Resource Directed Discovery and Routing in Mobile Ad Hoc Networks

Christine Julien
Meenakshi Venkataraman

TR-UTEDGE-2005-001
Resource-Directed Discovery and Routing in Mobile Ad Hoc Networks

Christine Julien and Meenakshi Venkataraman
The University of Texas at Austin
Department of Electrical and Computer Engineering
Austin, TX 78712
Email: c.julien@mail.utexas.edu, meens@ece.utexas.edu

Abstract—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.

I. 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. The lack of a static infrastructure requires the mobile hosts themselves 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, some applications may elect 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. Today’s construction sites are 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. Advances in user interfaces have enabled cars that communicate sophisticated information to drivers, 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:

- the network topology is dynamic, i.e., network links are neither constant nor predictable;
- the application environment is data rich;
- 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; and
- different devices and users require different data according to their instantaneous tasks and environments.

Much work on supporting applications in mobile ad hoc networks builds routing protocols that maintain communication between senders and receivers. As the topology of the network changes, the protocols adjust routes to maintain end-to-end connectivity. These protocols are motivated by the desire to support the end-to-end communication common in Internet applications instead of emerging mobile applications like those characterized above. 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 of the other hosts in the network with which it desires to communicate, which 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. These two-phase approaches have several drawbacks:

- Electing and maintaining a stable server set in an ad hoc network incurs a significant overhead in the highly dynamic scenarios we are targeting [1].
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 numbers of control messages [2].

In a purely ad hoc network, the servers may themselves be mobile and dynamic, requiring an initial discovery protocol targeted at finding the resolvers.

In this paper, we introduce a highly reactive communication protocol that removes the drawbacks of name resolution. The novel contributions of this work are as follows. We first identify a set of assumptions made by existing mobile ad hoc network communication mechanisms that are limiting to the protocols’ applicability to supporting real-world applications. Second, we present a protocol for communication that overcomes these assumptions, and we include an abstract model of the protocol’s behavior, an aspect often overlooked in protocol description. Finally, and most importantly, we present a performance evaluation that serves to not only compare our protocol to other alternatives but to demonstrate the feasibility of incorporating non-fixed length addressing into a reactive mobile ad hoc routing protocol.

II. RELATED WORK

Routing protocols for mobile ad hoc networks can generally be divided into two categories: proactive and reactive. Proactive protocols [3], [4], [5], [6], [7], [8] maintain routes between each pair of hosts in the network. Reactive, or on-demand protocols [9], [10], [11], [12], 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 [13], [14], [15]. 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 [16] is a hybrid that leverages proactive behavior within a local “zone” surrounding a node and switches to reactive behavior outside of that zone.

These routing protocols require the application to provide the unique address of the destination to create a route. This is analogous to routing in traditional wired networks, where applications take advantage of the static network 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.

Recent work has focused on building DNS equivalents for mobile ad hoc networks [17], [18] that use reactive routing and multicast to create a name resolution phase that occurs before the routing phase. Service discovery approaches, e.g., [1], [19], [20], [21], [22], [23], [24], [25], 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 infrastructure mobile networks [26] and even in mobile ad hoc networks [27]. 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 behavior 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 behavior of applications.

Our work is not the first to propose application-level or content-directed communication. Content Based Multicast (CBM) [28] pushes messages to receivers based on the message’s content. These protocols are complementary to ours in that they support applications which require strictly push interactions. Network Abstractions [29] 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 [30] extends TORA [12] to create a service provision architecture. This approach maintains a combination of proactive and reactive behavior, 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 [31] 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 investigates integrates resource discovery and route construction in mobile ad hoc networks [2], providing an implementation of the architecture requirements first elucidated in [32]. This work enhances AODV [10] 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 [33] 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 networks commonly coordinate to perform a specific sensing task and can take advantage of this cooperation to aggregate messages destined for a sink node. We target a drastically different communication environment, where the traffic flows are neither predictable,
persistent, nor deterministic and many nodes serve as “sinks.” In addition, we focus on the feasibility of non-fixed length addressing in a dynamic scheme, where the presented directed diffusion implementation makes strong assumptions regarding an agreed-upon naming representation. None of these existing protocols includes a flexible naming scheme or evaluates the impact of routing with non-fixed length addressing on the overhead of communication in highly mobile environments.

