RouteCache*(P.*spec*)).*head*). The selected route serves as the packet's *route_record*. The route building process and ordering of routes in the route cache are discussed later in this section. Each packet has a unique number (*packet_num*) which we use when errors occur. The simplified notation "send *ApplicationPacket*(P) to P.*route_record.successor*(*A*)" is used to indicate that ApplicationPacketReceived(P) is triggered on the host whose id is the same as the second host in the *route_record* (i.e., *route_record.successor*(*A*)). In the implementation, this is equivalent to unicasting the packet. After propagating the packet, the host stores a copy of it in *SentPackets*, which is used in the event of a transmission failure.

```
ApplicationPacketReceived_A(P)
  Effect:
    if P.route_record.tail = A then
      deliver application packet
    else
      if P.route_record.successor(A) ∈ eighbours then
        send ApplicationPacket(P)
            to P.route_record.successor(A)
      else
        reverse_route := reverse(P.route_record)
        RE :=
          ⟨ A, P.route_record.successor(A),
            P.packet_num, reverse_route ⟩
        send RouteError(RE)
            to RE.route_record.successor(A)
      end
    end
```

Figure 3.2: Propagating an Application Packet

Figure 3.2 shows the action ApplicationPacketReceived. First

the host checks to see if it is the intended destination (by comparing its own id to P.*route_record.tail*). If the host is the intended destination, the packet is delivered to the application. Since much communication (even in mobile ad hoc networks) is bidirectional, this reception will likely generate an application level response to the source that sent the application packet. This creates another packet and uses the reverse of the packet's route. In the event that the routes are not bidirectional, an additional route discovery may be necessary for this return message where the destination specification simply contains the unique id of the original source.

If this host is not the intended recipient, the packet is propagated further by the following mechanism. The host first selects the next host in the route record (i.e., *route_record.successor*(A)), and then triggers the ApplicationPacketReceived(P) action on the selected host.

Finally, if the next link referred to in P's *route_record* no longer exists, an error message is generated. This error message serves as a single attempt to notify the original source that the delivery failed. It uses the reverse of P's route record to target the source host. We discuss the propagation and processing of route error messages later in this section.

3.3.2 Resource Discovery

When an application above CDR attempts to send a packet, and a route for the specified description is not present in the node's *RouteCache*, the source performs the action "send *ResourceDiscovery*(RD) to *Neighbours*". This action triggers ResourceDiscoveryReceived, shown in Figure 3.3, on each of the source's neighbours (as was explained earlier). A receiver of an RD first checks the *route_record* to ensure there are no routing loops. The

```
ResourceDiscoveryReceived_A(RD)
  Effect:
    if A ∉ RD.route_record then
      if ResourceTable satisfies RD.spec then
        RR := RD
        RR.route_record := RD.route_record + A
        send RouteReply(RR)
            to RR.route_record.predecessor(A)
      else if ⟨ RD.source, RD.seq_num ⟩ ∉
          KnownRequests then
        KnownRequests :=
          KnownRequests ∪ {⟨ RD.source, RD.seq_num ⟩}
        RD′ := RD
        RD′.route_record := RD.route_record + A
        send RouteRequest(RD′)
            to Neighbours
      end
    else
      drop the packet to avoid routing loops
    end
```

Figure 3.3: Propagating a Resource Discovery Packet

receiver then determines whether or not it can act as a destination for the discovery. While the simple one-line "**if** *ResourceTable* **satisfies** RD.*spec*" performs this check in our model, the check uses application specified information from the source, application provided information on this host, and an application-defined mechanism for determining matches between the two. This necessitates the protocol's cross-layer design, as application-level information must be accounted for in the resource discovery process. If this host does not satisfy the specification, it ensures that it has not previously processed the same request by checking *KnownRequests*. If the packet has not previously been processed, the receiver adds it to *KnownRequests* and continues to

propagate the resource discovery by appending its id to the route record and triggering the ResourceDiscoveryReceived on each of its neighbouring hosts.

In the case in which this host can serve as a destination, the host generates a Route Reply (rr) that it returns to the original source. The rr propagates back to the original source using the reverse of the discovered route. It starts this process by triggering RouteReplyReceived, shown in Figure 3.4 on its predecessor in the *route_record*. Unless the host is the source,

```
RouteReplyReceived_A(rr)
  Effect:
    if rr.source = A then
      RouteCache(rr.spec) :=
        RouteCache(rr.spec) + rr.route_record
    else
      send RouteReply(rr)
          to rr.route_record.predecessor(A)
    end
```

Figure 3.4: Propagating a Route Reply Packet

RouteReplyReceived simply triggers the same action on its predecessor in the *route_record*. If this host is the source, the route carried by the reply is stored in the *RouteCache* and associated with the appropriate application-level specification (rr.*spec*). For an initial route discovery, this insertion into the *RouteCache* triggers TransmitApplicationPacket shown in Figure 3.1.

This description assumes that the network has symmetric links and that it is therefore possible for the packet to traverse the reverse route. If this is not the case, the destination must perform a reverse resource discovery

to the source using the source's unique network address as the specification requirement. The RR can be piggy-backed on the reverse RD.

A single destination may reply to the resource discovery message more than once to indicate multiple routes to it from the source. The source maintains these multiple routes in the *RouteCache* in case the preferred path fails. In CDR, unlike existing mobile ad hoc routing protocols, it is also possible that multiple destinations will satisfy a resource request.

In our initial protocol, a source simply selects the first host from which it receives a route reply (subsequent replies are simply added to the end of the *RouteCache* list for the specification). The discussion in Chapter 5 addresses this design decision and using context properties and context-sensitive requirements of the application to select the best path according to different metrics (our current metric being "resource discovery latency"). This includes readjusting the destination when the source discovers a new destination that is "better" by an application specified metric.

3.3.3 Route Error Propagation

When a link breaks, data transmissions encounter errors. The host detecting the broken link sends a *Route Error* (RE) to the original source. On receiving an RE, if the host's route cache contains no additional routes for the desired specification, the source reinitiates route discovery.

In Figure 3.2 we see how the RE is generated by the host that detects the broken link. This host initiates the propagation by triggering RouteErrorReceived on the previous host in the source route. Figure 3.5 shows how this packet is propagated back to the source node. The propagation of an RE does not guarantee that it reaches the original source; it may encounter link

failures itself, and we do not attempt to recover from these. In such cases, the original source may not learn that its packet was not properly delivered. Applications should maintain internal timers in order to detect such anomalous situations. A host receiving an RE deletes any routes in its *RouteCache* that

```
RouteErrorReceived_A(RE)
  Effect:
    for each route ∈ RouteCache do
      if RE.link_end1 ∈ route and
        RE.link_end2 = route.successor(link_end1) then
          RouteCache := RouteCache - route
      end
    end
    if RE.route_record.tail = A then
      Resends := Resends ∪ SentPackets(RE.packet_num)
    else
      send RouteError(RE)
          to RE.route_record.successor(A)
    end
```

Figure 3.5: Propagating a Route Error Packet

also use the broken link. When the packet reaches the source, it pulls a copy of the packet that experienced the transmission error from *SentPackets* and queues it for retransmission on an alternate route.

To complete our protocol's specification, we must also ensure that these packets are retransmitted. To this purpose, we add the action shown in Figure 3.6, which is the same as SendApplicationPacket in Figure 3.1 except that the packet comes from *Resends* instead of directly from the application. This new action does not guarantee fairness between application sends and resends due to route errors; if the application continues to send packets, packets that need to be resent may never get the chance.

```
RetransmitPacket_A(P)
  Precondition:
    PendingPacket = NULL
    P ∈ Resends
  Effect:
    PendingPacket := P
    Resends := Resends - {P}
    if RouteCache(P.spec) = ∅ then
      RD := ⟨ seq_num++, A, P.spec, {A} ⟩
      send ResourceDiscovery(RD)
          to Neighbours
    end
```

Figure 3.6: Retransmitting a Failed Packet

The protocol can also benefit from optimization that enables quicker recovery from link failures. If a link fails upon delivery of an application packet, the host detecting the failure can attempt a local recovery. This is best accomplished if an application running on the host that detects the link breakage has requested a route for the same specification (or for a specification that subsumes the requested one) as that in the packet that detected the link failure. This, however, requires storing each applications' specification at every host, an overhead we are not willing to accept at this time due to the large numbers of differing application requests. We have not yet evaluated our protocol under this optimization. However, the host detecting a link failure can perform a simpler optimization, by checking its personal route cache for a route to the same destination (determined by the unique address within the source route). If such a route exists, the host attempts to use the alternate route.

Chapter 4

Performance Evaluation

This chapter provides a first step in demonstrating the feasibility of incorporating resource directed routing into highly dynamic mobile applications. This performance demonstration characterizes the impact such a modification will have on performance metrics such as the latency of discovery, the packet delivery ratio, the message delivery delay, and the control overhead of the protocol. We used the *ns-2* [17] network simulator to generate these results.

4.1 Simulation Settings

The following parameters were used to configure *ns-2* for all the simulations.

- *MAC Protocol* - The IEEE 802.11 MAC Distributed Coordination Function (DCF) [25] was used, which uses a RTS/CTS exchange to initiate unicast messages and CSMA/CA for broadcast packets. The former allows a sender to receive an acknowledgement when packets are successfully sent. If the protocol doesn't receive an ACK back from the destination within a certain timeout period, it assumes that collision has occurred. It then listens to the medium after a random backoff period, and does the RTS/CTS exchange again. During broadcast however, this mechanism is not used. Hence, using the IEEE 802.11 protocol ensures

that routes carrying unicast packets use bidirectional links. Some of the stringent assumptions made in the formal descriptions are thereby removed.

- *Traffic pattern* - We used network scenarios similar to those found in [4, 6, 12]. Specifically, we utilized node mobility patterns based on *random waypoint* mobility [6] and distributed 51 nodes in a rectangular field of size 1500m × 300m. The speed of the nodes was uniformly distributed between 0.01 and 20m/s, with the exception of one node, which is stationary in the center of the field. The purpose of the stationary node will be explained further in the chapter. Throughout the experiments, we vary the nodes' pause times from 0 seconds (for high mobility) to 900 seconds (for relatively static networks). Each simulation is run for 900 simulation seconds, and unless otherwise specified, the traffic between sources and destinations for all protocols was generated at the rate of 20 packets per second. All data packets were of size 512 bytes.

- *Statistics collection* - Each plotted point indicates an average over 200 samples, a "sample" being defined as a simulation run over a particular scenario of node movement and positioning, and possess the same statistical characteristics described above. 95% confidence intervals are shown for all graphs, showing how reliable the simulation runs are, and how they relate to each other statistically.

4.2 Performance Metrics

The most significant aspect of our protocol that will impact performance is the inclusion of a non-fixed length description of the destination

in the discovery packets. For this reason, we have selected metrics that will highlight the impact of this design decision on overall performance. In the discussion below, a "destination" is defined as any node that can provide the resource/service as specified in the *Resource Discovery* packet it receives.

