The Need for Cross-Layer Service Discovery in MANETs

Alex Varshavsky, Bradley Reid and Eyal de Lara
walex,brad,delara@cs.toronto.edu
Department of Computer Science
University of Toronto

Abstract

Service selection has a crucial effect on wireless multi-hop ad hoc network (MANET) performance. Because of the broadcast nature of wireless transmission, good service selection groups clients with nearby service providers, localizing communication, which in turn reduces inter-node interference and allows for multiple concurrent transmissions in different parts of the network. Less optimal service selection spreads traffic over the network, increasing interference and reducing overall network throughput.

This paper presents a novel cross-layer architecture for service discovery, selection and rediscovery in MANETs that closely integrates service discovery with ad hoc routing mechanisms. We have analytically showed superiority of the cross-layer approach over traditional approaches in terms of message overhead. Moreover, extensive simulation results for DSR and DSDV (two representative MANET protocols) show that the cross-layer approach achieves network throughput up to five times larger than a traditional application-layer implementation. By leveraging existing routing protocol traffic, the cross-layer architecture allows clients to re-evaluate their service selection, and switch to a better server to offset the effects of changes in network topology.

1 Introduction

A multi-hop mobile ad hoc network (MANET) consists of a group of mobile wireless nodes that self-configure to communicate information beyond the transmission range of individual nodes by routing packets over intermediate nodes [23, 29, 30].

MANETS have been proposed for disaster relief operations, police and military applications, and other situations where there is no deployed communication infrastructure or the existing infrastructure is not available. We expect that one of the main uses of ad hoc networks will be for accessing services. Examples of services include: a long-range radio that allows communication between a team of emergency personnel on the field and commanders overseeing the operation from the command center; maps showing the physical location of other team members; large libraries with detailed instructions and procedures; and processing capacity to perform CPU-intensive operations such as automatic text or voice recognition and translation.

When a service is offered by multiple nodes in an ad hoc network, the specific matching between client nodes and service providers, typically referred to as *service selection*, has a crucial effect on the performance of the wireless multi-hop ad hoc network. Because of the broadcast nature of wireless transmission, the communication pattern of nodes in the network affect the number of concurrent wireless transmissions that the network can sustain, and therefore the achievable overall network throughput. Good service selection localizes communication, which in turn reduces inter-node interference and allows for multiple concurrent transmissions in different parts of the network. Less optimal service selection spreads traffic over the network, increasing interference and reducing overall network throughput. Over time, changes in network topology degrade the optimality of service selection requiring clients to continuously re-evaluate their choice of a server, a process we refer to as *reselection*; and even actively probe the network for availability of new service providers, a process we refer to as *rediscovery*.

Unfortunately, existing service discovery mechanisms are not appropriate for MANETs. Traditional service discovery mechanisms [1–3, 13, 18, 20] have limited knowledge of network topology and assume a mostly-static environment with infrequent network topology changes. In contrast, frequent topology changes are the norm in MANETs, and good service selection is highly dependent on up-to-date knowledge of the network topology.

This paper presents a novel architecture for service discovery in MANETs. Central to the architecture is a cross-layer optimization [11, 31] that closely integrates service discovery functionality with the routing mechanisms of MANETs. The cross-layer service discovery architecture increases overall network throughput by leveraging existing routing traffic to reduce service discovery cost, optimize service selection, and reduce the cost for tracking and reacting to changes in network topology.

We make two significant contributions to the state-of-the-art. First, we provide analytical and experimental proof that the cross-layer approach to service discovery is superior to traditional application-layer approaches that preserve the modularity of the networking stack. Second, we identify service rediscovery and reselection as vital components of service discovery in MANETs, and formalize the space of possible rediscovery and reselection policies. While the idea of using routing information to improve the efficiency of service discovery in MANETs [5, 8, 11, 12, 26, 27, 36] is not new, this is the first study that quantifies the performance advantages of cross-layer integration, demonstrates the dramatic effect that service selection has on network throughput, and identifies efficient service rediscovery and reselection as fundamental requirements for MANETs.

Extensive simulation-based experiments with two cross-layer prototypes based on the DSR [23] and DSDV [29] routing protocols show that the cross-layer implementations consistently outperform an application-layer service discovery implementation that closely models the Service Location Protocol (SLP) [20]. In a mobile environment, the cross-layer implementations achieve up to 5 times higher network throughput than standard SLP and 3.7 times higher throughput than an extended variant of SLP that tries to determine the closest server by sending *ping* messages. These results show that in order to localize communication, service selection in ad hoc networks needs to be based on accurate knowledge of the underlying network topology. Furthermore, we show that due to interference from the underlying routing traffic, timing-based mechanisms for service selection, such as pinging, are highly inaccurate and fail to localize communication.

The rest of this paper is organized as follows. Section 2 discusses service discovery in MANETs, motivates the need for a cross-layer approach for service discovery and presents an analytical upper bound to the cost of service discovery. Section 3 describes a novel cross-layer architecture for service discovery in MANET. Section 4 presents experimental results. Finally, Section 5 describes related work, and Section 6 concludes the paper and discusses opportunities for future research.

2 Service Discovery

We consider a *service* to be a resource (either hardware or software) that is accessible through the network and is capable of generating or receiving data. We refer to nodes that host services as *service providers* or just *servers*, and to nodes that run the *client applications* that access services as *clients*.

Service discovery is the process of finding a mapping from a service description (e.g., “long-range radio”) to a service location, which usually consists of a server address and a port number on which the service “listens” to the incoming requests (e.g., 172.16.1.1:80). Because there can be multiple instances of the same service in the network and because a service description may match multiple similar services, the service discovery process can result in multiple mappings. A client can then choose a specific server among the available mappings based on a myriad of factors including service-specific properties [4] (e.g., select the least-loaded printer) and network-layer parameters [23, 30] (e.g., network latency, hop-count, bandwidth). In this paper, however, we follow the trend taken by the majority of routing protocols [23, 29, 30] and define *best server* as the one that is deemed to be closest to the client, either because it has the shortest path length or because it has the smallest round trip time. In Section 4, we show that this practice improves overall network throughput and consequentially maximizes performance.

Service discovery requires that servers and clients describe services using a common *service description language*. For example, the Service Location Protocol [20] uses service descriptions that consist of a service type (e.g., “printer”) followed by service attributes (e.g., “color=true,type=laser”). Jini [2], on the other hand, uses the Java class hierarchy to achieve this goal. Other architectures define their own proprietary formats [4], which are frequently based on XML [22]. Because the concepts discussed in this paper are independent of the choice of service description language, we do not consider this issue further.

In the rest of this section, we compare two possible implementations of service discover on MANETs: a traditional application-layer implementation which preserves the modularity of the network stack, and a novel cross-layer implementation that exploits existing routing traffic to perform service discovery. For each implementation, we consider the network cost it incurs in terms of the number of messages it generates, and provide analytical upper bounds on the cost of performing service discovery.

2.1 Application-Layer Service Discovery

Service discovery can be implemented at the application-layer based on both centralized [2, 38] and distributed [20, 22] architectures. In centralized implementations, service providers register their services at directory servers, and client nodes unicast their service discovery queries to the directory server. Distributed implementations, in contrast, rely on broadcast or multicast of server-initiated service advertisements or client-initiated service discovery queries. In the former, clients passively listen for advertisements that are periodically broadcasted by the service providers. In the latter, clients broadcast their service discovery queries, which are answered by unicast responses from services that match the client’s query.

Both approaches have advantages and disadvantages when deployed in MANETs. Centralized approaches rely mostly on unicast messages, which can reduce network traffic and service discovery latency as clients only need to communicate with the directory server. On the other hand, centralized approaches require manual configuration or the use of election algorithms that incur significant communication cost. Furthermore, keeping the directory server up to date as servers join or leave the network may require significant traffic. Finally, network partitions may preempt service discovery when the directory server is not reachable, even in cases where a client and server could communicate with each other. Distributed approaches to service discovery do not require manual configuration and can operate even if the network is partitioned. On the other hand, distributed approaches rely on broadcasts which may result in excessive network overhead.

When multiple mappings exist for a service description, the client chooses the server that has the shortest path length or smallest round-trip-time. Because network topology information is not readily available in application-layer service discovery implementations, clients have to do extra processing to obtain it. One of the most common application-layer techniques for obtaining network topology information is *pinging*. Pinging determines the round-trip-time to each service provider by sending a short message to each server and measuring the time it takes for the responses to come back. The client can then select the server which has the lowest response time.

It is interesting to note, that in distributed application-layer service discovery implementations that use client-initiated broadcasts for service discovery, a client can find the approximate round-trip-time to all servers by broadcasting a service discovery query and keeping track of the arrival times of unicast responses sent by services that match the query.

Node mobility and the arrival and departure of nodes that host services degrade the optimality of initial server selection. To overcome this problem, clients need to constantly reevaluate their choice of service provider and switch to a better service provider if one is available. As we show in Section 4, rediscovery has a paramount effect on network throughput.