III. MOTIVATION AND GENERAL APPROACH

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 [34] 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 descriptive languages.

The network overhead and message delivery latency must be of the utmost concern because interactions are opportunistic and hosts must take advantage of communication partners while they are connected. For this reason, we are motivated to avoid an approach which requires several phases of communication over the network. The alternatives are pictorially compared in Fig. 1 which shows the network traffic generated by a combined discovery and routing protocol on the left and a two-phase approach on the right. Service discovery approaches, while promising from an abstract perspective, 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 Fig. 2).

Our approach directly addresses the assumptions of existing approaches, and constantly concerns itself with the performance implications of our design decisions. In the next section, we provide the details of a protocol that combines resource resolution and routing into a single step. Our 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 behavior, we introduce a level of indirection into a source routing protocol (e.g., Dynamic Source Routing (DSR) [11]). We selected source routing 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 [14], [15]. The protocol presented is straightforward and as yet unoptimized. This is due to the fact that the 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 Section V, we evaluate this baseline protocol with respect to several routing metrics and in comparison 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 [2], [35].

IV. A Protocol for Cross-Layer Discovery

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.

A. 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 [21], [22], [36], [37], [38]. In general, successful, decentralized approaches use semi-structured data [39] 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 Section V. As such, our protocol must function independently of how matching decisions are made. Therefore, for now, we forgo 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.

B. 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 not only the standard textual description, but also an abstract model of the behavior. Such a formalization of proposed communication protocols is generally unavailable in the literature. We view the elucidation of this model as essential to clearly stating the behavior of a protocol and the assumptions on which it relies. This eases understanding of the protocol, makes explicit the state maintenance needs at each host, and guides the careful design of our implementation.

CDR utilizes four packet types, described in Fig. 3. Resource Discovery begins when a source host requires a service or data that it does not itself have, thereby necessitating the discovery of another host. To participate in CDR, each host stores several pieces of state information, shown in Fig. 4, relating to its previous and pending requests, existing routes, and a minimal amount of information stored about requests made by other sources. The figure shows the state held by a single host; every host has its own set of these variables.

C. Application Interaction

To use the protocol, an application sends packets to destinations designated by restrictions on semi-structured data. Fig. 5 shows the send action triggered by the application. We use I/O Automaton notation [40] 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., send ApplicationPacket_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
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.

Fig. 3. CDR Packet Types

Neighbors
the set of neighboring 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. When the
preferred route for a description fails, this cache is used to select a new route without performing another
resource discovery.

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
numbers application packets sent by this host; may be used in resending packets that experience errors.

SentPackets(packet_num)
application packets sent by this host; used in resending packets that experience errors.

ResourceTable
the semi-structured descriptions of this source’s resources, matched against other hosts’ requests

Resends
application packets queued to be resent due to transmission failures.

Fig. 4. CDR State Information

<table>
<thead>
<tr>
<th>SENDAPPLICATIONPACKET_A(P)</th>
<th>TRANSMITAPPLICATIONPACKET_A(P)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Precondition:</strong> PendingPacket = NULL</td>
<td>Precondition: PendingPacket = P</td>
</tr>
<tr>
<td><strong>Effect:</strong> if RouteCache(P.spec) = $\emptyset$ then</td>
<td>RouteCache(P.spec) $\neq$ $\emptyset$</td>
</tr>
<tr>
<td>Send ResourceDiscovery(RD) to Neighbors</td>
<td>Route := RouteCache(PendingPacket.spec).head</td>
</tr>
<tr>
<td></td>
<td>p.route_record := route</td>
</tr>
<tr>
<td></td>
<td>p.packet_num := ++packet_num</td>
</tr>
<tr>
<td></td>
<td>Send ApplicationPacket(P)</td>
</tr>
<tr>
<td></td>
<td>to p.route_record.successor(A)</td>
</tr>
<tr>
<td></td>
<td>SentPackets(packet_num) := P</td>
</tr>
<tr>
<td></td>
<td>PendingPacket := NULL</td>
</tr>
</tbody>
</table>

Fig. 5. Sending an Application Packet

of actions that ultimately triggers RESOURCEDISCOVERYRECEIVED (Fig. 7) on each neighbor.