- *Route discovery latency* - This is the amount of time a source takes on average to learn the first satisfactory route to a destination.
- *Data delivery latency* - This metric determines on average how long it takes to deliver a data packet from an application on the source to an application on the destination.
- *Packet delivery ratio* - A measure of the fraction of the packets sent by the routing agent that are actually delivered at the destination.
- *Application Packet delivery ratio* - This is a measure of the fraction of the packets that are intended to be sent by the application above the routing protocol that are actually delivered at the destination. The need for this metric is discussed later in the chapter.
- *Normalized packet overhead* - The number of control packets sent in the network for each data packet delivered, per node in the network. This measures the average amount of control traffic generated by a node over an extended period of time. In addition to packets generated by the source and the destination, this metric counts packets that were forwarded by nodes in the middle of the network. Such a distinction is necessary owing to the fact that in mobile ad hoc networks, communication consumes a large fraction of the total power of a node in the network, and hence every transmission — application packet generation or forwarding, contributes to this total power consumed.

- *Normalized byte overhead* - This metric measures the number of bytes of control messages sent for each data packet delivered, per node — showing the effects of carrying non fixed-length addresses in the control packets. Again, this metric is power related since the length of the packet determines the power expended in transmitting it.

- *Average route length* - In application scenarios where it is meaningful, this metric shows the average length a data packet had to travel in order to reach the desired destination.

4.3 Protocols

To gauge CDR's improvement over the current state of the art, we compare it to a protocol we call DWD (DSR with Discovery). Such a protocol is representative of current approaches that utilize a lookup service to resolve the name or type of a service before subsequently contacting the node based on its id.

4.3.1 CDR

The CDR protocol functions as described in Chapter 3 with the following modifications from theoretical behaviour:

- Multiple data packets can be queued at the source. As is common in the implementation of routing protocols, a CDR protocol agent contains a *send buffer*, which buffers packets upto a certain maximum buffer size. This maximum limit is configurable. For simulation purposes, this is an infinite buffer — a CDR agent will not block the application or overwrite old packets due to lack of space.

- A time-to-live field is used in CDR packets to prevent route requests from propagating forever. This TTL is decremented at each hop. In order to be comparable to DSR, this parameter was set to the same maximum as DSR.

- A node does not contain the *Neighbours* field (see Figure 3.2), as the IEEE 802.11 MAC protocol handles one hop communication between the nodes.

4.3.2 DSR with Discovery

The protocol we used for comparison was DSR with discovery (DWD). This protocol is illustrative of the *two-phase* approach described in Chapter 1. The discovery server resides on a stationary node at the center of the simulation area. This node is assumed to possess omniscient knowledge of the services offered by any node in the network. Before creating a route to a destination, a source in DWD must contact the lookup server with the description of its desired resource. The lookup server responds with the address of the node (or nodes) in the network that provide that service. This is similar to communication in the wired Internet, where a source host wishing to communicate with a destination first contacts a DNS server to resolve the destination's name and then employs a routing protocol to find a path to the destination. In *ns-2*, the lookup server was implemented as a new protocol agent running on top of the routing agent. All source and sink nodes in the network implement the "client" portion of this discovery agent. While source nodes use the DWD agent to discover a resource provider, the sinks need the agent during registration with the server. In the present implementation overhead due to registration has been ignored, and the server is assumed to know *a priori* the

addresses of the resource providers. Details of the protocol are elucidated in the sections below.

4.3.2.1 DWD State Variables

Figure 4.1 shows the minimal state required in the DWD agent in order to exhibit the behaviour described above. *ResourceLookupTable* is the table maintained at the DWD server, which contains the mapping for all possible resource requests to all possible destinations that can provide it. Destinations wishing to provide a resource must register here in order for their resources to be used. *ResourceProviderList* is the state held by all source nodes (or the "clients") in the network. This table is populated after the node contacts the lookup server with a resource description, and obtains a list of destinations that can provide the resource.

ResourceLookupTable	maps destination addresses to the resources they can provide. Any other additional information can also be added here, such as when a resource on particular destination node would time out, etc.
ResourceProviderList	maps resources that this node needs to destination addresses.

Table 4.1: DWD State Variables

4.3.2.2 DWD Protocol Methods

The protocol has two packet types - *Resource Request* (RREQ), which is sent by a client node when it wants to learn the destinations that can satisfy the application's intention as specified by semi-structured data in the RREQ.

In response to an RREQ, the DWD agent at the lookup server sends a *Resource Reply* (RREP) to the node that requested the information. This packet contains a list of destinations that can satisfy the RREQ. DWD performs the following actions in its interactions with applications above it and peer DWD agents.

- *Send Application Packet* - An application wishing to send data hands the packet down to the DWD agent on the node. The agent first searches this list to find a matching destination. If a suitable destination is found, the agent populates the data packet with the destination address, and hands it down to the routing agent, which is DSR in our case. If a suitable destination is not found, the node sends a *Resource Request* packet to the lookup server.

- *Send Resource Request* - The DWD agent sends this packet to the lookup server when there are no known routes for a resource as specified by the application above it. The routing agent below the DWD agent is responsible for finding a route from the source to the lookup server.

- *Send Resource Reply* - The lookup server sends this packet in response to a *Resource Request* from a source node. It contains a list of destinations that can provide the resource specified in the *Resource Request* packet and other information that the source can use to determine which route to pick.

All route maintenance actions are being delegated to the routing agent. The DWD agent thus acts as an intermediate layer between the application and the routing agent. The performance implications of adding another layer are discussed further in this chapter.

4.3.2.3 DWD Simulation Assumptions

In order to obtain a first order comparison between the two protocols, any extraneous communication that may add to the overhead was removed, and the following simplfying assumptions were made about the behaviour of DWD:

- The lookup node has complete knowledge of the service providers in the system over the duration of the simulation. We do not require service providers to register with the lookup service, so our measurements do not include the overhead associated with these registrations and their renewals.
- A source node looking for a particular service already knows the id of the lookup server and does not need to discover it. In pure mobile ad hoc networks, this communication would result in additional control traffic.
- Route creation and maintenance between sources and the lookup server, and between sources and destinations is performed by DSR.
- In comparing CDR and DWD, to ensure a degree of fairness to CDR, we have turned off optimizations from DSR as implemented in *ns-2*, pertaining to maintaining flow state in nodes in the middle of the network.

4.4 Resource Descriptions

For both CDR and DWD, the length of the resource request may have a significant impact on the performance of the protocol and the overhead it incurs. While packets are of higher length for all control packet communication

in CDR, DWD carries large packets only during the Resource Request RREQ and RREP phases. To determine how increasingly complex resource or data requests affect performance, we measure each of our metrics (described in Section 4.2) using increasingly complex resource requests.

Description	**Size (bytes)**	**Label**
(service=printer)	18	size-1
(service=printer(location=ACES(floor=fifth)))	46	size-2
(service=printer(location=ACES(floor=fifth(type= laserjet(resolution=720dpi(print_on_both_sides=yes))))	106	size-3

Table 4.2: Resource descriptions used

Figure 4.2 shows the requests that we used and the sizes (in bytes) of those requests. The example application we chose to use is one where a source node requests traffic information with increasing specificity. By also measuring the length of the requests in bytes, we can generalize these measurements to other applications. The resource descriptions are written in a generic semi-structured data format (as shown in Figure 4.2), but in the implementation, any other description representation can be easily swapped in for this simple language.

4.5 Basic Performance Evaluation

In this section, we compare CDR and DWD in a basic sense. A single node tries to discover a node that can service its request. In this set of experiments, we placed only a single node in the network that could satisfy the source's request. In effect, there is a "single flow" in the entire network, allowing us to determine the lower bound on the performance of CDR. For

the CDR and DWD results in this set of experiments, we show the results for description *size-1* (see Figure 4.2). In addition, nodes selected as service providers offer the service for the lifetime of the simulation.

4.5.1 Discovery and Delivery Latencies

Figure 4.2 shows the resource discovery latency for the two protocols. This metric measures the time between the instant an application requested a resource, and the instant at which the application could send the first packet to the discovered resource provider. As can be seen from the figure, the time required for a node to discover a route to a satisfactory service provider is much lower for CDR than for DWD. The reason for this will become apparent when we consider how DWD discovers a route to the final destination. Consider the DWD route discovery mechanism for DWD, which is reproduced here from Chapter 1, in Figure 4.1.

When an application running on top of DWD wishes to communicate with a resource provider, the node must find out which nodes can possibly service its request. In order to do this, the node must contact the discovery server. Though the node knows the id of the discovery server, due to mobility, it may not always have a route to it. In this case, it must first find a route to the server. The node then queries the server for the id(s) of nodes that can provide the desired resource. The time taken for this transaction is labeled t_1 in Figure 4.1. The source node (node A in Figure 4.1) then performs a route discovery to find the resource provider. Let us say that the time taken for this is t_2. The total resource discovery time then becomes

$$t_{total} = t_1 + t_2 \tag{4.1}$$

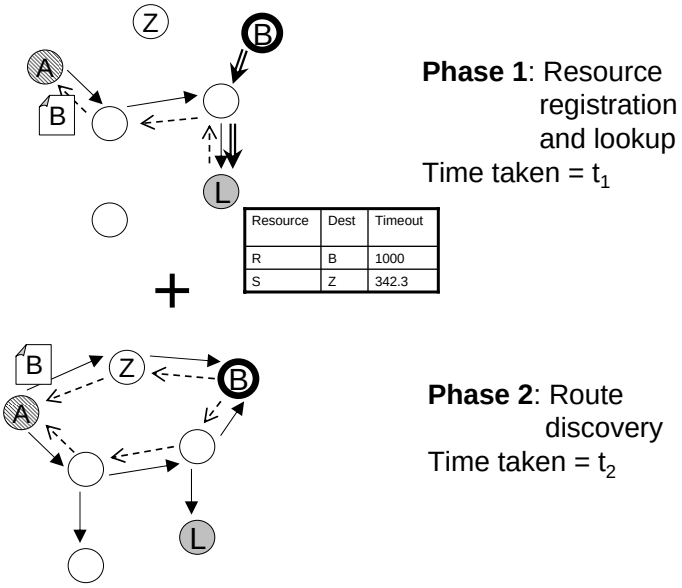

Figure 4.1: The two phase approach showing the different discovery times

In mobile ad hoc networks, due to link failures and in some cases network partitioning, it is difficult to estimate an average value for t_{total}, as

1. the discovery server may be many hops away from the source node or even unreachable. This increases t_1, or

2. the chosen resource provider may be far away (see Figure 1.4) or unreachable (see Figure 1.3), increasing t_2. The implication is that DWD does not the "best" destination, as opposed to CDR.