Rediscovery can be either server or client-driven. In server-driven rediscovery [22], service providers periodically broadcast advertisements, thus constantly refreshing each client's knowledge about available services. In client-driven rediscovery, clients have to periodically check for new servers by either communicating with the directory server or by broadcasting a new service discovery query. In both cases, however, clients need to periodically re-evaluate their service selection by pinging available servers as described above.

While the functionality of application-layer approaches to service discovery is independent of underlying routing protocol, the actual network cost (in terms of the number of messages send to perform service discovery and selection) is highly dependent on the underlying routing protocol. MANET routing protocols can be characterized as either *on-demand* or *proactive*. On-demand routing protocols discover routes only when data is being actively sent to a destination. Proactive protocols, on the other hand, maintain an updated view of the network topology by requiring nodes to periodically exchange routing information.

In the rest of this section we first describe two representative MANET routing protocols: DSR [23] (on-demand) and DSDV [29] (proactive). We then describe the Service Location Protocol (SLP) [20], a widely-used application-layer service discovery protocol. Finally, we analyze the network cost of operating centralized and distributed versions of SLP on top of DSR and DSDV.

2.1.1 DSR

DSR is an on-demand source routing protocol. DSR has two operation modes: *route discovery* and *route maintenance*. Whenever a node needs to route a packet to some other node in the network, DSR first checks its local cache for a route to the destination. If DSR finds a route, it inserts the route into the packet and forwards the packet toward its destination. If no route is found, DSR switches to route discovery mode and broadcasts a route request packet. On receiving a route request packet, a node appends itself to the source route in the packet, and either (i) identifies itself as the destination by sending a route reply to the source via a reversed source route, or (ii) rebroadcasts the route request packet. On receiving the route reply, the source node adds the route to its cache and forwards the data packet along the newly acquired source route. Route discovery may result in many route responses (multiple routes to a destination). These source routes are cached by DSR and the shortest source route (fewest intermediate nodes) is used. DSR reduces the latency and frequency of route discoveries by allowing intermediate nodes to cache overheard routes and respond to route requests for which they have a cached source route.

Route maintenance is DSR's standard operation mode. While in route maintenance, DSR routes data packets using the source route. On receiving a data packet, a node unicasts the packet to the node listed as the next hop in the source route. If the link to the next node is broken, the node detecting the failure sends a route error packet back to the sender. Nodes overhearing the route error packet invalidate entries in their routing caches as needed. Upon receiving the route error packet, the sender attempts to find a new route to the destination node in its cache, and if none is found, switches to the route discovery mode.

2.1.2 DSDV

DSDV is a table-driven proactive routing protocol, where every node has a routing table entry for each destination node in the network. Each routing entry includes a destination's address, the next hop to the destination and a metric (usually path length). Nodes exchange route entries periodically, thus propagating network topology changes through the network.

2.1.3 SLP

The Service Location Protocol [20] (SLP) is an IETF standard for automatic service discovery in IP networks. The abstract architecture consists of ”User Agents” (clients), ”Service Agents” (servers) and ”Directory Agents” (directories). SLP can operate in either centralized or distributed mode. In the centralized mode, servers advertise their services to Directory Agents (DAs), and clients unicast their requests directly to the DAs, which respond with a list of all services that match the client’s request. Clients and servers learn about DAs by reading from a statically configured file, using DHCP [15], waiting to receive periodic DA advertisement message or sending a multicast message to trigger DA advertisement. In the distributed mode, no DAs are present and clients query servers directly by sending multicast requests to a known address. On receiving a multicast request, a server unicasts a response directly to the querying client. In the rest of this paper we refer to the centralized version of SLP as SLP-CENT and to the distributed version as SLP-DIST.

2.1.4 Application-Layer Network Cost

In this section, we provide analytical upper bounds on the cost of performing service discovery and selection. We define the terms *discovery cost* and the *selection cost* to be the number of additional network messages sent (on behalf of the underlying routing protocol or the application-layer service discovery system) for the purpose of discovering the identities of available service providers, and selecting a service provider, respectively.

Assume **s** service providers and one directory agent (DA) are present in the network. Further, assume that the cost of sending a unicast message is **u**, the cost of performing routing-layer broadcast is **b**, and the cost of performing application-layer broadcast[1] is **B**. Note that B is usually larger than b, which in turn, is much larger than u.

DSR In SLP-CENT$_{dsr}$, a client learns the identities of servers by unicasting a request to DA. However, when the client does not cache a valid DSR route to DA, DSR first broadcasts a route request for DA. On receiving the route request, DA unicasts a DSR route reply back to the client. For the sake of expediency, in the rest of this paper we use the term *route request-reply exchange* to describe a DSR route request broadcast followed by a route reply (which has a cost of $b+u$). Once a route to DA is available, the client can finally unicast the message to DA, which responds (via unicast) with a list of available service providers. Thus, the discovery cost for SLP-CENT$_{dsr}$ is $b+3*u$.

We next determine the selection cost for two SLP-CENT$_{dsr}$ implementations: SLP-CENT-PING$_{dsr}$, which tries to select the closest server[2] by unicasting ping messages to all known servers and choosing the server whose ping reply arrives first, and SLP-CENT-RD$_{dsr}$, which picks one of the available servers at random.

The cost of performing server selection for SLP-CENT-PING$_{dsr}$ protocol over DSR is the cost of discovering routes to each of the servers and then sending and receiving a ping from every server. Thus, the selection cost for SLP-CENT-PING$_{dsr}$ is $s*b+3*s*u$. Therefore, the total cost of discovery and selection for SLP-CENT-PING$_{dsr}$ is equal to $(s+1)*(b+3*u)$.

For MANETs where the number of nodes significantly exceeds the number of servers (which we expect to be the common case), the service discovery cost is dominated by the broadcast traffic. Hence, the discovery and selection cost for SLP-CENT-PING$_{dsr}$ can be approximated to $(s+1)*b$.

Because SLP-CENT-RD$_{dsr}$ picks one of the servers randomly it does not generate any extra traffic, and therefore its selection cost is 0. However, when comparing the selection cost of SLP-CENT-RD$_{dsr}$ with that of SLP-CENT-PING$_{dsr}$ it is important to note that in SLP-CENT-PING$_{dsr}$ the client has a valid route to the selected server, whereas in SLP-CENT-RD$_{dsr}$ this is likely not the case. A more realistic selection cost for SLP-CENT-RD$_{dsr}$, which includes the cost of finding a DSR route to the randomly selected server, is $b+u$. Therefore the discovery and selection cost for SLP-CENT-RD$_{dsr}$ is equal to $2*b+4*u \approx 2*b$.

In SLP-DIST$_{dsr}$, a client learns the identities of servers by multicasting a service discovery query. For simplicity, we assume that the multicast message is implemented using application-layer broadcast. Servers that match the broadcasted query unicast a response back to the client. However, for servers that do not cache a valid DSR route to the client[3], DSR first has to perform route discovery. Thus, the discovery cost of SLP-DIST$_{dsr}$ is $B+s*(b+u)+s*u$.

The selection cost for SLP-DIST$_{dsr}$ is 0. A client just picks the server who replies first to the broadcasted service discovery query. The total discovery and selection cost for SLP-DIST$_{dsr}$ is $B+s*(b+2*u) \approx B+s*b$.

DSDV Because DSDV routing traffic is periodic and independent of the pattern of application traffic, in calculating the discovery and selection costs we only need to consider application-layer messages sent between the client, DA and the servers. Hence, the discovery cost for SLP-CENT$_{dsdv}$ and SLP-DIST$_{dsdv}$ are equal to $2*u$ and $B+s*u$, re-

[1]Application-layer broadcast differs from the DSR route request broadcast in several ways. First, an application layer broadcast is more expensive, since it is delivered to every node in the network; while a DSR route request is forwarded only if the node does not know a route to the required destination. Second, an application layer broadcast does not contribute to network topology information propagation (e.g., do not create reverse source routes).

[2]SLP is an application-layer service discovery implementation, and as such does not have direct knowledge of network topology.

[3]The application-layer service discovery query does not create DSR reverse source routes

	DSR	DSDV
SLP-CENT-RD	$2*b$	$2*u$
SLP-CENT-PING	$(s+1)*b$	$(s+1)*2*u$
SLP-DIST	$B+s*b$	B
CL	b	0

Table 1: Cost of service discovery and selection in terms of overhead messages (s is the number of servers, u is the cost of unicast message, b is the cost of routing-layer broadcast and B is the cost of application-layer broadcast). $B > b \gg u$.

spectively. The selection costs for SLP-CENT-RD$_{dsdv}$ and SLP-DIST$_{dsdv}$ are both equal to 0, while the selection cost for SLP-CENT-PING$_{dsdv}$ is equal to $2*s*u$. Therefore, the total cost of discovery and selection for SLP-CENT-RD$_{dsdv}$, SLP-CENT-PING$_{dsdv}$ and SLP-DIST$_{dsdv}$ are $2*u$, $(s+1)*2*u$ and $B+s*u \approx B$, respectively.