As Fig. 5 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 neighbor (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 Fig. 5.

To send the application packet, the host selects the first available route for the specification ((RouteCache(p.spec)).head). Each packet has a unique number (packet_num) which we use when errors occur. Again, we use the simplified notation “send ApplicationPacket(P) to p.route_record.successor(A)” 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, in case of a transmission failure.

Fig. 6 shows APPLICATIONPACKETRECEIVED. If the host is the intended destination, the packet is delivered to the application. If this host is not the intended recipient the packet is propagated by selecting the next host in the route record (i.e., route_record.successor(A)) and triggering that host’s APPLICATIONPACKETRECEIVED(P) action.

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.

D. Resource Discovery

The above process triggers RESOURCE DISCOVERY RECEIVED, shown in Fig. 7, on each of the source’s neighbors.

```
APPLICATION_PACKET_RECEIVED_A(P)
Effect:
if P.route_record.tail = A then
deliver application packet
else
if P.route_record.successor(A) ∈ neighbors 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
```

Fig. 6. Propagating an Application Packet

A receiver of an RD first checks the `route_record` to ensure there are no routing loops. The 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 and continues to propagate the resource discovery.

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 ROUTE_REPLY RECEIVED, shown in Fig. 8 on its predecessor in the `route_record`.

```
ROUTE_REPLY_RECEIVED_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.successor(A)
end
```

Fig. 8. Propagating a Route Reply Packet

Unless the host is the source, ROUTE_REPLY RECEIVED 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 TRANSMIT APPLICATION PACKET shown in Fig. 5.

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 is 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 Section VI addresses this design decision and using context properties and context-sensitive requirements of the application to select the best path according to different metrics.

E. 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 Fig. 6 we see how the RE is generated by the host that detects the broken link. This host initiates the propagation by triggering ROUTE ERROR RECEIVED on the previous host in the route. Fig. 9 shows how this packet is propagated. 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.

A host receiving an RE deletes any routes it stores that 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.

To complete our protocol’s specification, we must also ensure that these packets are retransmitted. To this purpose, we add the action shown in Fig. 10, which is the same as SEND APPLICATION PACKET in Fig. 5 except that the packet comes from Resends instead of directly from the application. This new action does not guarantee fairness between applications running on the host that has requested a route for the same specification (or for a specification that subsumes the requested one). We have not yet evaluated our protocol under this optimization. However, the host detecting a link failure can check 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.

V. ANALYSIS AND EVALUATION

In this section we provide 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 network simulator to generate these initial results.

A. Simulation Settings

We used network scenarios similar to those found in [13], [14], [15]. Specifically, we utilized node mobility patterns based on random waypoint mobility [14] and distributed 51 nodes in a rectangular field of size 1500m × 300m. The speed of the nodes was uniformly distributed between 0 and 20m/s, with the exception of one node, which is stationary in the center of the field. 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 each plotted point indicates an average over 20 samples. Unless otherwise specified, the traffic between sources and destinations for all protocols was generated at the rate of 10 packets per second. All data packets were of size 512 bytes.

B. 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.

- **Packet delivery ratio** is a standard measure of the fraction of the sent packets that are actually delivered.
- **Route discovery latency** measures how long it takes to discover the intended destination.
- **Data delivery latency** demonstrates, on average, how long it took to deliver a data packet from an application on the source to an application on the destination.
- **Normalized packet overhead** counts the number of control packets sent in the network for each data packet delivered. This metric counts each individual packet separately (e.g., if a node forwards a route request, the forwarded packet is counted in addition to the received packet).
- **Normalized byte overhead** measures the number of bytes of control messages sent for each data packet delivered.
### Description