Both cases adversely affect the discovery time for DWD. CDR, however, has lower resource discovery times because of single phase communication employed. In CDR, the resource discovery latency is highly correlated with the proximity (in hops) of the service provider to the source. Theoretically, the

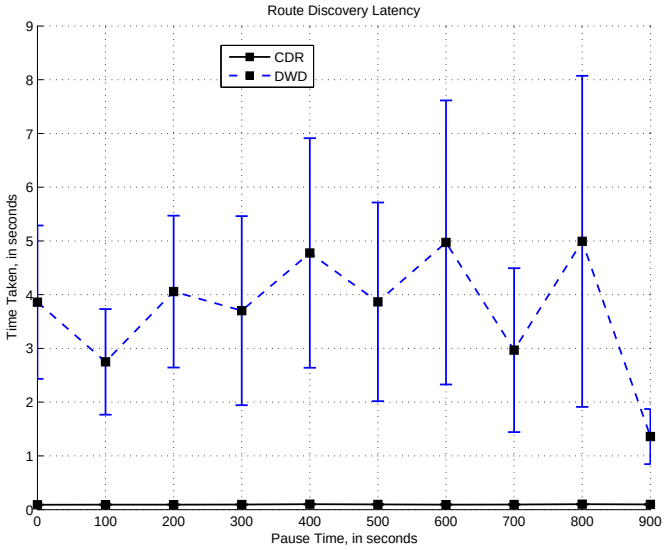

Figure 4.2: Route discovery latency for single traffic flow

upper bound on CDR's performance is DSR, since DSR employs the same mechanism for route discovery as CDR. DWD, in its basic form cannot utilize proximity information without employing an additional algorithm that keeps track of the absolute locations of different nodes in the network, which is reinforced by the wide confidence intervals for DWD in Figure 4.2. In mobile ad hoc networks, where it is likely that the server itself is mobile, keeping such information current incurs a lot of communication overhead, and hence infeasible.

The data delivery latency as shown in Figure 4.3 measures the time elapsed between an application on the source node sending a packet and a satisfactory destination receiving the packet. It can be seen from Figure 4.3 that data delivery latency monotonically decreases for both protocols as mobility decreases. As the pause times increase, the mobility of the nodes decreases

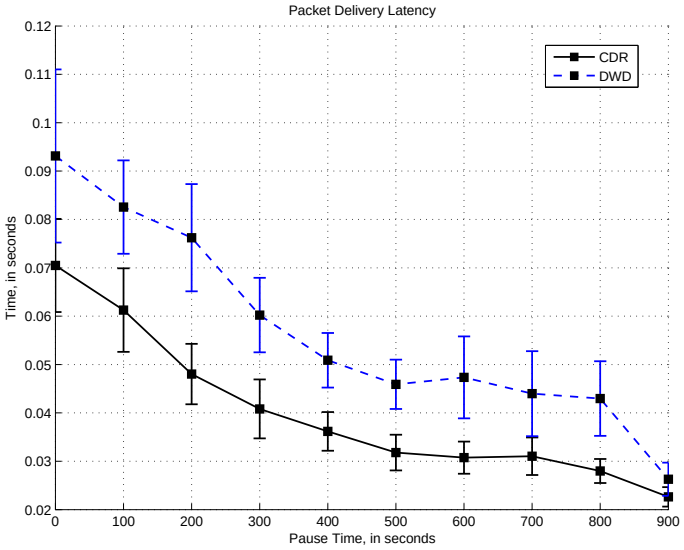

Figure 4.3: Data delivery latency for single traffic flow

resulting in fewer link breakages. It is also observed that data delivery latency for CDR is lower compared to DWD. This trend is related to the route discovery latency, since during the discovery phase, some packets are queued in the send buffer, waiting for a route to be discovered. In DWD such packets need to wait for an interval equal to the resource discovery latency, t_{total} (see Equation 4.1) to elapse before they can be sent. Once sent, these packets experience the average latency that a packet experiences when DSR is used for routing packets. Let us call this latency t_{DSR}. Hence, the variance of data delivery times is high in DWD, with the lowest average latency being t_{DSR} and the highest average data delivery latency being $t_{total} + t_{DSR}$.

The effect of resource discovery latency over data delivery times is amortized over time, as a resource discovery is not performed for every data packet that is sent. Due to amortization, the occasional resource discoveries manifest

themselves as a constant offset added to t_{DSR} in the data delivery latency. The delivery times for CDR exhibit a smaller variance, due to the single phase nature of resource discovery. On average, this latency is much lower than t_{total}. In cases where CDR does not have to perform a discovery, the data delivery time for CDR should theoretically be equal to t_{DSR}.

4.5.2 Delivery Ratios

Packet delivery ratio is considered in the following paragraphs. Figure 4.4 compares the packet delivery ratio for the two protocols. In our experiments, *packet delivery ratio* is defined as the ratio of the packets received at a resource provider to those that were actually sent by the routing agent. While

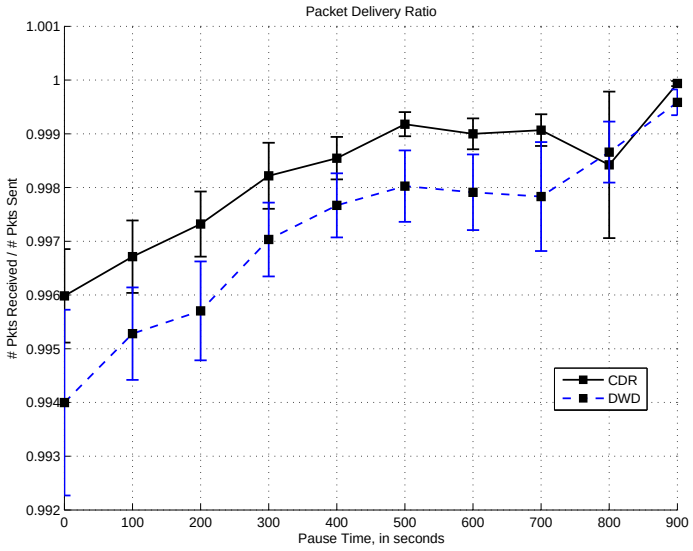

Figure 4.4: Packet delivery ratio for single traffic flow

looking for a route to the destination — either when the application is looking for a resource provider for the first time or due to link failures, packets

are buffered in the send buffer of the routing agent. Some of these packets time out before a satisfactory route is sent back to the source that initiated the discovery. In packet delivery ratio, these lost packets are not taken into consideration, as they were never attempted to be transmitted. *Application packet delivery ratio* takes into account these buffered packets as packets that were buffered. Buffered packets are representative of the application packets that could "potentially" be sent, but were not; and is defined as the ratio of the number of packets received to the number of packets that were attempted to be sent by the application. By definition, this ratio is equal to or lower than the *packet delivery ratio*. Application packet delivery ratio is the delivery ratio as "perceived" at the application level. At the application level, this metric is a measure of how reliably the routing protocol below it would deliver the packets to the desired destination.

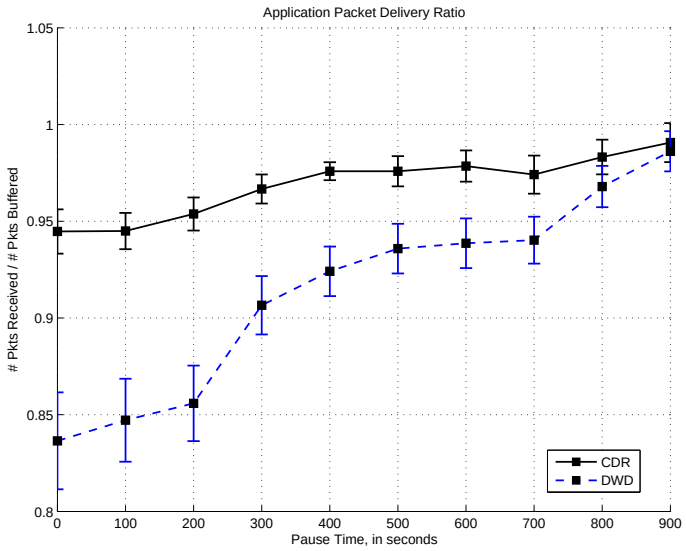

Figure 4.5: Application Packet delivery ratio for single traffic flow

In the simulations, the send buffer timeout for DSR, CDR, and the resource discovery agent were all set to 30 seconds. The timeout for the discovery agent is to ensure that if a source does not hear back from the lookup server within the specified timeout period, it will send another RREQ packet.

Figure 4.4 shows that CDR has a higher packet delivery ratio than DWD in the base case. As mobility increases, the packet delivery ratio decreases, but only marginally. With optimizations turned off for DSR in *ns-2*, route error recovery mechanisms (RERR propagation and packet retransmission) for both protocols are comparable to each other. The poorer performance of DWD can be attributed to the two phase communication mechanism. In certain scenarios, the source may be able to reach the potential service provider, but unable to reach the discovery server. If the source sent an RREQ in this situation, it would never reach the server. Eventually, some packets in the send buffer would timeout and be discarded, hence lowering the packet delivery ratio. This phenomenon doesn't affect the performance of CDR, as it removes the lookup phase altogether and contacts the resource provider directly. The issue is more evident in Figure 4.5, which shows that though the application packet delivery ratio for both protocols is worse than the packet delivery ratio (shown in Figure 4.4), DWD's application packet delivery ratio is affected to a greater extent than CDR.

4.5.3 Communication Overheads

In this section, we compare the protocols' network overhead. Overhead is measured in terms of *normalized packet overhead* and *normalized byte overhead*, as defined in Section 4.2. Figure 4.6 shows that the average packet overhead per node is comparable for both protocols. In a single traffic flow sce-

nario, where there is only one destination that can satisfy a source's resource requirements, CDR degenerates to DSR, provided the resource specification length is the same as the "address" field length in DSR's request packet. In this case, CDR's packet overhead should theoretically be the same as that of DSR's. As mobility decreases, the packet overhead reduces, as fewer packets are needed to discover and maintain routes, which constitute a major portion of the overhead.