Table 1 summarizes the total costs of discovery and selection for three variants of SLP running over DSR and DSDV.

2.2 Cross-Layer Service Discovery

Because MANET routing protocols are distributed by design, a cross-layer (CL) approach to service discovery for MANETs can only be implemented following a distributed architecture. In the rest of this section we first describe how DSR and DSDV can be extended to provide service discovery functionality. We then analyze the network cost of performing service discovery using the extended protocols.

2.2.1 CL$_{dsr}$

A simple extension to the DSR route discovery mechanism enables nodes to discover routes to nodes hosting services. The extended cross-layer protocol, CL$_{dsr}$, supports service discovery by broadcasting a service request packet that contains a description of the requested service. On receiving the service request packet, a node appends itself to the source route and determines if it hosts a service that matches the service description. If this is the case, the node replies to the source by sending a service reply packet via a reversed source route. Otherwise, the node rebroadcasts the service request packet. At the end of this process, the node that initiated the service discovery learns the identity of one or more nodes that host services matching the service description and one or more source routes to these nodes. Analogous to DSR, CL$_{dsr}$ reduces the latency and frequency of service discoveries by allowing intermediate nodes to cache overheard mappings between service descriptions and service locations, and to respond to service requests for which they have a cached mapping. CL$_{dsr}$ learns about topology changes implicitly when either a source route breaks or it overhears a service reply that has a new service mapping.

2.2.2 CL$_{dsdv}$

DSDV can be extended to support service discovery by integrating service descriptions into routing table entries. In the resulting cross-layer protocol, CL$_{dsdv}$, each entry in the routing table has an additional field with the service descriptions of the services hosted by the node.

2.2.3 Cross-Layer Network Cost

The total cost of discovery and selection for CL$_{dsr}$ consists of a broadcast of a service discovery request packet followed by the unicast replies from the the matching servers. Given that the message cost of a service discovery broadcast is similar to that of a route discovery broadcast, the total cost is $b+s*u$.

Since no additional messages are sent into the network, the total cost of cross-layer discovery and selection for CL$_{dsdv}$ is 0! However, this comes at the cost of larger routing table entries.

2.3 Comparison

The total cost of discovery and selection is summarized in Table 1. As can be seen, cross-layer approach is cheaper than any of the application-layer approaches. Moreover, cross-layer implementation adapts better to changes in network topology than application-layer implementations, which have to proactively probe the network via pings incurring significant overhead ($s*b+3*s*u$ over DSR and $2*s*u$ over DSDV). In contrast, cross-layer approach may either learn about network topology implicitly (i.e., for free) or initiate a new service discovery with a maximal cost of $b+s*u$.

Application-layer service discovery implementations preserve the traditional modularity of the networking stack and thus are portable across transport and routing protocols. A cross-layer service discovery implementation, on the other hand, has significant dependencies on the underlying routing protocol, and therefore has to be customized for different routing protocols. However, exposing the routing information has considerable benefits for service discovery, selection and rediscovery, which we feel outweigh the disadvantage of breaking the abstraction layers.

3 Implementation

In this section, we first present a general architecture for cross-layer service discovery. We then describe two prototype implementations based on DSR and DSDV.

3.1 Architecture

Several factors have to be considered in the design of a general architecture for cross-layer service discovery. On the one hand, a cross-layer service discovery implementation has to be closely coupled to the underlying routing mechanisms to exploit routing traffic for service discovery. On the other hand, the functionality that a cross-layer service discovery implementation needs to provide is largely independent of the underlying routing protocol. Specifically, the architecture has to propagate service information across the network, match service discovery requests with advertisements, provide the application with accurate information for selecting the best server among those available, track network topology changes and inform the application so that it can take corrective measures (e.g., switch to a closer server). Finally, the cross-layer architecture should provide the means for efficient server selection and rediscovery, but leave it to an application to determine how to select servers and when and how to re-evaluate its choices.

These consideration lead to the split architecture design shown in Figure 1, which consists of two main components: a routing-protocol independent Service Discovery Library (SDL), and a Routing Layer Driver (RLD) that is closely coupled with MANET routing mechanisms. SDL provides a consistent view of the cross-layer service discovery mechanism to client applications and service providers, isolating them from much of the intricacies of the underlying routing protocol. RLD interacts closely with the MANET's routing protocol to propagate service discovery messages and track network topology changes.

SDL's main data structure that stores information about known servers is called *service table*. Table entries have five fields: service description, service location (e.g., "172.16.1.1:80"), minimum hop count from the current host to a service provider, optional routing protocol specific information provided by RLD (e.g., a list of available routes to the destination), and optional service-specific metrics supplied by a service provider (e.g., current load, CPU usage).

All interactions between client applications, service providers and SDL, as well as between SDL and RLD, follow well-defined interfaces. Clients and servers call on SDL to issue service discovery requests and propagate service advertisements, whereas SDL notifies applications of changes to the service table by invoking an application-specified callback function. SDL calls on RLD to disseminate service discovery requests and advertisements, and propagate service discovery replies. RLD forwards service discovery messages it receives to SDL and informs it about changes in network topology.

3.1.1 Discovery

Client applications learn about available servers by instructing SDL to find all entries in its service table that match a service description. When no matching entries are found, the application has to wait for servers to be discovered.

SDL supports two modes of service discovery: *active discovery* and *passive discovery*. Active discovery is client

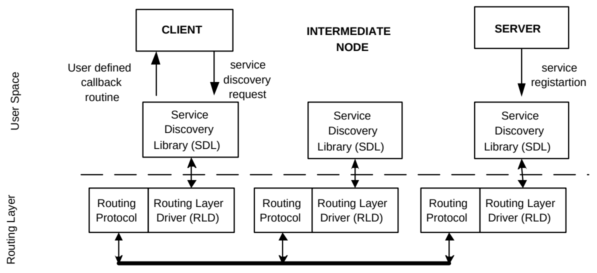
CLIENT
INTERMEDIATE NODE
SERVER
User defined callback routine
service discovery request
service registartion
User Space
Service Discovery Library (SDL)
Service Discovery Library (SDL)
Service Discovery Library (SDL)
Routing Layer
Routing Protocol
Routing Layer Driver (RLD)
Routing Protocol
Routing Layer Driver (RLD)
Routing Protocol
Routing Layer Driver (RLD)

Figure 1: Cross-Layer Service Discovery Architecture

driven. In active discovery, SDL instructs RLD to disseminate discovery requests for a specific service, and waits for explicit responses to flow back. The actual mechanisms used by RLD to disseminate service discovery requests are implementation dependent. For example, RLD implementations build on top of on-demand routing protocols such as DSR and AODV would disseminate discovery requests by broadcasting modified route discovery messages. In contrast, an RLD implementation for a hierarchical MANET protocol such as CBRP [28] would unicast discovery requests to the cluster-head.

Passive discovery, in contrast, is server driven. In passive discovery, SDL instructs RLD to periodically disseminate advertisements for a specific service. As was the case for active discovery, the mechanism used by RLD to disseminate advertisements depends on the underlying routing protocol. For example, for proactive protocol such as DSDV, RLD would extend route table entries with service information.

RLD hands service discovery requests, replies and advertisements it intercepts from the network to SDL, which inspects them and modifies entries in its service table accordingly (e.g., adds an entry for a newly discovered server). On receiving a service discovery request, SDL checks its service table for a match. To make SDL independent of a particular service description language, the matching between the service description as advertised by service providers and the service discovery requests is performed by a pluggable *matching module*. For our experiments, we used a simple module that performs a string based comparison. If SDL finds a match, it instructs RLD to compose a service response. Otherwise, it instructs RLD to rebroadcast the request. The actual mechanisms used by RLD to propagate service responses or forward service requests depends on the underlying routing protocol.

3.1.2 Selection

When multiple entries in the service table match a client's service description, the client application selects one based on the metrics stored in the SDL service table. In our current implementation, clients choose the server with the low-

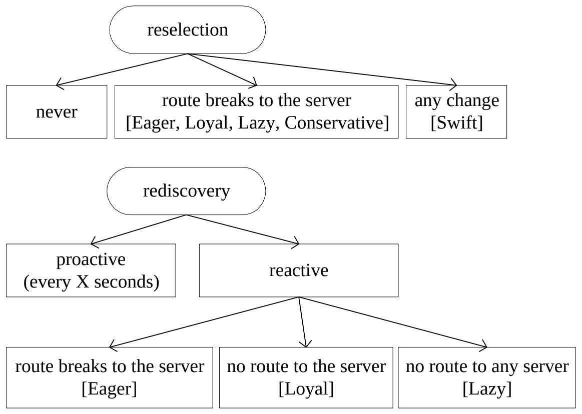
reselection
never
route breaks to the server [Eager, Loyal, Lazy, Conservative]
any change [Swift]
rediscovery
proactive (every X seconds)
reactive
route breaks to the server [Eager]
no route to the server [Loyal]
no route to any server [Lazy]

Figure 2: Reselection and rediscovery policies.

est hop count; however, other service or routing specific metrics can be used in server selection (e.g., choosing a server that has the least load).