<table>
<thead>
<tr>
<th>Description</th>
<th>Size (bytes)</th>
<th>Label</th>
</tr>
</thead>
<tbody>
<tr>
<td>(traffic=service)</td>
<td>17</td>
<td>size-1</td>
</tr>
<tr>
<td>(traffic=service(city=Austin))</td>
<td>30</td>
<td>size-2</td>
</tr>
<tr>
<td>(traffic=service(city=Austin(street=Parmer_Lane)))</td>
<td>50</td>
<td>size-3</td>
</tr>
<tr>
<td>(traffic=service(city=Austin(street=Parmer_Lane))(age&lt;10_minutes))</td>
<td>66</td>
<td>size-4</td>
</tr>
<tr>
<td>(traffic=service(city=Austin(street=Parmer_Lane))(age&lt;10_minutes)(incident=collision))</td>
<td>86</td>
<td>size-5</td>
</tr>
</tbody>
</table>

**Fig. 11.** Resource descriptions used.

### C. Protocols

In addition to providing results for our protocol, we provide comparisons to two other protocols.

- **CDR.** The CDR protocol performs as described in Section IV with the exception that multiple data packets can be queued at the source and a time-to-live is used to prevent route requests from propagating forever.

- **DSR.** As a baseline for comparison, we provide results for an omniscient DSR protocol. For this, we used the DSR implementation within ns-2. We refer to the protocol as omniscient because it is assumed that the application knows the ids of the destinations with which it chooses to communicate *a priori*. The results for this protocol serve as an ideal bound for CDR.

- **DWD.** The final protocol we used for comparison was DSR with discovery (DWD). We implemented a simple lookup scheme on the stationary center node. 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 node (or nodes) in the network that provide that service. We assumed that the lookup node had 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. We also assume that a source node looking for a particular service already knew the id of the lookup server and did not need to discover it. DSR was used to create routes between sources and the lookup server and between sources and destinations.

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. For DSR, addresses are of a fixed length, so the choice of destination has no effect. To determine how increasingly complex resource or data requests affect performance, we measure each of our above metrics using increasingly complex resource requests. Fig. 11 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 Fig. 11), but in the implementation, any other description representation can be easily swapped in for this simple language.

### D. Basic Comparison

The following results compare CDR with DWD and DSR in a basic sense. For the CDR and DWD results in this set of graphs, we used the description size-1 (see Fig. 11). Nodes selected as service providers offer the service for the lifetime of the simulation. In Section V-F, we examine the impact of a dynamically changing set of services on our performance metrics. Fig. 12 shows the delivery ratios for the three protocols. At low mobility levels (e.g., pause time of 400 or greater), the three protocols are very similar. In high mobility, CDR seems to perform significantly better than the two DSR based protocols. This is artificial and due to the fact that CDR is not optimized, while DSR is. These optimizations help DSR decrease its routing overhead and bolster performance especially under heavy traffic loads. Exploring potential optimizations and their impacts on CDR is a point for future work. The message from Fig. 12 is that using application information in routing requests does not negatively affect the number of packets that are successfully delivered.

The data delivery latency as shown in Fig. 13 measures the time between an application sending a packet and a satisfactory destination receiving the packet. In cases where
the sending triggered route discovery, the time for route discovery is included. CDR remains on par with the two comparison protocols except in cases of high mobility, where CDR seemingly outperforms DSR and DWD. The real difference among these protocols will become apparent under a version of CDR optimized to reduce overhead. This implementation of this optimization is complicated by the nature of CDR, as discussed in Section VI.

Fig. 14 looks just at the portion of latency for route discovery. It is important to note that this chart measures the average, for each time discovery was triggered, to send a route request and receive a route reply. For DWD, this includes the time to contact the discovery server if necessary. While CDR and DSR are comparable, the measurements for route discovery latency for DWD contain some outliers, especially in scenarios for low mobility. These cases represent scenarios when the discovery service was out of contact, and therefore significantly delayed discovery of a route to a satisfactory destination. This does not appear at pause time 900 seconds because the nodes were effectively stationary. If a route discovery was initiated, and not immediately successful, it was highly unlikely to be later successful.

Figs. 15 and 16 measure the protocols’ overhead normalized to the number of delivered packets. We expect that the number of control packets sent for DSR and CDR to be about the same, which is not the case in the picture. This is due to the fact that DSR contains optimizations for minimizing the number of control packets, and CDR does not. As we optimize
our protocol (see the discussions in Section VI), we expect to improve our relative performance. The trends for DWD and CDR are similar and demonstrate that the added benefits of completely decentralized resource discovery may outweigh the overhead cost, especially when one considers the added overhead of DWD not considered here (see the discussion in Section V-G).