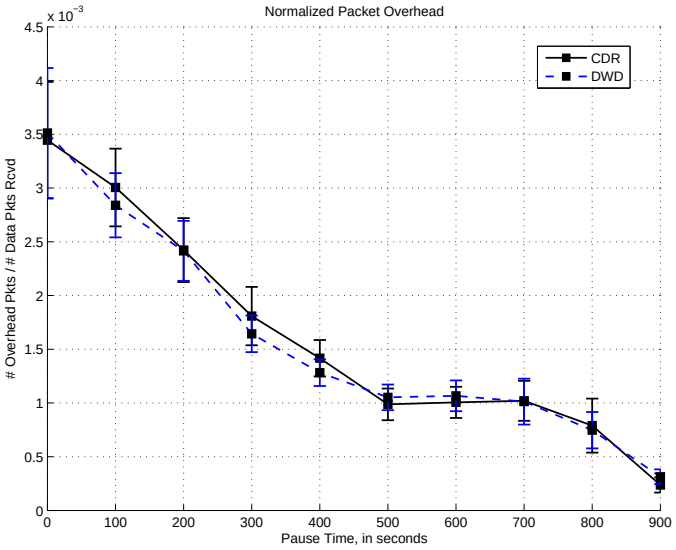

Figure 4.6: Normalized Packet Overhead for single traffic flow

On the other hand, DWD's packet overhead includes overhead incurred while discovering the lookup server (for *phase 1*) in addition to DSR's overhead (for *phase 2*). In the worst case scenario, a source may have to discover the server for every data packet sent, and the overhead for *phase 1* would be the same as that for *phase 2*, pinning DWD's overhead at approximately twice that of CDR's, which requires a single phase to communicate with the resource

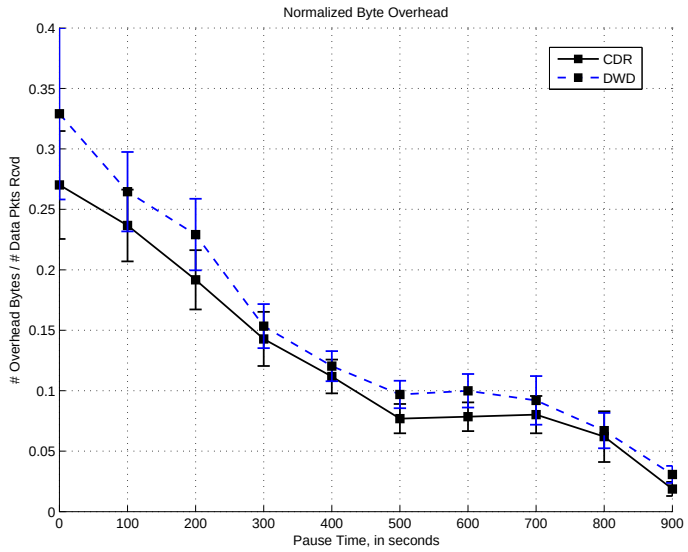

Figure 4.7: Normalized Byte Overhead for single traffic flow

provider. As can be seen from Figure 4.6, in the average case, DWD performs approximately the same as CDR in terms of packet overhead.

As for byte overheads, for small resource description lengths, the byte overhead for CDR is marginally lower than that of DWD's as seen in Figure 4.7. Again, this is attributed to the different approaches these protocols use in order to discover a resource provider, and a then discovering a route to the potential resource provider. DWD requests carry the description length during *phase 1*, and the fixed-length address during *phase 2*, while CDR behaves similar to DWD in *phase 1* and does not have a second phase at all. The overlapping confidence intervals for byte overhead graphs suggest that the two graphs are statistically indistinguishable from each other. However, as description lengths increase, CDR's byte overhead increases, as DWD uses these large descriptions only in *phase 1*, while CDR request packets are larger

in throughout. We demonstrate this variable length effect in the next section.

4.6 Impact of Variable Length Addresses

We devote this section to studying some of the interesting effects of variable length control packets on the performance of CDR, compared and contrasted to DWD. Experiments were designed along the same lines as Section 4.5 above, and a single flow of traffic was simulated. Graphs in this section include all results from above, in addition to graphs for more complex description lengths from Table 4.2. The graphs labeled 'len-2' and 'len-3' show the performance of CDR and DWD with descriptions *size-2* and *size-3* respectively, where an application is looking for a printer with increasingly complex specifications.

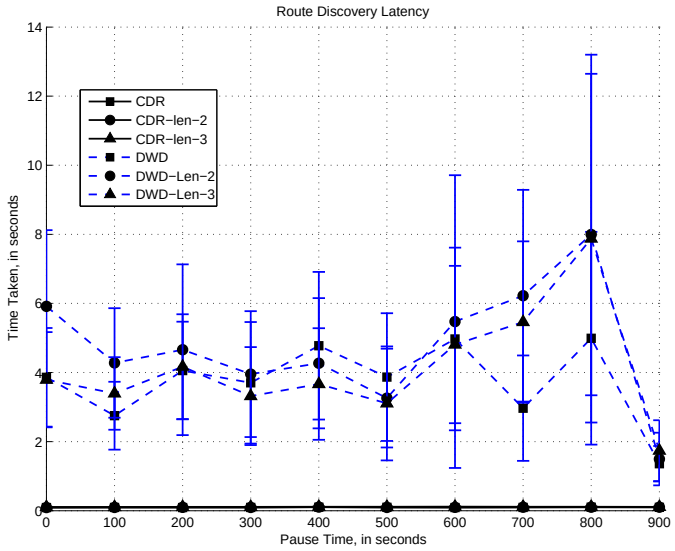

Figure 4.8: Resource discovery latency for a single flow with multiple request lengths

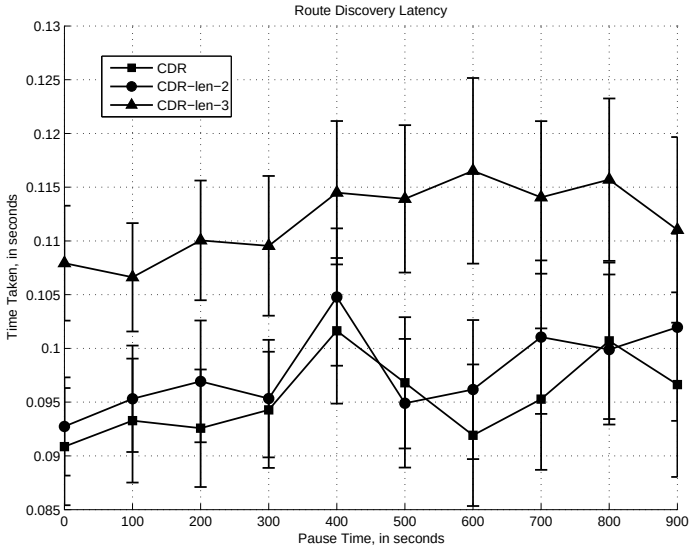

Figure 4.9: Resource discovery latency for CDR only, for a single flow with multiple request lengths

Figure 4.8 shows the resource discovery latencies for descriptions of different length. As discussed in Section 4.5.1, route discovery latency for DWD is much higher compared to CDR and shows a large variance. Figure 4.9 shows only the CDR portion of Figure 4.8. Although the discovery latency for requests of different lengths is much lower than that of DWD, there is a definite increase in resource discovery times as the description length increases. This is due to the fact that discovery packets may be very large, and are hence fragmented as they are handed to the protocol layers below. The different packet segments may take different amounts of time to reach the destination. The receiver has to then reassemble the different parts of the packet, thus increasing the overall packet latency.

Another significant consequence of varying resource description lengths is the effect on communication overheads. Figure 4.10 and Figure 4.11 show

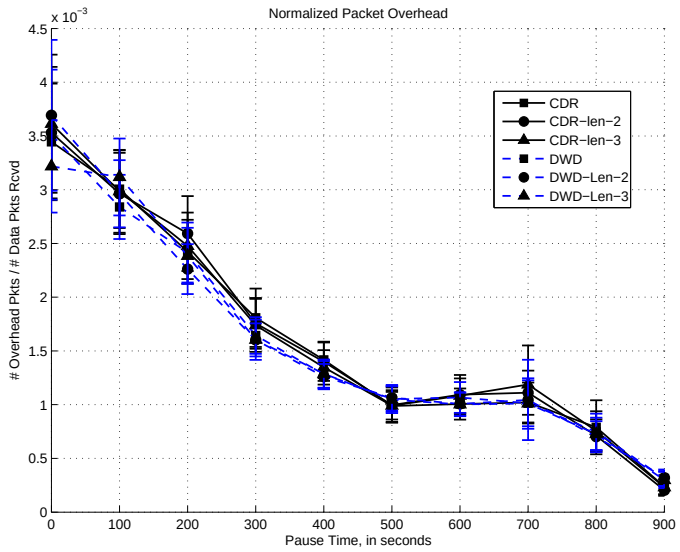

Figure 4.10: Normalized packet overhead for a single flow with multiple request lengths

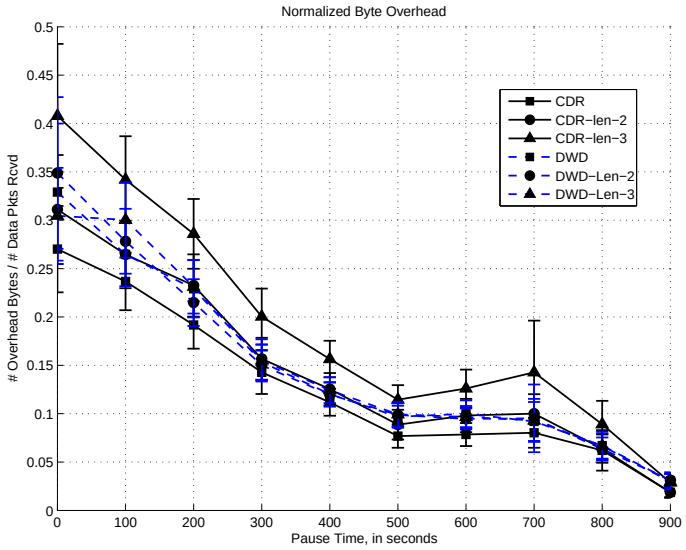

Figure 4.11: Normalized byte overhead for a single flow with multiple request lengths

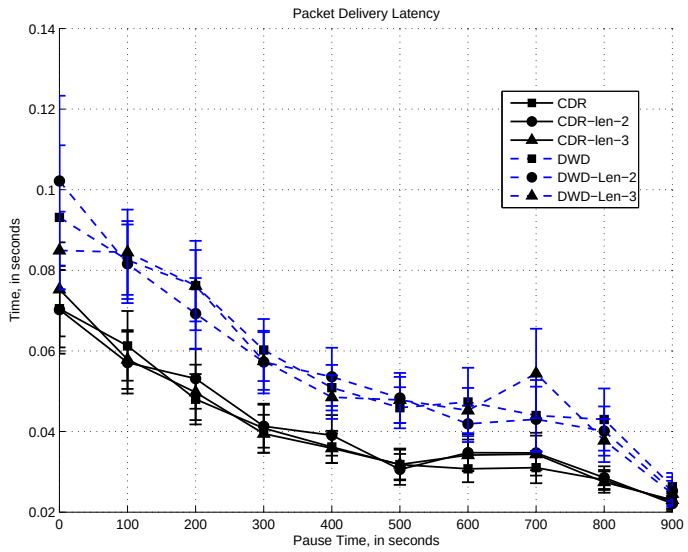

Figure 4.12: Data delivery latency for a single flow with multiple request lengths