3.1.3 Reselection and Rediscovery

To optimize performance, MANET clients need to constantly re-evaluate their choice of a service provider. Re-evaluation has two components: *reselection* and *rediscovery*. Reselection reconsiders server selection based only on the current entries in SDL service table. Rediscovery involves probing the network for up-to-date information about available service providers, and is therefore available only on cross-layer implementation that support active discovery. Rediscovery is usually followed by reselection.

In designing a re-evaluation policy, application developers need to determine when to do reselection and (if available) rediscovery. Figure 2 lays out the design space for reselection and rediscovery policies. The names in brackets refer to our implemented policies discussed later in the section.

The simplest *reselection* policy is not to do reselection at all. This policy, however, will likely result in poor performance. An alternative is do reselection in reaction to a change in the SDL service table. There is a wide spectrum of possible reactive reselection policies. On one end are policies that do reselection in response to any change to the service table, such as finding a new server, or learning of a change to a service-specific metric (e.g., server is overloaded). On the other hand are policies that do reselection only when there is no valid route to the current server. An intermediate approach is to do reselection when the active route to the current server breaks [4].

Applications may choose to trigger *rediscovery* either proactively or in reaction to a change in the service table.

[4]Some cross-layer implementations keep multiple routes to a destination. The active route is the one that is currently being used to forward packets between the client and the server.

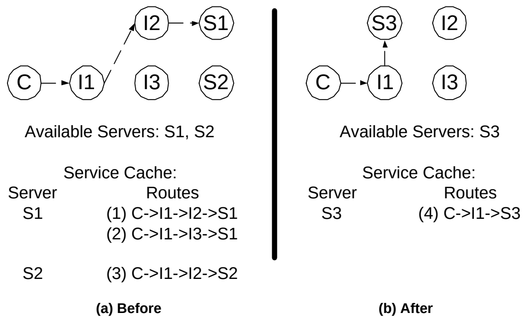

Figure 3: Rediscovery example

There is a wide spectrum of possible reactive rediscovery policies. On one end, an *eager* reactive policy triggers rediscovery as soon as the active route to the current server breaks. On the other, a *lazy* reactive policy delays active rediscovery until it has tried all known routes to all known servers that implement a service. Between these two extremes lies a large set of possible reactive rediscovery policies that try a subset of the available routes to a subset of the known servers before triggering rediscovery. In this space, a policy we found works well in practice triggers a rediscovery after trying all known routes to the current server. We refer to this policy as *loyal*.

Figure 3 illustrates the use of a reactive rediscovery policies for a network that consists of three servers (S1, S2 and S3) that provide the same service, one client node (C) and three intermediate nodes (I1, I2, I3) that participate in the communication but do not host or make use of the service. Figure 3(a) shows the initial state of SDL service table for node C, which contains three routes to two servers S1 and S2. Let us assume that the C communicates with S1 over route (1), and that without notice S1 and S2 leave the network. As a result, the next use of route (1) will result in a routing failure. In this scenario, *eager* will trigger rediscovery right away; *loyal* will first try the route (2), and then, on failure, will trigger rediscovery; whereas *lazy*, will try routes (2) and (3) before triggering a rediscovery. Figure 3(b) shows the state of the SDL service table after rediscovery.

3.2 Prototypes

3.2.1 CL$_{dsr}$

We extended the existing DSR route request/reply mechanisms to perform service discovery by adding fields to the standard route request and reply packet headers. We added two fields to the route request packet: (1) service description, which contains a description of the service to be discovered, and (2) service discovery flag, which is set when the service description field is non-empty. We added four additional fields to the route reply packet: (1) service description, which contains a description of the service as advertised by the service provider, (2) service discovery flag, (3) service location, which contains the location of the service and (4) service metric, which contains additional metrics as advertised by a service provider.

When a client issues a request for a service that is not in the local SDL's service table, RLD broadcasts a service discovery request packet to the network. The packet is a modified DSR route request with the service description field populated to the required service description and the service discovery flag set.

RLD investigates every received packet and delivers it to the local SDL if the packet is a service discovery packet. SDL matches the service description in the packet with the data found in its local service table and on a successful match, instructs the driver to issue a reply to a source node. RLD generates a service discovery reply packet, which is a modified DSR route reply packet, populates all the required fields and unicasts the packet back to the original sender.

Similar to DSR's intermediate node route caching and replying, our implementation allows intermediate nodes to (i) learn about available services by overhearing service discovery reply messages, and (ii) reply to service discovery requests, by checking local service table and responding if the match is found. If not used or updated, service table entries are periodically invalidated.

We developed 3 rediscovery and reselection policies for CL_{dsr}. All policies choose the server with the shortest route. They differ, however, in the eagerness with which they trigger a service rediscovery.

Eager Triggers service rediscovery immediately after the current route to the server breaks.

Loyal Triggers service rediscovery only if no valid route exists to the current server. Loyal delays rediscovery until it has tried all cached routes to a server.

Lazy Triggers service rediscovery only when there are no valid routes to any of the servers that offer the service. Lazy defers rediscovery as much as possible, waiting until all routes to all known service providers prove to be faulty.

3.2.2 CL_{dsdv}

We extended the DSDV routing table entries with three additional fields that store the service description, location and extra service-specific metrics. Nodes learn about available services while performing the regular routing table exchange operation, and no additional service discovery packets are sent into the network.

CL_{dsdv} does not perform active service discovery. Instead, the RLD monitors changes in the routing table and notifies SDL when either changes in network topology occur or new services are passively discovered during the routing table exchange.

We have implemented 2 reselection policies for CL_{dsdv}:

Swift Switches servers as soon it becomes aware of a server with a smaller hop count.

Conservative Switches servers only when there is no longer a valid route to the current server.

3.3 Discussion

While our current implementation assists applications in selecting the best available server, it is not aware of the actual client-server communication and does not provide additional support for migrating client sessions between servers. While migrating a stateless service may be as simple as opening a new connection to the new server, migrating stateful services requires guaranteeing that application-specific state is consistent across the old and the new server. Additionally, if the client connects to the server over a reliable transport protocol, such as TCP, it is also necessary to ensure that connection-specific state is consistent across the two servers. Application-state migration and connection-state migration are outside the scope of this work and have been researched by other groups [19, 34].

However, we believe that there are many useful stateless services that could benefit from server reselection. Clients of stateful services may still use our architecture for initial service discovery and selection, and simply choose not to be notified when a better service provider is available.

Due to the tight integration between service discovery and routing mechanisms, the scalability of our cross-layer architecture is tied to the inherent scalability of the underlying routing protocol. We expect the scalability of the architecture to improve with advances in MANET routing.

4 Evaluation

In this section, we use extensive simulations to compare the performance of the cross-layer (CL) implementation described in Section 3 to that of an application-layer service discovery implementation closely modeled after the Service Location Protocol [20] (SLP) described is Section 2. We further refer to the version of CL that runs over DSR as CL_{dsr} and to the version that runs over DSDV as CL_{dsdv}.

We have implemented both centralized and distributed versions of SLP, which we refer to as SLP-CENT and SLP-DIST, respectively. While SLP clients can base service selection on a myriad of factors[5], in this paper we evaluate two approaches: random (RD), which picks one of the available servers at random, and pinging (PING), which tries to select the closest server by unicasting ping messages to all available servers and choosing the server whose ping reply arrives first. Altogether, we consider three variations of SLP,

[5]The SLP standard does not cover server selection, leaving this decision entirely to the client.

two centralized SLP-CENT-RD and SLP-CENT-PING, and the distributed SLP-DIST, which picks the server whose response to the broadcasted service discovery query arrives first.

To eliminate any transient effects due to initializations, in our SLP-CENT variations we have preloaded clients and servers with a DA's address and the DA with the information about services in the network. In our SLP-DIST variations, we have implemented the multicast functionality via a simple application-layer broadcast. We made this implementation decision to avoid the high cost associated with maintaining a multicast group in wireless networks [27]. Finally, all our SLP implementations communicate over UDP. Similarly to [7], we chose not to use TCP communication to allow more accurate comparison between protocols[6].

In the rest of this section, we first describe our experimental environment and then present the results of our evaluation.

4.1 Experimental Environment

For our simulations we used the `ns-2` simulator [16] with CMU wireless extension [37]. The physical radio characteristics were chosen to approximate the Lucent WaveLAN direct sequence spread spectrum radio with a 250m nominal transmission range and a raw capacity of 2Mb/s. All experiments use distributed coordination function (DCF) of IEEE 802.11 as the MAC protocol.

