A Cross-Layer Approach to Service Discovery and Selection in MANETs

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

***Abstract*— When a service is offerend by multiple servers in a Mobile Ad Hoc Network (MANETs), the manner in which clients and servers are paired together, referred to as *service selection*, is crusial to network performance. Good service selection groups clients with nearby servers, localizing communication, which in turn reduces inter-node interference and allows for multiple concurrent transmissions in different parts of the network.**

Although much previous research has concentrated on service discovery in MANETs, not much effort has gone into understanding the effects of service selection. This paper demonstrates that service selection in MANETs has profound implications for network performance. Specifically, we show that effective service selection can improve network throughput by up to 400%. We show that to maximize performance service selection decisions need to be continuously reassessed to offset the effects of topology changes. We argue that effective service selection in MANETs requires a cross-layer approach that integrates service discovery and selection functionality with network ad hoc routing mechanisms. The cross-layer approach leverages existing routing traffic and allows clients to switch to better servers as network topology changes.

I. Introduction

A multi-hop mobile ad hoc network (MANET) consists of a group of mobile wireless nodes that self-configure to communicate information beyond their individual transmission range by routing packets over intermediate nodes [1], [2], [3]. 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 anticipate that MANETs will be used to access services, and that services are likely to be replicated across a number of nodes. For example, imagine a network consisting of soldiers equipped with video cameras and vehicles equipped with long-range radios (i.e., services) that allow communication between the soldiers and their commanders at the control center. The cameras capture images and transmit them to any of the radios, which forward the images to the control center. Another example is a mobile sensor network consisting of light moving sensors that collect observations and more computationally-rich nodes capable of transmitting the observations to the base station. Delivering observations to one of the computationally-rich nodes is sufficient to accomplish the task.

While much previous work has concentrated on service discovery in MANETs, not much effort has gone into understanding the performance implications of *service selection*, the manner in which clients and servers are paired together when multiple nodes in the MANET provide the same (or equivalent) service.

This paper shows that service selection has a crucial effect on network capacity. Because of the broadcast nature of wireless transmissions, the communication pattern affects the number of concurrent transmissions that the network can sustain. Good service selection localizes communication, which reduces 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. The optimality of service selection in MANETs degrades rapidly as a result of node mobility and the arrival and departure of nodes that host services. Therefore, service selection decisions in MANETs have to be continuously reevaluated to optimize network performance.

We argue that effective service selection in MANETs requires the cross-layer [4], [5] integration of service discovery and selection functionality with MANET routing mechanisms. A cross-layer service discovery and service selection approach has two significant advantages over traditional application-layer implementations that preserve the modularity of the networking stack. First, because it leverages the routing mechanisms for service discovery and service selection, clients learn about available servers and routes to them simultaneously. This routing information greatly reduces the cost and increases the accuracy of service selection. More importantly, the availability of explicit routing information, such as route breaks or updates, enables clients to efficiently detect changes in network topology and switch to closer servers without additional cost.

We make the following contributions: (a) We present a cross-layer service discovery and service selection architecture that provides for efficient and timely reevaluation of service selection decisions; (b) We present experimental results that show that cross-layer service selection implementations based on the DSR [1] and DSDV [2] routing protocols consistently outperform application-layer implementations that closely models the Service Location Protocol (SLP) [6]. 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; (c) We show that due to interference from the underlying routing

traffic, application-layer timing-based mechanisms for service selection, such as pinging, are highly inaccurate and fail to localize communication.

In this paper, we focus on the problem of timely and efficient service selection. We do not, however, address the issue of migrating application [7] and connection state [8] between servers. Instead, we consider a best case scenario where there is no migration overhead, and clients can switch between servers based solely on network proximity. Therefore, our results represent an upper bound on the achievable network performance.

The rest of this paper is organized as follows. Section II describes our cross-layer architecture for service discovery and service selection. Section III describes two prototype implementations of our cross-layer architecture based on the DSR and DSDV routing protocols. Section IV presents experimental results. We describe related works in Section V, and discuss our conclusions in Section VI.

II. Architecture

Several factors have to be considered in the design of a general architecture for cross-layer service discovery and service selection. On the one hand, a cross-layer implementation has to be closely coupled to the underlying routing mechanisms to exploit routing traffic. On the other hand, the functionality that the 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, and 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 service discovery and selection, but leave it to an application to determine how to select servers and when and how to reevaluate its choices.