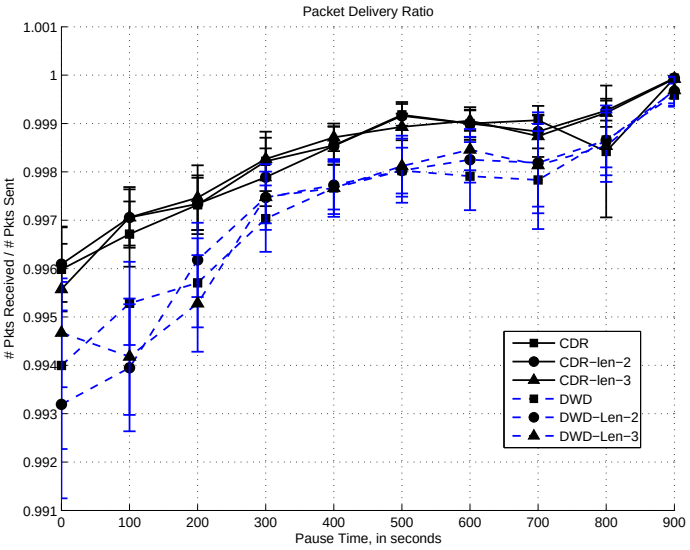

Figure 4.13: Packet delivery ratio for a single traffic flow with multiple request lengths

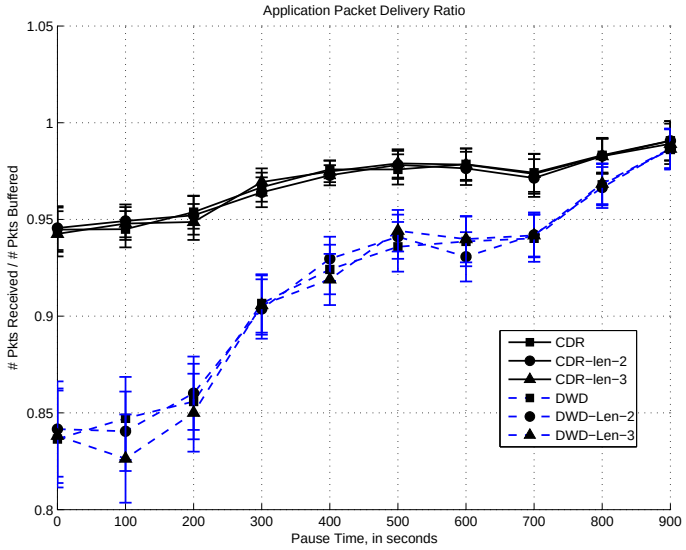

Figure 4.14: Application packet delivery ratio for a single flow with multiple request lengths

the normalized packet and byte overheads, respectively. Figure 4.10 shows that increase in the size of packets does not greatly affect the packet overhead in the network. However, observing the relative offsets between the byte overhead graphs for CDR and comparing them to the relative difference in the resource description lengths in Table 4.2, we can conclude that byte overhead increases approximately linearly with description length (see Figure 4.11). This emphasizes the need for expressive resource description languages, which result in shorter resource discovery packets, and yet completely specify the resource requirements of the source.

Figures 4.12, 4.13, and 4.14 show the metrics that are statistically not affected by the change in resource description lengths. Considering Figure 4.12, when a suitable resource provider has been discovered by a source, application data packets carry the unique id of the destination, instead of the resource

descriptions. Hence, data delivery latency is affected in the same manner as described in Section 4.5.1 above. Similarly, packet delivery ratios (shown in Figure 4.13 and Figure 4.14) do not change significantly as description lengths vary.

4.7 Effect of Resource Multiplicity

This section elucidates the performance of CDR and DWD in a network where multiple destinations satisfy the resource requirements of a source node. These scenarios are more likely to be representative of emerging mobile applications where multiple nodes can provide the same or similar functionality. In order to observe the effects of increasing the number of providers, we performed experiments with 2, 4, 8, 16 and 32 potential resource providers. We provide the results for a network with high mobility (pause time = 0 seconds); and a relatively static network (pause time = 500 seconds). There is still only one flow in the network, as in Section 4.5. The difference between the two scenarios is that in the single provider scenario, in the event of link breakages, the source node is forced to look for the same destination, while in the multiple provider scenario, the source may pick a route to an alternate provider from its route cache; provided the semantics of the application allows this. If the application requires a persistent connection to the service provider, a switch from one provider to another in the middle of a transaction may cause the application to behave in an undefined manner. We assume the existence of a stateless application which, upon detecting that all routes to a particular destination are broken, immediately picks another destination based on its requirements; and tries to send application packets to it.

We now look at the performance metrics for CDR and DWD in the

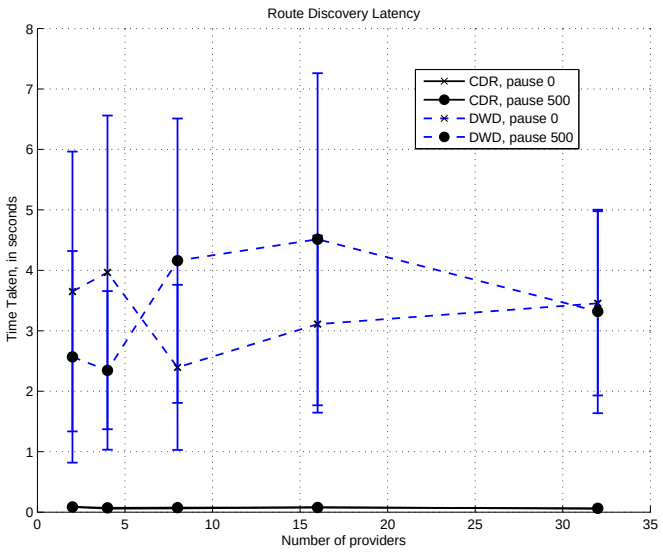

Figure 4.15: Discovery latency for a single requester, multiple providers scenario under different mobility conditions

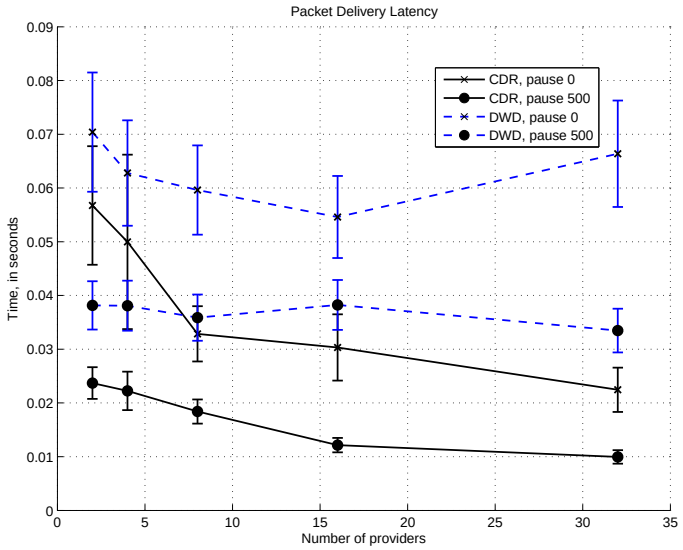

Figure 4.16: Data delivery latency for a single requester, multiple providers scenario under different mobility conditions

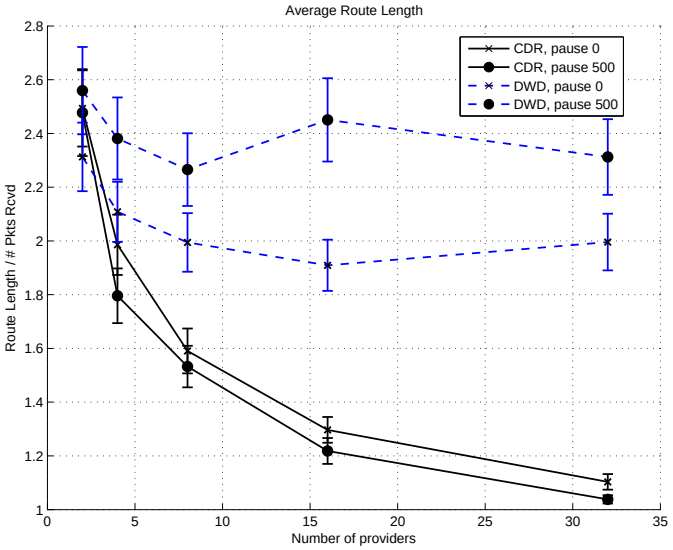

Figure 4.17: Average route length for a single requester, multiple providers scenario under different mobility conditions

multiple provider scenario. Figures 4.15 and 4.16 show the resource discovery latency and route discovery latency, respectively. In addition to reasons mentioned in Section 4.5 above, CDR outperforms DWD with respect to latencies, since an increase in the number of providers increases the probability that a provider may be closer to the source. Discovering a physically nearer resource provider is quicker for CDR, as it employs a single phase discovery approach, and hence the discovery times for CDR are much lower than for DWD.

The above mentioned theory is corroborated by the graphs for average route length, shown in Figure 4.17. *Average route length* is defined as the number of hops a data packet has to travel on average to reach the destination. Mathematically, it is the average route length per data packet received at the resource provider. The graphs show that as the number of providers increases, a simple flooding mechanism, on average, selects a provider that is closer to the

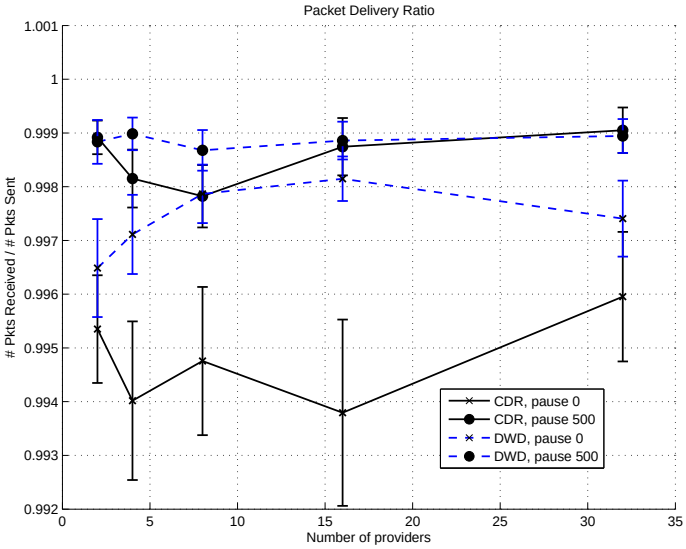

Figure 4.18: Packet delivery ratio for a single requester, multiple providers scenario under different mobility conditions