We report results for a network of 100 nodes randomly placed on a rectangular 300m x 2000m flat space. Similarly to [7, 14], the rectangular shape was chosen to force the use of longer routes between communication pairs. Nodes move following the random waypoint model [7] with no pause time to stress test our implementations. In this model, a node chooses a random point within the space and starts moving toward that point at a speed randomly chosen from an interval 0-V_{max} (in our experiments V_{max} is equal to either 2m/s or 20m/s). Upon reaching its destination, the node selects another destination and speed, repeating the process.

Clients communicate with servers by sending 100 byte packets at a constant bit rate (CBR). We experimented with several client sending rates to simulate both unsaturated and saturated networks (2.5 packets/sec and 7.5 packets/sec) [14], number of servers (1, 2, 4 and 6) and number of clients (30, 50, and 70). For every configuration, we present results averaged at 5 movement scenarios.

All experiments start with a 100 second service discovery phase during which clients discover servers for further communication. After 100 seconds of simulation, clients start sending data to servers. Without loss of generality, all servers in the network offer the same service.

[6]TCP source varies the time at which it sends packets based on its perception of the network's congestion state. Consequentially, the time at which packets are sent and the position of nodes at that time will differ between protocols.

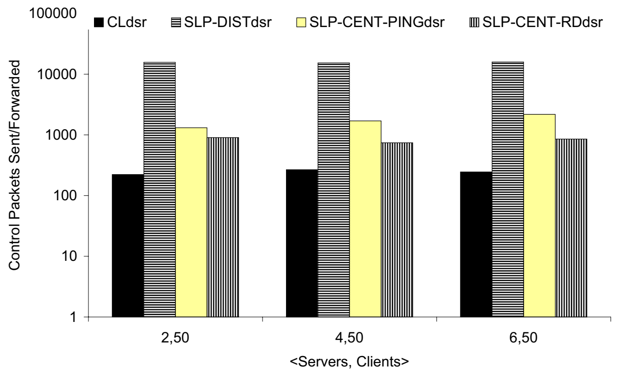

Figure 4: **[DSR]** Cost of service discovery (2, 4 and 6 servers, 50 clients).

4.2 Experimental Results

4.2.1 Stationary Network

This section evaluates the behavior of the proposed service discovery protocols in stationary networks in terms of control message overhead, throughput, delivery ratio and initial route length to a server. For all presented stationary experiments we keep the number of clients constant at 50 while we increase the number of servers from 2 to 6.

We experimented with both DSR and DSDV. However, because of space limitations, and given that the results are very similar for both protocols, we only present figures that plot results for DSR.

Control Message Overhead

Figure 4 shows the message overhead for service discovery for the various protocols over DSR. Included are DSR packets and service discovery packets (no MAC packets are included). Those experimental results confirm our analytical calculations summarized in Table 1.

CL$_{dsr}$ consistently outperforms the other protocols, as it benefits from using the routing traffic to learn about servers and routes to them simultaneously. The huge overhead of SLP-DIST$_{dsr}$ results from using application-layer broadcast, which as opposed to the route discovery broadcast does not propagate DSR routes throughout the network. Varying the number of clients preserved the relative cost between the protocols.

Experimental results over DSDV (not shown due to space limitations) exhibit similar pattern, with the exception that the protocols suffer an additional constant overhead as a result of the periodic routing table exchanges.

Route Length

Figure 5 shows the optimal average path length from a client to the closest server and the average route length for each of

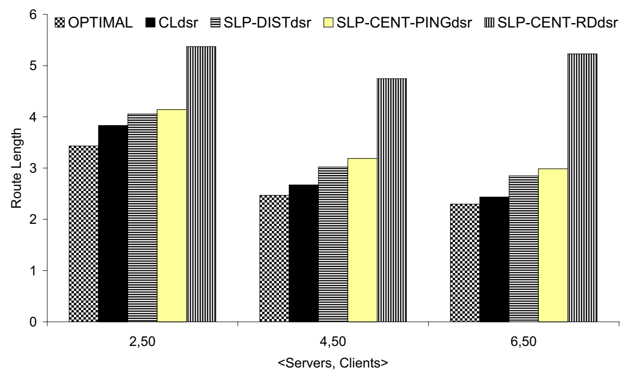

Figure 5: **[DSR]** Average route length (2, 4 and 6 servers, 50 clients).

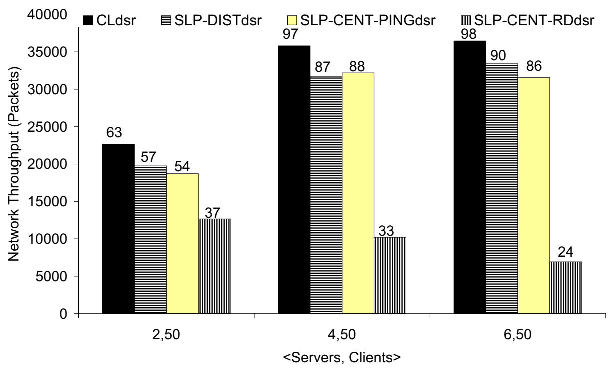

Figure 6: **[DSR]** Network throughput and delivery ratios (2, 4 and 6 servers, 50 clients).

the service discovery protocols running over DSR. Varying the number of clients did not affect initial route lengths.

While CL_{dsr} picks optimal routes most of the time. The gap that exists between CL_{dsr} and Optimal results from packet losses due to collisions that lead CL_{dsr} to choose sub-optimal routes. SLP-$DIST_{dsr}$ and SLP-CENT-$PING_{dsr}$ have longer average routes because they do service selection based on limited knowledge of network topology. Both SLP-based protocols try to find the closest server by measuring the round-trip-time. Unfortunately, the time to unicast a message from the client to a server or from a server back to the client varies greatly based on the availability of DSR routes. Therefore, a remote server that has a DSR route to the client can reply ahead of a closer server that has to do DSR route discovery. The SLP-CENT-RD_{dsr} picks servers randomly and is obviously the worst case. Note that SLP-CENT-RD_{dsr} is not a purely hypothetical worse protocol. It represents a case where no topology information is available or recorded (as typically the case when a client requests all available services for browsing and then chooses one randomly).

Experimental results over DSDV are very similar and are therefore not shown. Because of packet losses due to collisions, SLP-$DIST_{dsdv}$ and SLP-CENT-$PING_{dsdv}$ tend to choose longer routes than CL_{dsdv}.

Throughput and Delivery Ratio

Figure 6 shows overall network throughput for 100 seconds of communication of the service discovery protocols running over DSR. Clients send data at a fixed rate of 7.5 packets/second (750 bytes/sec). The numbers on top of the bars show delivery ratios.

At this sending rate (which comes close to saturating the network), there is a strong correlation between the average path lengths of Figure 5 and network throughput. We also ran experiments with a 2.5 packets/second sending rate (not shown due to space limitations). At the lower sending rate, the performance of CL_{dsr}, SLP-$DIST_{dsr}$ and SLP-CENT-$PING_{dsr}$ is comparable with delivery rates reaching 99%. For the lower sending rate with no node mobility, the network is not saturated enough and packets experiencing low contention levels successfully reach even the farthest servers.

Once again, the throughput and delivery ratio results for DSDV are very similar and are thus omitted. This similarity is not surprising since the overall throughput is highly dependent on the average route length, which is very similar for both DSR and DSDV.

In summary, server selection has a critical effect on overall network performance. Near optimal service selection based on route length clusters clients and servers, localizing communication and reducing interference. Moreover, with good service selection, as the number of servers increases, overall network throughput goes up. Poor service selection, on the other hand, results in interference that severely limits network throughput. It is interesting to note that with poor service selection, as the number of servers grows, interference increases further degrading network capacity (as seen by throughput results of SLP-CENT-RD_{dsr} protocol).

4.2.2 Effects of Motion

This section evaluates the effects of node mobility on the service discovery protocols. In all mobility experiments, clients start sending data at 100 seconds, motion starts at 200 seconds, and the experiments end at 800 seconds. We do not include results for SLP-$DIST_{dsr}$ with mobility because the initial server selection of SLP-CENT-$PING_{dsr}$ and SLP-$DIST_{dsr}$ protocols are very similar, and consequentially, there is no difference between them in the mobility simulations. Since results for DSDV and DSR are similar, we present DSR-based results only.