These considerations 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 stores information about known servers in a *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,

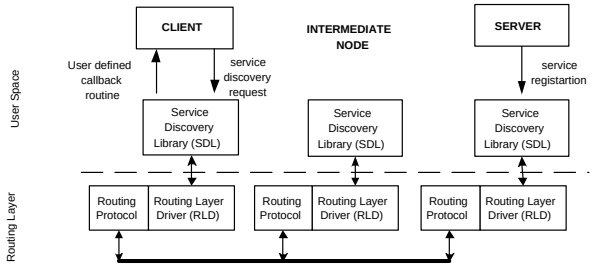

Fig. 1. Cross-Layer Service Discovery Architecture

follow well-defined interfaces. Clients and servers call on SDL to issue service discovery requests and propagate service advertisements. 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 intercepts from the network to SDL and informs it about changes in network topology.

A. 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 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 built 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 [9] would unicast discovery requests to a cluster-head.

Passive discovery, in contrast, is server driven. In passive discovery, SDL instructs RLD to periodically disseminate advertisements for a specific service. 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*. If SDL finds a match, it instructs RLD to compose a service response. Otherwise, it instructs RLD to rebroadcast

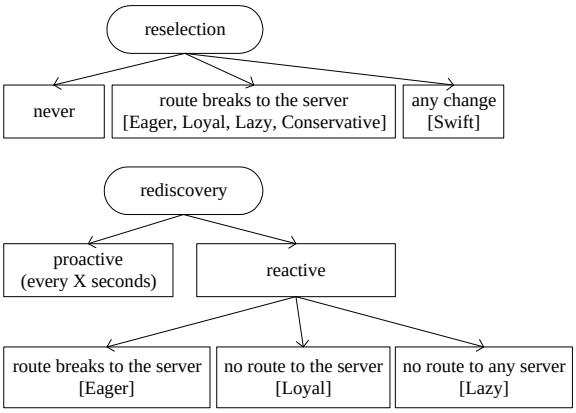

Fig. 2. Reselection and rediscovery policies.

the request. The actual mechanisms used by RLD to propagate service responses or forward service requests depends on the underlying routing protocol.

B. 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. We expect that in most cases, clients will choose the server with the lowest hop count; however, other service or routing specific metrics can be used in service selection (e.g., choosing a server that has the least load).

C. Reevaluation

To optimize performance, MANET clients need to constantly reevaluate their choice of service provider. Reevaluation has two components: *reselection* and *rediscovery*. Reselection reconsiders service 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 implementations that support active discovery. Rediscovery is usually followed by reselection.

In designing a reevaluation 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 policies we have implemented as part of our cross-layer prototypes. We discuss these policies in more detail in Section III.

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 end 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[1].

Applications may choose to trigger *rediscovery* either proactively or in reaction to a change in the service table. 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 a policy we found works well in practice triggers a rediscovery after trying all known routes to the current server.

III. Prototypes

We implemented two prototype of our cross-layer architecture based on the DSR [1] and DSDV [2] routing protocols. We refer to these prototypes as CL_{dsr} and CL_{dsdv}, respectively. In both prototypes, matching of service descriptions with discovery requests is done using a simple string based comparison, and clients choose the server with the lowest hop count.

The rest of this section describes CL_{dsr} and CL_{dsdv}. For each prototype, we first describe the underlying routing algorithm. We then describe our extensions for service discovery and selection.

A. CL_{dsr}

Dynamic Source Routing (DSR) is an on-demand source routing protocol. DSR has two operation modes: *route discovery* and *route maintenance*. Whenever a node sends a packet, 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 is used. DSR reduces the latency and frequency of route discoveries by allowing intermediate nodes to cache overheard routes and respond to route requests with routes stored in their cache.

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. When a route breaks, 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.

[1]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.

1) Extensions: We extended the existing DSR route request/reply mechanisms to perform service discovery and service selection 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 requests 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 and the service discovery flag set.

RLD examines all received packet and delivers service discovery packets to the local SDL. In turn, 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 the 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.

Following DSR's idiosyncrasy, CL_{dsr} allows intermediate nodes to (i) learn about available services by overhearing service discovery reply messages, and (ii) reply to service discovery requests, by checking their local service table and responding if a match is found. If not used or updated, service table entries are periodically invalidated.

We developed 3 reevaluation policies for CL_{dsr}: *Eager*, *Loyal*, and *Lazy*. 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 (at this point DSR would normally trigger a route discovery). 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. Figure 2 shows the place these policies occupy in the design space of reevaluation policies.