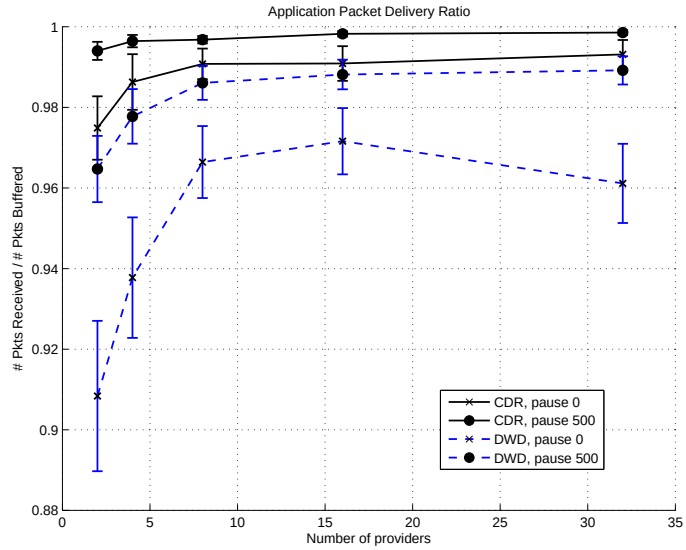

Figure 4.19: Application packet delivery ratio for a single requester, multiple providers scenario under different mobility conditions

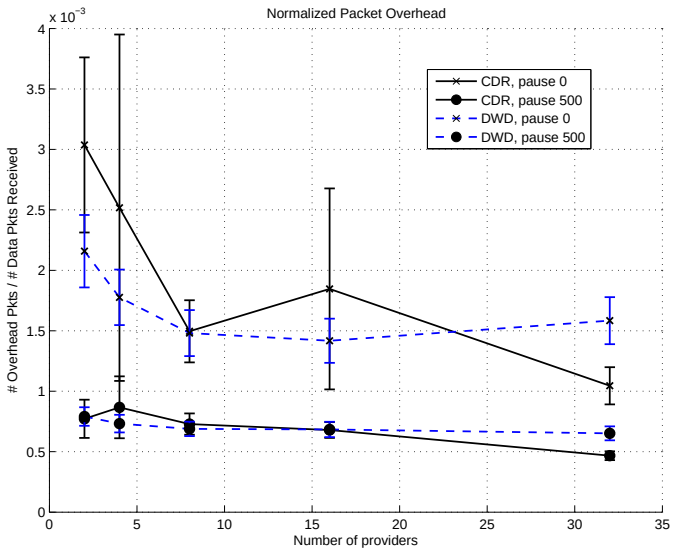

Figure 4.20: Normalized packet overhead for a single requester, multiple providers scenario under different mobility conditions

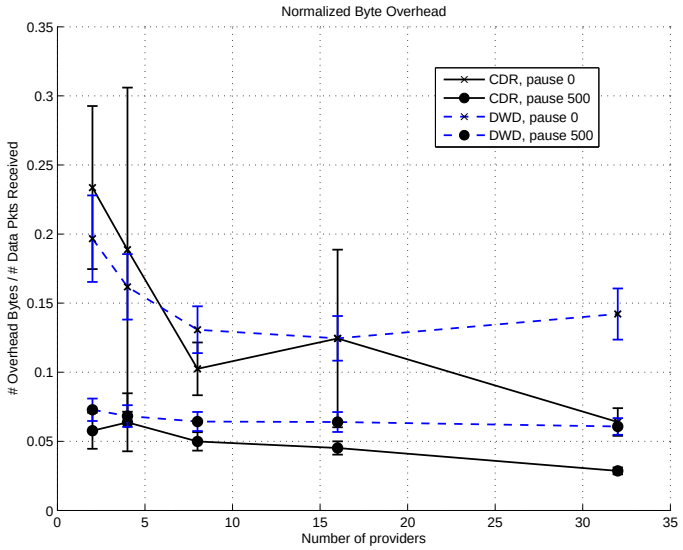

Figure 4.21: Normalized byte overhead for a single requester and multiple providers scenario under different mobility conditions

source. As the number of providers increases, the route length asymptotically reaches unity for CDR, i.e., in the limiting case the resource provider is a one-hop neighbour of the source node.

Figures 4.18 and 4.19 show how the packet delivery ratios for the two protocols compare in the multiple provider scenario. While CDR's packet delivery ratio is much lower than DWD's (see Figure 4.18) under high mobility, CDR again outperforms DWD as far as application packet delivery ratio is concerned (see Figure 4.19). CDR attempts to deliver more packets than DWD, and succeeds more often. The increase in success for CDR in comparison to DWD reflects the fact that CDR's single phase of communication quickly targets locally available resources. As we can see from Figures 4.20 and 4.21, network packet overheads for CDR and DWD are comparable when mobility is low. Under high mobility conditions, CDR generates a larger network overhead, due to frequent discoveries. However, the large variance in CDR's overhead and the dip in the packet delivery ratio need more thorough investigation, before further conclusions can be drawn.

In summary, CDR protocol promotes network locality of resource connections. That is, based on our naïve selection process that chooses the first responding service to connect to, we connect clients to resources that are "nearby" in a network sense (i.e., latency). This makes sense with respect to the applications discussed in Chapter 1; in general these applications implicitly favor a "local" matching resource over a more distant one. The definition of locality and its use within our protocol is discussed in more detail in the next chapter. It is difficult to achieve a similar behaviour in DWD without including additional information within registrations, thereby incurring additional overhead. Examples as shown in Figure 1.3 abound, where the lookup server

cannot tell from its registration list exactly which of the matching resources it should return to a requester.

4.8 Supporting Multiple Traffic Flows

The above sections dealt with how CDR performs when there is a single flow in the network. This section looks at the performance of CDR and DWD under high traffic conditions. More specifically, we investigate the scalability of CDR to simultaneously support multiple independent flows of traffic.

In the experiments conducted to study the scalability of CDR, two independent traffic flows were simulated. Two requesters were placed at random locations in the network. One resource provider existed for each service, each placed at a random location. The sources generated packets at the rate of 20 packets per second. This effectively doubles the traffic in the network, from the single flow scenario. The fallout of this construction was that nodes in the middle of the network (the nodes that forward packets) often ran out of space in their interface queues between the link layer and the MAC layer. The consequences of this phenomenon is evident in the performance of both protocols in high traffic conditions.

Figure 4.22 shows the discovery latency for the two protocols under different traffic conditions. CDR's route discovery latency is much lower than DWD's. In going from one flow to two flows, DWD's route discovery latency has increased five fold. The reason for this is that some route discovery packets get lost due to queue buildup at intermediate nodes in the network. The source cannot detect this until the discovery timer expires at the source node.

Data delivery latency of DWD on the other hand is much lower than

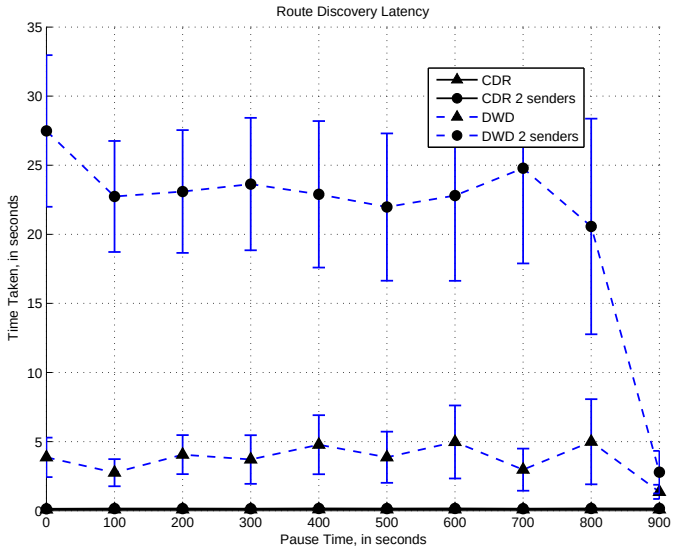

Figure 4.22: Discovery latency for the two requesters, two resource providers scenario

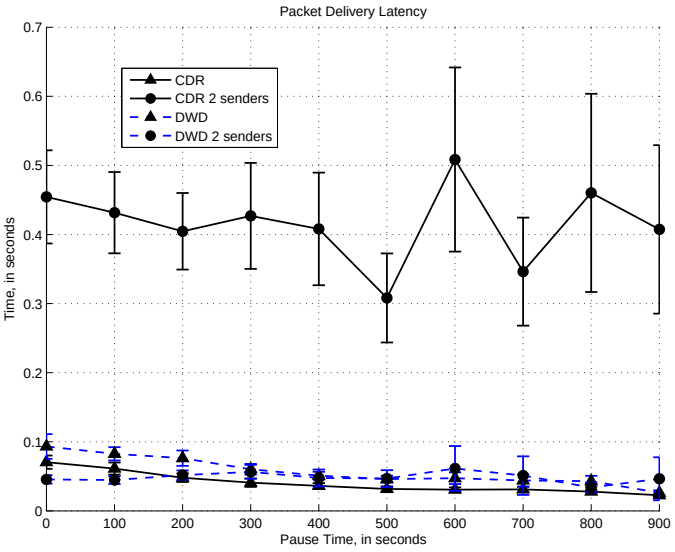

Figure 4.23: Data delivery latency for the two requesters and two resource providers scenario

that of CDR's, as shown in Figure 4.23. The data delivery latency has increased roughly five fold for CDR in going from one flow to two flows. This can be attributed to the loss of packets in the middle of the network.

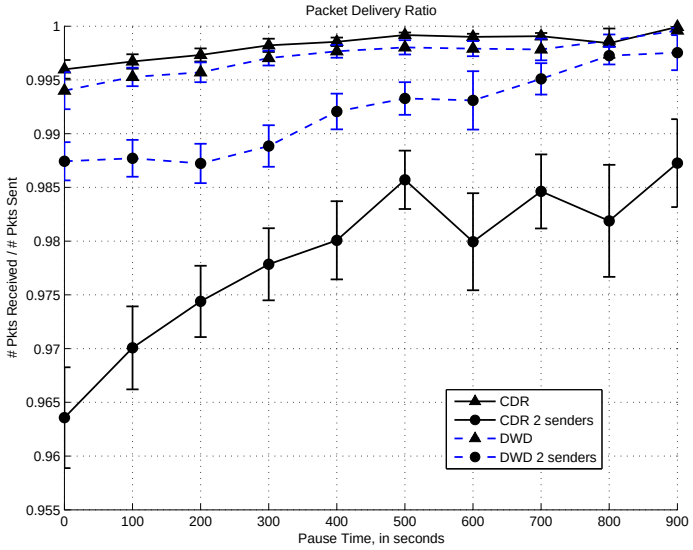

Figure 4.24: Packet delivery ratio for the two requesters and two resource providers scenario

The packet delivery ratio for CDR is much lower than that of DWD under high traffic conditions (see Figure 4.24) for the same reasons as mentioned above. Under these circumstances, in order to ensure some kind of guarantee that the data packets have indeed reached the destination, some connection-oriented semantics need to be employed at the application level. While the application packet delivery ratio (see Figure 4.25) for CDR has still reduced in going from one flow to two flows, it is comparable to DWD's application packet delivery ratio.