Figure 7 plots the network throughput over time for all protocols for 4 servers and 50 clients with a CBR sending rate of 7.5 packets/sec. Nodes move with a V_{max} of 20 m/sec. The figure shows that without rediscovery, all the protocols converge to the same throughput - that of the

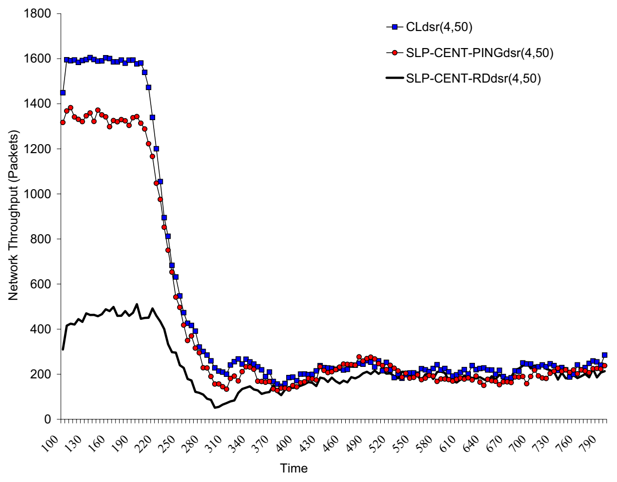

Figure 7: **[DSR]** Effects of mobility on network throughput (4 servers, 50 clients). The numbers in parenthesis follow the pattern of (server, client).

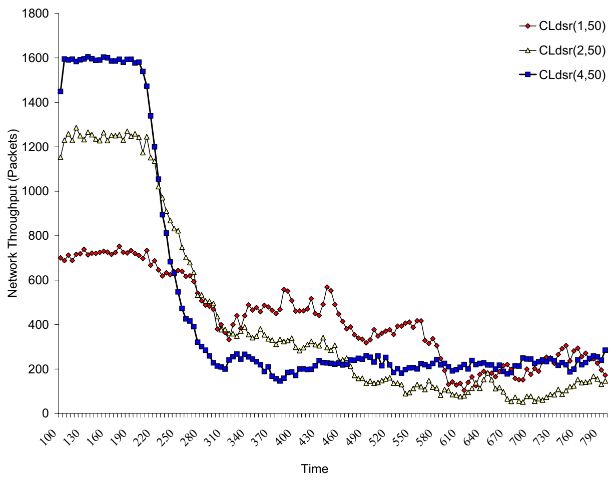

Figure 8: **[DSR]** Effects of mobility with different number of servers on network throughput (1, 2 and 4 servers, 50 clients).

protocol with a random server choice. This is an expected result, as the initial server choice becomes irrelevant after a period of movement. Results for a similar experiment with V_{max} set to 2 m/sec show a similar trend (not shown due to space limitations), with the caveat that it takes longer for the protocols to converge because more time is needed at the low speed to "shuffle" the nodes into a random topology.

Figure 8 presents an interesting result. The figure plots the network throughput over time for CL$_{dsr}$ with 1,2, and 4 servers and 50 clients with a CBR sending rate of 7.5 packets/sec. Nodes move with a V_{max} of 20 m/sec. Before mobility starts, network throughput goes up with the number of servers. With mobility, however, an increase in servers leads to a decrease in network throughput. This result is consistent with the static SLP-CENT-RD$_{dsr}$ throughput measurements in Figure 6. Although the average path length to a server degrades with mobility to the same random value for all instances, the increased number of servers in the network cause more interference between client-server pairs. So, for mobile networks with no rediscoveries decreasing the number of servers may actually increase network performance. This result is not specific to CL$_{dsr}$. The graphs for SLP-CENT-RD$_{dsr}$ and SLP-CENT-PING$_{dsr}$ (not included in the paper) show a similar trend.

The trends outlined by Figure 7 and 8 motivate the need to re-evaluate server selection to offset the effects of node mobility on network capacity. It is compelling that as the number of servers in a network increases, the need to re-evaluate server selection becomes more urgent.

4.2.3 Rediscovery Strategies

Figure 9 and 10 show the performance of CL$_{dsr}$ and CL$_{dsdv}$ rediscovery strategies. SLP-CENT-PING60$_{dsr}$ is a version of SLP-CENT-PING$_{dsr}$ that ping all known servers every 60 seconds, and selects the one with the shortest round-trip-time. We present results for a 60 second pinging period because this is the period that achieves the best throughput for our network. SLP-CENT-RD$_{dsr}$ is included for comparison. All protocols run on a network with 4 servers and 50 clients with a CBR sending rate of 7.5 packets/sec. Nodes move with a V_{max} of 20 m/sec. (experiments with V_{max} set to 2m/sec show similar trends, and are therefore not included).

In DSR-based simulations, CL$_{dsr}$-Loyal achieves the best performance, validating the DSR policy of trying all cached routes to a given node before issuing a route rediscovery (see Section 3.1.3 for description of the policies). CL$_{dsr}$-Eager suffers from network congestion as a result of sending rediscovery requests as soon as the active route breaks. CL$_{dsr}$-Lazy, on the other hand, is too conservative. Because of the network mobility, routes in the DSR's route cache are often stale. Holding rediscovery until all routes to all known servers have been tried makes the client wait for a long time before doing a rediscovery and resuming sending data to a server. CL$_{dsr}$-Lazy also chooses longer routes over a rediscovery leading to poor locality.

In DSDV-based simulations, CL$_{dsdv}$-Swift achieves the best performance. Since no additional packets are being transmitted into the network, CL$_{dsdv}$-Swift does not incur any penalty for switching to the best possible service as it becomes available. CL$_{dsdv}$-Conservative, on the other hand, waits for the active route break and thus misses opportunities of switching to better servers.

SLP-CENT-PING60$_{dsr}$ shows that server rediscovery based on timing measurements such as pinging can increase network throughput. However, a substantial gap still remains between the best cross-layer rediscovery technique, and the application-layer implementation. Two factors ac-

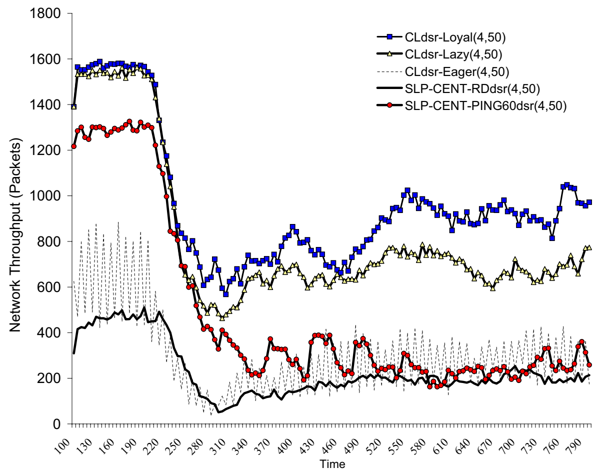

Figure 9: **[DSR]** Effectiveness of rediscovery strategies in heavily-loaded networks (4 servers, 50 clients).

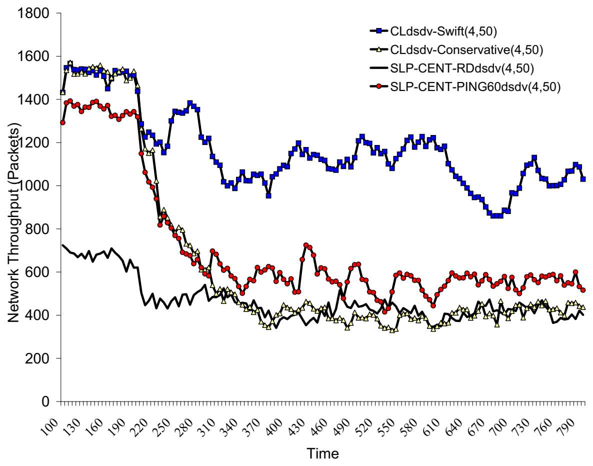

Figure 10: **[DSDV]** Effectiveness of rediscovery strategies in heavily-loaded networks (4 servers, 50 clients).

count for SLP-CENT-PING60$_{dsr}$ lower performance. First, pinging adds a significant amount of message overhead to an already heavily-loaded network, creating more congestion and interference. Second, to be effective the pinging period needs to be compatible with the rate of network topology change. To determine the sensibility of SLP-CENT-PING$_{dsr}$ to the rediscovery period, we varied the pinging period between 30 and 90 seconds. While the 60 second pinging period achieves the best throughput, overall network throughput was not significantly affected by varying the pinging period.

Figures 11 and 12 show results for a network with 4 servers and 50 clients with a low CBR sending rate of 2.5 packets/sec. Nodes move with a V_{max} of 20 m/sec. SLP-CENT-PING60 performs better in this less congested scenario, but still falls short of the best CL rediscovery policy.

Interestingly, although the 60 seconds pinging period still achieves the best performance over DSR, it fails to beat the 30 seconds pinging rediscovery over DSDV. This suggests that it will be hard for the application-layer service discovery to derive the optimal periodic rediscovery policy, and the trivial policy of performing the rediscovery every *x* seconds will fail to achieve the optimal performance independent of the value of *x*.