**E. Impact of Variable Length Addresses**

With respect to increasing the size of resource requests, we are most concerned with the impact on the control overhead of the protocol. Fig. 17 shows the routing overhead in bytes for increasing sizes of resource descriptions for both CDR and DWD. As the graph shows, increasing the size of the description has some impact, but the added semantic information in the description (see Fig. 11) may well be worth this cost to the application. CDR still has slightly more overhead than DWD, but with optimizations of CDR and measurements of complete overhead for DWD, we expect this gap to lessen or completely switch in CDR’s favor. Measurements for the other metrics show no statistically significant variation from one description size to another for either CDR or DWD.

**F. Impact of Service Timeouts**

Another variable that can impact the performance of a discovery-based protocol is the dynamics of the services and data. We now consider the case when the each data provider is available only for 60 seconds. In the simulations, we always ensure that every resource or data request can be satisfied by at least one service provider in the network (though since our scenarios are randomly generated, we do not guarantee that the requester and the service provider are connected). Such scenarios are more typical of dynamic environments than those where a client will stay connected to the same service provider forever and that service providers will continue to be available to provide the service forever. The trends and relative measurements for these simulations were the same for the five differently sized resource descriptions. For that reason, this section provides only results for size-5 (see Fig. 11) which is the largest description we created. For the first three metrics, the trends and values were very similar to the results without timeouts. Because a resource is always available, the data delivery ratio remains high (above 97% for DWD and above 99% for CDR). The route discovery latency for DWD remains significantly higher than for CDR (due to the need to contact the discovery server) and the data delivery latency for DWD is significantly higher at high mobility (low pause times) but pulls closer to CDR at low mobility (high pause times).

Figs. 18 and 19 show the relative normalized overhead in
packets and in bytes for both DWD and CDR. These scenarios show that, in terms of increased overhead, the dynamic nature of resources more negatively affects DWD than CDR (as the CDR curve pulls closer to the DWD curve when compared with Figs. 15 and 16). This single scenario represents only a moderate amount of service dynamics, and we believe it likely that CDR will adapt even better when the services are more dynamic. Future additional measurements will allow us to confirm this belief.

G. 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.

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. It 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 [22]) 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 Section I. 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 [1], [24]. 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 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.

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.

3) Locality of Resource Responses: The CDR protocol promotes network locality of resource connections. That is, based on our naive 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 Section I; 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 section. It is difficult to achieve a similar behavior in DWD without including additional information within registrations, thereby incurring additional overhead. Examples as shown in Fig. 2 abound, where the lookup server cannot tell from its registration list exactly which of the matching resources it should return to a requester.

VI. DISCUSSION AND FUTURE WORK

In the previous sections, we presented a novel model of communication that holds promise in supporting future mobile applications, particularly those that satisfy the characteristics we highlighted in Section I. 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.

4) Resource Descriptions and Requests: Our protocol currently utilizes a simple description language based on semi-structured data [39]. This approach is common among well-accepted description languages [22], [37], [36], [20], [21], [38]. 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. 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 [24] explicitly formats descriptions and specifications in name trees. An extension to the model [25] 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) Reactive versus Proactive Approaches: Section III 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 behavior 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 will investigate the feasibility of such modifications with respect to metrics for measuring when to adapt and the degree to which proactive behavior should be used. This type of adaptive protocol differs from the Zone Routing Protocol (ZRP)’s [16] use of hybrid proactive/reactive behavior in that their scheme uses only network topology information to adapt, while we promote using network and application context.

6) 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. 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 [41].

VII. Conclusion

This paper 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 (Section II). 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 (Section III). We then presented CDR, providing a formal abstract characterization of the protocol in addition to the common textual description (Section IV). 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 (Section V). Finally, we examined the implications of the most fundamentally unique aspects of our protocol and identified areas for enhancements (Section VI). The work presented in this paper provides a necessary and significant first step in supporting real-world dynamic and adaptive applications for emerging mobile ad hoc network scenarios.

Acknowledgments

The authors would like to thank Amal Banerjee for his contributions to this work.

References