Finally, Figures 4.26 and 4.27 show that CDR's performance is affected

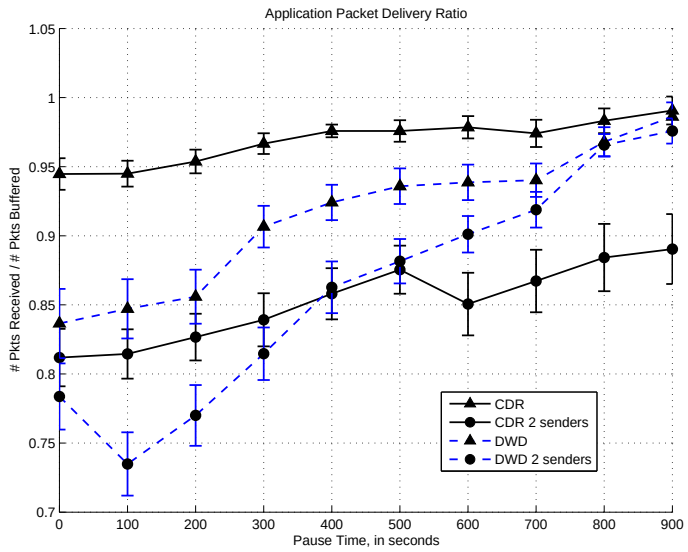

Figure 4.25: Application packet delivery ratio for the two requesters, two resource providers scenario

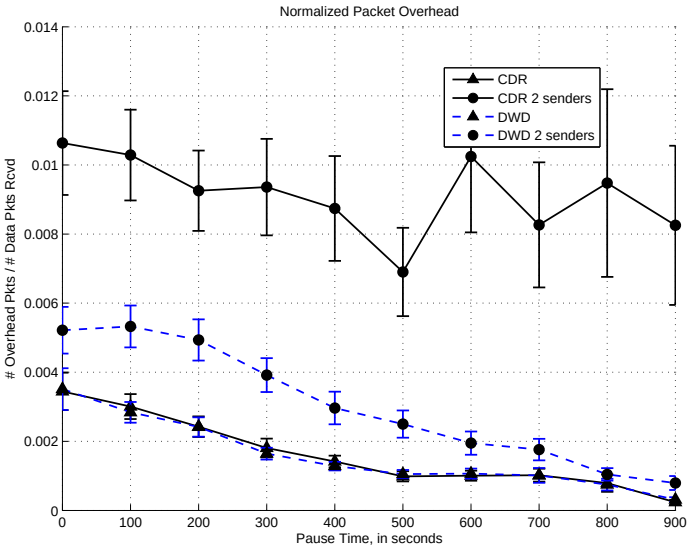

Figure 4.26: Normalized packet overhead for the two requesters, two resource providers scenario

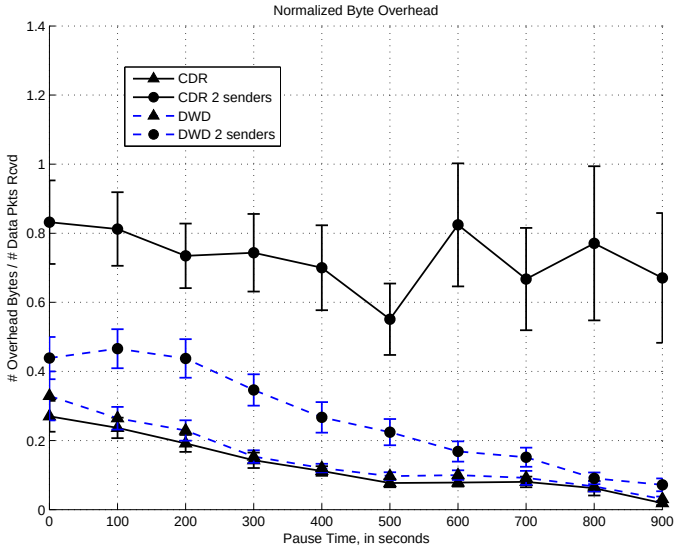

Figure 4.27: Normalized byte overhead for the two requesters, two resource providers scenario

to a greater extent by the loss of packets at the link layer. The delivery latencies and communication overhead are indicative of CDR's inability to perform well under high traffic conditions. This leaves us with the conclusion that CDR needs to employ some optimizations that will detect when the interfaces below it are full, and stop sending packets under such conditions. CDR's current performance leaves room for more evaluation under different data traffic patterns, to determine which traffic patterns CDR is best suited for.

4.9 Analysis

The results described above show that integrating the resource discovery process directly with the route discovery process has the potential to change the way applications use communication in mobile networks. The CDR

protocol shows performance comparable to DWD, which is representative of the two-phase approach. In this section, we examine aspects of the protocols, that, when viewed in conjunction with the above performance characterization, point to CDR as a valuable contribution to the body of communication protocols for mobile ad hoc networks.

4.9.1 Idealized Lookup Server

Our simulation results use an idealized lookup server that demonstrates better performance for DWD than would actually be perceived in a real implementation.

We assumed that the lookup server knew in advance what services would be offered at all nodes in the network. In a mobile ad hoc network, it is infeasible for any centralized authority to have all of this information due to the inherently unpredictable nature of the environment. Instead, service providers must register services they offer and deregister services when they are no longer offered. A common solution (e.g., as used in Jini [13]) is *leasing*, where, when services are registered, they are assigned a lease time, and, to remain available in the network, registrations must be renewed when the lease time expires. The issue of registration becomes even more important when we consider the applications discussed in Chapter 1. Many of these applications search for specific data items based on meta-data. Registering *all* of the data available in the system with a lookup server becomes prohibitively costly. This problem can be addressed using indirection; a node that provides a particular kind of data offers some generic service which can be registered. The problem then becomes one of description—how much information should be provided about a particular data provider in its registration? For example, in the case

of a traffic monitoring application, location should clearly be provided, but at what granularity? The tradeoffs associated with minimizing the overhead of registration and maximizing the quality of matches makes deploying efficient and general lookup services very difficult. In any of these cases, the registration process incurs periodic additional overhead, which, as the number of available services increases will have a dramatic impact on DWD's performance.

In our experiments, the DWD lookup server was well-known. In reality, it is likely that, either every node will have a list of potential lookup servers that it must attempt to contact in succession (as in many peer-to-peer systems), or the network will have to employ a distributed protocol to elect and maintain a set of such servers [2, 33]. This process incurs an additional cost that we are not yet measuring in our evaluation. In addition, we placed the lookup server in the center of our simulation environment and left it immobile. This greatly increases the availability of this service, and in the real world, the responsibility of the lookup service will most likely be delegated to a node that is itself mobile. Again, this can impact the availability of the lookup service and the overhead incurred in contacting it.

4.9.2 Matching Language Independence

Another significant benefit of the CDR protocol is its independence from the particular matching language used. Any matching language can be used; the only requirement is that clients and servers that are paired must speak the same language to be matched. This allows the communication protocol we have presented to be placed underneath existing applications, and, given a proper interface for communication constructs, the protocol can carry resource requests and replies for existing service provision systems.

Chapter 5

Discussion

In the previous chapters, we presented a novel model of communication that holds promise in supporting future mobile applications, particularly those that satisfy the characteristics we highlighted in Chapter 1. This section examines the subsequent steps that must be taken to build on these results to create a deployable, usable, and expressive approach to resource-directed discovery and routing.

5.1 Resource Descriptions and Requests

Our protocol currently utilizes a simple description language based on semi-structured data [1]. This approach is common among well-accepted description languages [5, 10, 13, 20, 21, 44]. In general, semi-structured data approaches relate attributes hierarchically (e.g., locations have addresses which have zip codes). At this stage, we choose not to restrict ourselves to a particular language and have therefore intentionally designed a flexible mechanism that leaves the specification and matching capabilities outside of the protocol mechanics. As we continue with evaluations of our protocol, we expect it to encounter performance degradations due to the fact that service descriptions are of variable lengths, and, in general, are longer than the fixed length addresses traditionally used by routing protocols. In addition, our protocol incurs increased processing time at each intermediate host due to the fact that

we run a more sophisticated matching algorithm than standard routing protocols. Addressing optimality of representation and optimality of matching are essential next steps in ensuring the protocol used is as efficient as possible. Efficient solutions may require the assumption of a particular description language, and the inflexibility of this modification will have to be met with significant performance improvements to justify its inclusion.

Our protocol generates routes only for exact matches between a specification and a resource. A potential optimization might allow a resource discovery process to complete early if a "close enough" match is discovered. Approaches relating to this concern have recently taken advantage of the structural properties of the data specifications. For example, INS [2] explicitly formats descriptions and specifications in name trees. An extension to the model [3] breaks the trees into *strands*, or unique subsequences of paths in the attribute trees. A client's request, originally also formed as a tree, is also broken into strands. The matching algorithm compares the two sets of strands and returns to the client the resource with the highest number of subsequence matches. This style of approach is a first step, but operates under the condition of complete information (i.e., the matching algorithm can return what it knows for sure to be the best match). Incorporating such an approach into a distributed algorithm such as ours requires significant reconceptualization because the protocol must be able to (in a distributed fashion) determine when a resource is a close enough match to create a route without global knowledge of available resources.

5.2 Reactive versus Proactive Approaches

Chapter 1 outlined our rationale for an entirely reactive protocol without advertisement. Briefly, our decision is motivated by the fact that our target applications operate in highly dynamic and data rich environments where advertising *all* of the available data proves too costly in terms of communication overhead. This is in stark contrast to service provision, which operates under the assumption that a widely-used set of services will be desired by multiple applications which therefore benefit from distributed advertisement of the services. Given the potential for success of our resource-directed protocol described in this paper, further extensions may include a limited proactive behaviour based not on the nature of the data or resource but on the nature of the requests for it. That is, once the frequency of requests or number of requesters reaches a certain (adaptive) threshold, it may make sense to proactively distribute data in a limited local region (depending on the extensiveness of the dynamics of the environment). Future work can possibly investigate the feasibility of such modifications with respect to metrics for measuring when to adapt and the degree to which proactive behaviour should be used. This type of adaptive protocol differs from the Zone Routing Protocol (ZRP)'s [22] use of hybrid proactive/reactive behaviour in that their scheme uses only network topology information to adapt, while we promote using network and application context.

5.3 Multiple Route Caching and Updating

By nature of source routing, a source request can generate multiple routes to the same destination. In addition, because we do not use a unique identifier to specify the destination, multiple distinct destinations may satisfy

the request. For now, we simply choose the one with the lowest latency. Additional metrics can be easily incorporated, e.g., relative location or load. In addition, when one route to a destination breaks, our protocol immediately picks up another different route to that destination if it has one cached.