Figure 13 shows the performance of CL$_{dsr}$-Loyal (the best-performing CL$_{dsr}$ variation) for networks with 1,2, and 4 servers. In all cases, there are 50 clients with a CBR sending rate of 7.5 packets/sec. Nodes move with a V_{max} of 20 m/sec.

These results show that CL$_{dsr}$-Loyal is able to keep traffic localized despite network mobility, and therefore can take advantage of an increase in the number of servers. Moreover, at either the lower node V_{max} speed of 2 m/sec or the CBR sending rate of 2.5 packets/sec., CL$_{dsr}$-Loyal's throughput comes close to that of the network without mo-

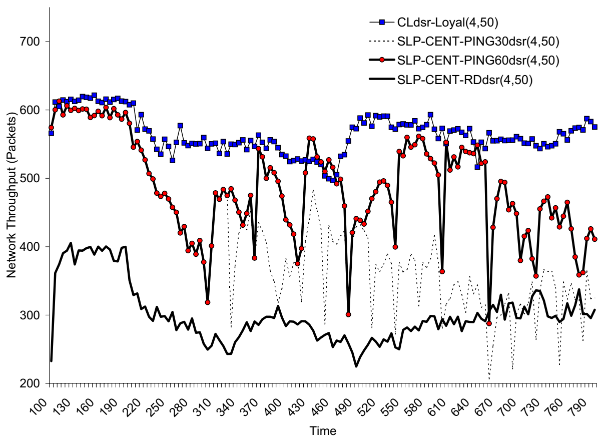

Figure 11: **[DSR]** Effectiveness of rediscovery strategies in lightly-loaded networks (4 servers, 50 clients).

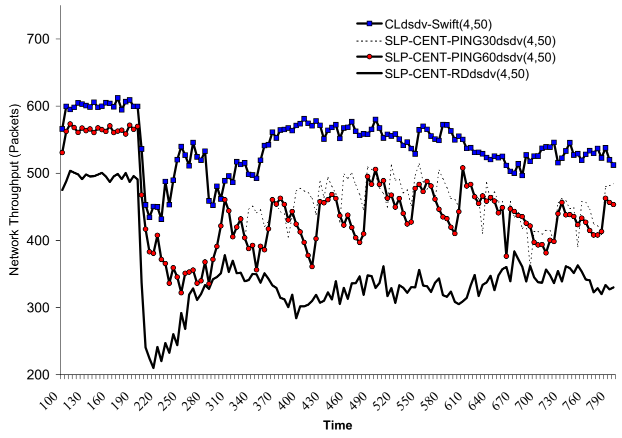

Figure 12: **[DSDV]** Effectiveness of rediscovery strategies in lightly-loaded networks (4 servers, 50 clients).

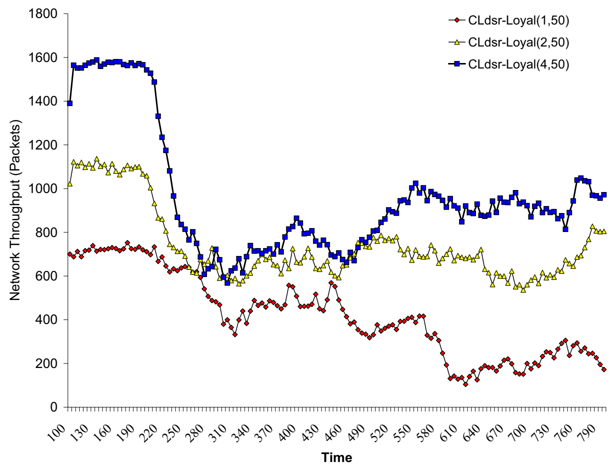

Figure 13: **[DSR]** Effects of rediscovery with different number of servers on network throughput (1, 2 and 4 servers, 50 clients).

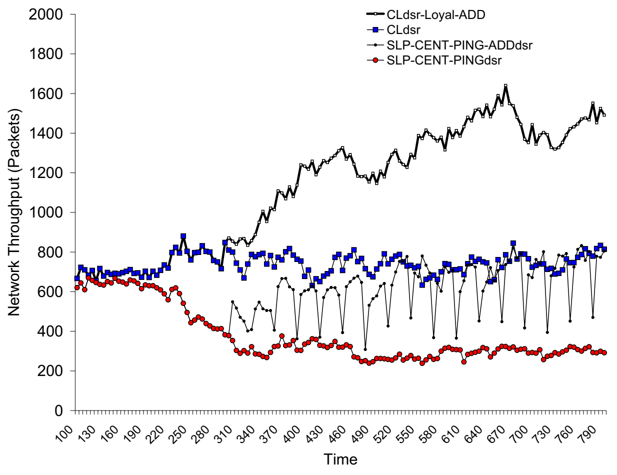

Figure 14: **[DSR]** Effects of adding 10 new servers to the network.

bility. This is illustrated in Figure 11 for an experiment involving 4 serves and 50 clients a CBR sending rate of 2.5 packets/sec and nodes moving with a V_{max} of 20 m/sec.

4.2.4 Handling New Services

Figure 14 evaluates the effectiveness with which CL$_{dsr}$ and SLP can take advantage of new servers as they are added to the network. The experiments starts with a network that has 2 servers and 50 clients with a CBR sending rate of 7.5 packets/sec. Nodes move with a V_{max} of 20 m/sec. We then add a new server every 30 seconds of simulation time, with a total of 10 servers added over the lifetime of the experiment. The results show that CL$_{dsr}$ is more effective than SLP in leveraging the new servers that are added to the network. As servers are incorporated to the network, CL$_{dsr}$ manages to localize communication more effectively with little extra cost for service discovery. In contrast, as servers are added to the network the cost of service discovery for SLP goes up as it has to ping a larger number of nodes to do service selection.

4.2.5 Evaluation Summary

In summary, CL$_{dsr}$ and CL$_{dsdv}$ protocols incur the least amount of message overhead, localize communication, and consequentially achieve the best throughput. In mobile networks, even with an optimal initial service selection, throughput of all the protocols degrades to that of random selection. This shows the need for effective rediscovery and reselection techniques to offset the effect of topology changes and improve network performance. Physical routing information, such as route lengths, have proved to be the best tool for localizing traffic during mobility. Round-trip-latency measures were shown to be effective in cases with small network saturation, but using them too often leads to congestion. Moreover, for heavily utilized networks round-trip-latency measures are ineffective and fail to localize traffic. In addition, periodic rediscoveries fail to capture the rate of network topology change, thus re-evaluating server status either too often or too rarely. Also, pinging adds significant amount of overhead to an already congested network, causing even more interference. In contrast, triggering rediscovery and reselection after detecting changes in network topology proved to be effective.

5 Related Work

Considerable previous research has gone into service discovery for distributed systems. However, most of the prominent architectures were not designed for use in wireless mobile environments. Some examples include SLP [20] from IETF, Jini [2] from Sun, SSDP [18] from Microsoft, Salutation [3] from IBM and Berkeley's SSDS [13]

There has been significant research on application-layer service discovery solutions for MANET. Allia [33] uses agent based service discovery, taking into account user specific policies. Handorean et al. [21] and MARE [35] combine agent based discovery with a distributed tuple space approach for storing and distributing services. Although maintaining the local tuple space data is costly, the service discovery phase usually narrows down to a lookup of the service information on the local device. Konark [22] uses multicast to advertise and discover services and allows for service delivery by running a lightweight HTTP server on every device that hosts services. The Intentional Naming System (INS) [4] is an example of a service discovery system that allows sending data to a service provider without discovering its address a priori. The network of Intentional Naming Resolvers (INRs) routes packets based on a service description included with the data payload. The user can either choose to send the data to any service provider that matches the requirement (anycast) or to all of them (multicast). Chen and Kotz [10] have extended INS with context-sensitive service discovery and Bisdikian et al. [6] propose an intelligent middleware for context-based services.

Works that use ontologies or some level of hierarchy for service description to reduce the amount of service advertisement information that gets propagated over the network include GSD [9] and Multi-Layer Clusters [25].

Azondekon et al. [5] argue for a need to select services based on a physical proximity (line of sight) and proposes two protocols based on a combination of infrared communication and SLP. Other systems that discover services based on the physical proximity include Satchel [17] and Cooltown [24].

Our approach differs from these efforts in that it exploits a close integration of service discovery functionality with the routing mechanisms of MANET. As a result, nodes in our system can exploit available topology information to reduce overhead and improve network performance.

Kozat and Tassiulas [27] propose a distributed service discovery architecture that relies on a virtual backbone for locating and registering available services within a dynamic network topology. Service Broker Nodes constitute a dominating set and act as directory agents, replying to discovery requests and registering service advertisements. The approach proves to be less costly than service discovery based on the multicast ODMRP protocol, but more costly than anycast based approaches. Tchakarov [36] propose a resource location protocols for multi-hop ad hoc networks that uses geographical information to reduce service advertisement and discovery traffic. Raman et al. [32] argue for extensive cross-layer optimizations in Bluetooth scatternets. As a result, scatternet wide floods are minimized by caching service discovery results at all intermediate nodes (this can be thought of as a distributed implementation of an SLP Directory Agent). Koodli et al. [26] propose extensions to MANET routing protocols to support service discovery. Cheng [12] suggests using On-Demand Multicast Routing Protocol for service advertisement and discovery. In this approach, each server and all its consumers make a multicast group. Servers advertise their services periodically and clients discover services by sending multicast requests to the group. However, the multicast groups have to be created and maintained, which is a costly process [27] resulting in a significant control message overhead.

In contrast to our work, these efforts either do not evaluate system performance at all or do not show the effects that server selection has on the overall network throughput. More significantly, these efforts do not consider the problem of server selection and rediscovery, which we have shown to be fundamental requirements for good performance on MANETs.

Chen et al. [11] propose a cross-layer design to allow QoS in MANET. Middleware uses specifically created location-aware routing protocol for lookup and replication of services. Routing-layer uses middleware's packet priority information to route packets. The communication between routing and middleware occurs through system profiles, which include information about node location and movement patterns. The system predicts group partitioning and replicates data from one partition to the other. The same data is hosted by only one of the group members. Our architecture, on the other hand, does not require a specialized location-aware routing protocol and is more concerned with the problem of service selection and discovery than with data replication.

6 Conclusions and Future Work

We identified three shortcomings of existing application-layer service discovery architectures. First, they suffer from a large control message overhead as a result of duplicate message generation of service discovery and routing-layers. Second, they do not provide any support for service selection once multiple comparable services are discovered in the network. Third, they lack support for rediscovery of services, to adapt to changes in network topology that result from node mobility or the arrival and departure of nodes hosting services.

We presented a new cross-layer architecture that integrates service discovery functionality with existing routing protocols. The architecture is designed up front with service selection and rediscovery in mind. It integrates service discovery with route discovery, thus allowing nodes to learn about available services and routes to them simultaneously, consequentially reducing control message overhead. Our cross-layer approach helps clients select the best possible service, and allows them to register call-back routines to be notified once a better service is available nearby.

We have analytically and experimentally showed that a cross-layer implementation consistently outperforms an application-layer service discovery implementation closely modeled after SLP. Not only does the cross-layer prototype incurs the least amount of overhead, it achieves network throughput up to 5 times higher than SLP.

In the future, we plan to extend our architecture to support other routing protocols (e.g., AODV). We also plan to continue development of our framework to include support for fault tolerance and recovery needed in cases where application state is stored at the service providers.

References

[1] Bluetooth Specification Part E. Service Discovery Protocol(SDP) http://www.bluetooth.com, July 1999.

[2] Jini javaspaces service specification. Available at http://www.sun.com/jini/specs.

[3] Salutation architecture. Available at http://www.salutation.org/whitepaper/originalwp.pdf.

[4] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. of 17th SOSP*, 1999.

[5] V. Azondekon, M. Barbeau, and R. Liscano. Service selection in networks based on proximity confirmation using infrared. In *Proc. of ICT*, 2002.

[6] C. Bisdikian, I. Boamah, P. Castro, A. Misra, J. Rubas, N. Villoutreix, D. Yeh, V. Rasin, H. Huang, and C. Simonds. Intelligent pervasive middleware for context-based and localized telematics services. In *Proc. of Workshop on Mobile Commerce*, 2002.

[7] J. Broch, D. A. Maltz, D. B. Johnson, Y.-C. Hu, and J. Jetcheva. A performance comparison of multi-hop wireless ad hoc network routing protocols. In *Proc. of MobiCom*, 1998.

[8] C. Carter, S. Yi, P. Ratanchandani, and R. Kravets. Manycast: Exploring the space between anycast and multicast in ad hoc networks. In *Proc. of MobiCom*, 2003.

[9] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin. GSD: A novel group-based service discovery protocol for manets. In *Proc. of MWCN*, 2002.

[10] G. Chen and D. Kotz. Context-sensitive resource discovery. In *Proc. of PerCom*, 2002.

[11] K. Chen, S. H. Shah, and K. Nahrstedt. Cross-layer design for data accessibility in mobile ad hoc networks. *Wireless Personal Communications: An International Journal*, 21(1):49–76, Apr. 2002.

[12] L. Cheng. Service advertisement and discovery in mobile ad hoc networks. In *Proc. of CSCW*, 2002.

[13] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz. An architecture for a secure service discovery service. In *Proc. of MobiCom*, 1999.

[14] S. R. Das, C. E. Perkins, and E. E. Royer. Performance comparison of two on-demand routing protocols for ad hoc networks. In *Proc. of INFOCOM*, pages 3–12, 2000.

[15] R. Droms. Dynamic host configuration protocol. RFC 2131, Mar. 1997. `http://www.faqs.org/rfcs/rfc2131.html`.

[16] K. Fall and K. Varadhan. ns notes and documentation. The VINT Project, UCBerkeley, LBNL, USC/ISI, and Xerox PARC, Nov. 1997. Available at http://citeseer.nj.nec.com/fall00ns.html.

[17] M. Flynn, D. Pendlebury, C. Jones, M. Eldridge, and M. Lamming. The satchel system architecture: mobile access to documents and services. *Mobile Networks and Applications*, 5(4):243–258, 2000.

[18] Y. Y. Goland, T. Cai, P. Leach, Y. Gu, and S. Albright. Simple service discovery protocol. IETF Internet Draft draft-cai-ssdp-v1-03, Oct. 1999.

[19] R. A. Golding. A weak-consistency architecture for distributed information services. *Computing Systems*, 5(4):379–405, 1992.

[20] E. Guttman. Service location protocol: automatic discovery of IP network services. *IEEE Internet Computing*, 3(4):71–80, July 1999.

[21] R. Handorean and G.-C. Roman. Service provision in ad hoc networks. In *Proc. of Coordination*, 2002.

[22] S. Helal, N. Desai, V. Verma, and C. Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. In *Proc. of WCNC*, 2003.

[23] D. B. Johnson and D. A. Maltz. Dynamic source routing in ad hoc wireless networks. In T. Imielinski and H. Korth, editors, *Mobile Computing*, volume 353, chapter 5, pages 153–181. Kluwer Academic Publishers, 1996.

[24] T. Kindberg, J. Barton, J. Morgan, G. Becker, D. Caswell, P. Debaty, G. Gopal, M. Frid, V. Krishnan, H. Morris, J. Schettino, B. Serra, and M. Spasojevic. People, places, things: web presence for the real world. *Mobile Networks and Applications*, 7(5):365–376, 2002.

[25] M. Klein and B. Konig-Ries. Multi-layer clusters in ad-hoc networks - An approach to service discovery. In *Proc. of the Workshop on Peer-to-Peer Computing*, 2002.

[26] R. Koodli and C. E. Perkins. Service discovery in on-demand ad hoc networks. IETF Internet Draft draft-koodli-manet-servicediscovery-00.txt, Oct. 2002.

[27] U. C. Kozat and L. Tassiulas. Network layer support for service discovery in mobile ad hoc networks. In *Proc. of INFOCOM*, 2003.

[28] J. Mingliang, L. Jinyang, and Y. Tay. Cluster-based routing protocol (cbrp). IETF Internet Draft draft-ietf-manet-cbrp-spec-00.txt, Aug. 1999.

[29] C. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. In *SIGCOMM*, pages 234–244, 1994.

[30] C. E. Perkins and E. M. Royer. Ad hoc on demand distance vector routing. In *WMCSA*, 1999.

[31] V. T. Raisinghani and S. Iyer. Cross-layer design optimizations in wireless protocol stack. *Computer Communications*, 2003.

[32] B. Raman, P. Bhagwat, and S. Seshan. Arguments for cross-layer optimizations in bluetooth scatternets. In *Proc. of SAINT*, pages 176–184, 2001.

[33] O. Ratsimor, D. Chakraborty, S. Tolia, D. Kushraj, A. Kunjithapatham, G. G. A. Joshi, and T. Finin. Allia: Alliance-based service discovery for ad-hoc environments. In *ACM Mobile Commerce Workshop*, 2002.

[34] A. C. Snoeren and H. Balakrishnan. An end-to-end approach to host mobility. In *Proc. of MobiCom*, 2000.

[35] M. Storey, G. Blair, and A. Friday. Mare: Resource discovery and configuration in ad hoc networks. *Mobile Networks and Applications*, 7(5):377–387, Oct. 2002.

[36] J. B. Tchakarov and N. H. Vaidya. Efficient content location in mobile ad hoc networks. In *Proc. of MDM*, 2004.

[37] The CMU Monarch project. wireless and mobility extensions to ns-2, Oct. 1999.

[38] X. Wang and H. Schulzrinne. An integrated resource negotiation, pricing, and qos adaptation framework for multimedia applications. *IEEE Journal on Selected Area in Communications*, 18, 2000.