In the same way, if a destination becomes unreachable, but the source heard from two possible destinations, it can automatically switch to the other resource. Due to the nature of source routing, this is simple to accommodate within our protocol. New issues arise, however, because we are possibly dealing with multiple different destinations. Some interactions between a source and a destination, once initiated, may have some long-lived state that impacts future interactions. This state may have to be maintained as the connection switches from one destination to another. For the moment, this concern is ignored in our protocol, and there are many cases when this is acceptable or even desirable. For example, if the data resource is local temperature information, it is desirable, that, as the device moves, a more local resource is selected in preference to an old connection to a more distant resource. On the other hand, if the interaction is a bidding negotiation between a buyer and a seller, automatically switching to a new seller would disrupt any ongoing transactions. Additional protocols can be integrated with CDR for transparently migrating existing state information from one resource to another when acceptable or, in the worst case, ensuring clean and announced disconnection from disappearing resources [24].

Chapter 6

Conclusion

This thesis has presented a novel communication protocol, *Cross-layer Discovery and Routing* (CDR) that alleviates the need for applications in a mobile ad hoc network to contact a well-known name resolver or repository to create dynamic routes among mobile hosts. We first compared existing communication provisions with the clearly stated needs and assumptions of a generic class of dynamic applications, demonstrating a significant mismatch between the two (Chapter 2). As we set out to bridge this gap, we started with the motivation that the combination of a reactive protocol with the source routing paradigm holds the most promise for an adaptive yet responsive and flexible mechanism (Chapter 1). We then presented CDR, providing a formal abstract characterization of the protocol in addition to the common textual description (Chapter 3). To examine the feasibility of incorporating non-fixed length data and resource descriptions into a reactive routing protocol, we performed a simulation analysis of our protocol and compared it with other alternatives (Chapter 4). Finally, we examined the implications of the most fundamentally unique aspects of our protocol and identified areas for enhancements (Chapter 5). The work presented in this thesis provides a necessary and significant first step in supporting real-world dynamic and adaptive applications for emerging mobile ad hoc network scenarios.

Bibliography

[1] S. Abiteboul. Querying semi-structured data. In *Proceedings of the 6th International Conference on Database Theory*, pages 1–18, 1997.

[2] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proceedings of the 17th International Symposium on Operating System Princples*, pages 186–201, 1999.

[3] M. Balazinska, H. Balakrishnan, and D. Karger. INS/Twine: A scalable, peer-to-peer architecture for intentional resource discovery. In *Proceedings of the International Conference on Pervasive Computing*, pages 195–210, 2002.

[4] A. Boukerche. Performance evaluation of routing protocols for ad hoc wireless networks. *Mobile Networks and Applications*, 9(4):333–342, 2004.

[5] T. Bremers-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, 284(5):34–43, 2001.

[6] J. Broch, D.A. Maltz, D.B. Jounson, Y.-C. Hy, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proceedings of the 4th International Conference on Mobile Computing and Networking*, pages 85–97, 1998.

[7] M. Caparuscio, A. Carzaniga, and A. Wolf. Design and evaluation of a support service for mobile, wireless publish/subscribe applications. 29(12):1059–1071, 2003.

[8] T.-W. Chen and M. Gerla. Global state routing: A new routing scheme for ad-hoc wireless networks. In *Proceedings of the IEEE International Communication Conference*, pages 171–175, 1998.

[9] C.-C. Chiang, H.-K. Wu, W. Liu, and M. Gerla. Routing in clustered multihop, mobile wireless networks with fading channel. In *Proceedings of the IEEE Singapore International Conference on Networks*, pages 197–211, 1997.

[10] E. Christensen, F. Gubera, G. Meredith, and S. Weerawarana. Web services description language (wsdl) 1.1. `http://www.w3.org/TR/wsdl`, 2001. Current as of 2005.

[11] S. Czerwinski, B. Zhao, T. Hodes, A. Joseh, and R. Katz. An architecture for a secure service discovery service. In *Proceedings of the 5^{th} International Conference on Mobile Computing and Networking*, pages 24–35, 1999.

[12] S.R. Das, C.E. Perkins, and E.M. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proceedings of the IEEE Conference on Computer Communication*, pages 3–12, 2000.

[13] K. Edwards. *Core JINI*. Prentice Hall, 1999.

[14] P. Engelstad, D.V. Thanh, and G. Egeland. Name resolution in on-demand MANETs and over external IP networks. In *Proceedings of*

the IEEE International Conference on Communication, pages 1024–1032, 2003.

[15] P. Engelstad, D.V. Thanh, and T.E. Jonvik. Name resolution in mobile ad hoc networks. In *Proceedings of the 10th International Conference on Telecommunication*, 2003.

[16] P. Engelstad, Y. Zheng, T. Jonvik, and D. Van Thanh. Service discovery and name resolution architectures for on-demand MANETs. In *Proceedings of the International Workshop on Mobile and Wireless Networks*, pages 736–742, 2003.

[17] K. Fall and K. Varadhan. The *ns* manual. `http://www.isi.edu/nsnam/ns/ns-documentation\.html`. Current as of 2006.

[18] C. Frank and H. Karl. Consistency challenges of service discovery in mobile ad hoc networks. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, pages 105–114, 2004.

[19] T. Fulford-Jones, G.-Y. Wei, and M. Welsh. A portable, low-power, wireless two-lead ekg system. In *Proceedings of the 26th Annual International Conference of the Engineering in Medicine and Biology Society*, 2004.

[20] E. Guttman. Service location protocol: Automatic discovery of IP network services. pages 71–80, July-August 1999.

[21] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service location protocol, version 2. IETF, RFC 2608, June 1999.

[22] Z.J. Haas, M.R. Pearlman, and P. Samar. The zone routing (ZRP) for ad hoc networks. IETF MANET Working Group Internet Draft, July 2002.

[23] R. Harbird, S. Hailes, and C. Mascolo. Adaptive resource discovery for ubiquitous computing. In *Proceedings of the 2nd Workshop on Middleware for Pervasive and Ad Hoc Computing*, pages 155–160, 2004.

[24] Qingfeng Huang, Christine Julien, and Gruia-Catalin Roman. Relying on safe distance to achieve strong partitionable group membership in ad hoc networks. 3(2):192–205, 2004.

[25] IEEE Standards Department. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE standard 802.11-1999, 1999.

[26] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heideman, and F. Silva. Directed diffusion for wireless sensor networking. *IEEE/ACM Transactions on Networking*, 11(1):2–16, February 2003.

[27] A. Iwata, C.-C. Chiang, G. Pei, M. Gerla, and T.-W. Chen. Scalable routing strategies for ad hoc wireless networks. 17(8):1369–1379, 1999.

[28] M. Jiang, J. Li, and Y.C. Tay. Cluster based routing protocol. IETF MANET Working Group Internet Draft, August 1999.

[29] M. Joa-Ng and I.-T. Lu. A peer-to-peer zone-based two-level link state routing for mobile ad hoc networks. 17(8):1415–1425, 1999.

[30] D. Johnson, D. Maltz, and J. Broch. DSR: The dynamic source routing protocol for multi-hop wireless ad hoc networks. *Ad Hoc Networking*, pages 139–172, 2001.

[31] C. Julien and M. Venkataraman. Resource directed discovery and routing in mobile ad hoc networks. Technical Report TR-UTEDGE-2005-003, The University of Texas at Austin, 2003.

[32] R. Koodli and C.E. Perkins. Service discovery in on-demand ad hoc networks. Internet Draft, October 2002.

[33] U. Kozat and L. Tassiulas. Service discovery in ad hoc networks: An overall perspective on architectural choices and network layer support issues. *Ad Hoc Mobile Networks*, 2:23–44, 2004.

[34] S.J. Lee, W. Su, and M. Gerla. On-demand multicast routing protocol in multihop wireless mobile networks. *Mobile Networks and Applications*, 7(6):441–453, 2002.

[35] N.A. Lynch and M.R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[36] M. Matthes, F. Apitzsch, M. Lauer, and O. Drobnik. Application-oriented routing for mobile ad hoc networks. In *Proceedings of the European Wireless Conference*, 2004.

[37] S. Murthy and J.J Garcia-Luna-Aceves. An efficient routing protocol for wireless networks. *Mobile Networks and Applications, Special Issue on Routing in Mobile Communication Networks*, 1(2):183–197, 1996.

[38] V.D. Park and M.S. Corson. A highly adaptive distributed routing algorithm for mobile wireless networks. In *Proceedings of IEEE Conference on Computer Communication*, pages 1405–1413, 1997.

[39] C.E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance vector routing (DSDV) for mobile computers. In *Proceedings of the Conference on Communication, Architectures, Protocols, and Applications*, pages 234–244, 1994.

[40] C.E. Perkins and E.M. Royer. Ad hoc on-demand distance vector routing. In *Proceedings of the 2nd Workshop on Mobile Computer Systems and Applications*, pages 90–100, 1999.

[41] V. Ramasubramanian, Z. J. Haas, and E. G. Sirer. SHARP: A hybrid adaptive routing protocol for mobile ad hoc networks. In *Proceedings of the 4th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, pages 303–314, June 2003.

[42] G.-C. Roman, C. Julien, and Q. Huang. Network abstractions for context-aware mobile computing. In *Proceedings of the 24th International Conference on Software Engineering*, pages 363–373, 2002.

[43] M. Roussopoulos, P. Maniatas, E. Swierk, K. Lai, G. Appenzeller, and M. Baker. Person-level routing in the mobile people architecture. In *Proceedings of the 2nd USENIX Symposium on Internet Technology and Systems*, 1999.

[44] Salutation Consortium. The salutation consortium webpage. `http://www.salutation.org`, 2004.

[45] E. Yoneki and J. Bacon. An adaptive approach to content-based subscription in mobile ad hoc networks. In *Proceedings of the 1st International Workshop on Mobile Peer-to-Peer Computing*, pages 92–97, 2004.

[46] H. Zhou and S. Singh. Content based multicast (CBM) in ad hoc networks. In *Proceedings of the 1st International Symposium on Mobile Ad Hoc Networking and Computing*, pages 51–60, 2000.

Vita

Meenakshi Venkataraman was born in Chennai, the capital of Tamil Nadu, India on January 17^{th} 1982. She did all her schooling in Kendriya Vidyalaya under the Central Board of Secondary Education. She went on to obtain her Bachelor's degree in Electronics and Communications Engineering from the University of Madras in 2003, during which period developed an interest for communication and networking systems. Her pursuit for knowledge in wireless communications brought her to the University of Texas at Austin, where she did her graduate study. In the summer of 2004 she worked as an intern at Voxpath Networks, Inc., where she learned about service based telephone networks. She started work under Dr. Christine Julien as a research student in December 2004, whom she assisted in formulating a cross-layer resource discovery protocol for Mobile Ad Hoc Networks. This thesis is a compilation of the work she did under Dr. Julien.

She is presently working as a software engineer at Spirent Communications, Inc., Fort Worth, Texas.

Permanent address: 3737, Hidden Spring Drive, Apt # 406
Fort Worth, TX — 76109

This thesis was typeset with LaTeX[†] by the author.

[†]LaTeX is a document preparation system developed by Leslie Lamport as a special version of Donald Knuth's TeX Program.