Figure 3 illustrates the behavior of the CL_{dsr} reevaluation 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 two routes to server S1, one route to server S2 and no routes to server S3. Let us assume that 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),

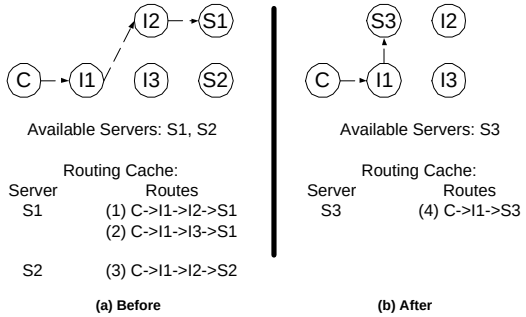

Fig. 3. Rediscovery example

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.

B. CL_{dsdv}

Destination Sequence Distance Vector (DSDV) is a table-driven proactive protocol, where every node has a routing table with entries for all other nodes in the network. Each routing entry includes a destination's address, the next hop to the destination, a sequence number and a metric (usually route length). Nodes exchange route entries periodically and on learning of a newer and shorter route.

1) Extensions: 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* and *Conservative*. Both policies chose initially the server with the shortest hop count. Swift switches servers as soon it becomes aware of a server with a smaller hop count. In contrast, Conservative switches servers only when there is no longer a valid route to the current server. Figure 2 shows the place these policies occupy in the design space of reevaluation policies.

IV. Evaluation

In this section, we compare the performance of our CL_{dsr} and CL_{dsdv} implementations to the performance of application-layer implementations modeled after Service Location Protocol (SLP) [6] running on top of unmodified versions of DSR and DSDV.

This rest of this section is structured as follows. We first describe the SLP protocol. We then describe our evaluation environment. Finally, we present our experimental results.

A. Service Location Protocol

SLP is an IETF application-layer service discovery standard that supports both centralized and distributed operating modes. 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 services that match the client's request. In the distributed mode, no DAs are present, and clients query servers directly by broadcasting service discovery requests[2]. Matching servers then reply by unicasting their responses back to the clients.

The SLP standard does not cover service selection, leaving this decision entirely to the client. We consider two approaches to service selection: **RD**, which picks one of the available servers at random, and **PING**, which tries to select the closest server by unicasting ping messages to all known servers and choosing the server whose ping reply arrives first[3]. Because the optimality of service selection in MANET degrades as a result of mobility, PING probes the network periodically by sending ping messages to known servers, and switches the client to the server with the lowest round-trip-time. Ideally, PING's probing rate should closely match the rate of changes in the network topology. However, given that explicit topology-change information, such as route break events, is not available at the application-layer, the optimal probing rate has to be estimated empirically. We show in Section IV-C that determining the optimal probing rate is not trivial.

Although we have implemented both centralized and distributed versions of SLP, we do not consider the distributed version of the algorithm further in this paper. This is because both centralized and distributed versions use the same service selection algorithms, and therefore their performance beyond the initial service discovery phase is virtually identical. Finally, we identify the routing algorithm over which SLP operates with a subscript (e.g., RD_{dsr} stands for the centralized version of SLP running on top of the DSR protocol that selects a server at random).

B. Evaluation Environment

We ran our experiments on the ns-2 simulator with CMU wireless extension [11]. We report results for a network of 100 nodes randomly placed on a rectangular 300m x 2000m flat space. Similarly to [12], [13], the rectangular shape was chosen to force the use of longer routes between communication pairs. Each node is equipped with a WaveLAN radio with a 250m nominal transmission range and a raw capacity of 2Mb/s. Clients and servers move following the random waypoint model [12] with no pause time to stress test the implementations.

Clients communicate with servers by sending 100-byte packets at a constant bit rate (CBR) of 7.5 packets/sec. Similarly to [12], we chose not to use TCP communication to allow more accurate comparison between protocols[4]. We vary the number of servers (1 to 4) and the maximum node speed (2m/s and 20m/s). All results are averages over 5 movement scenarios. All servers in the network offer the same service.

To eliminate transient initializations effects, clients start sending data after a 100 sec. initialization phase where all clients discover all available servers, make an initial service selection and obtain at least one route to the selected server. Motion starts at 200 sec., and the experiments ends at 800 sec.

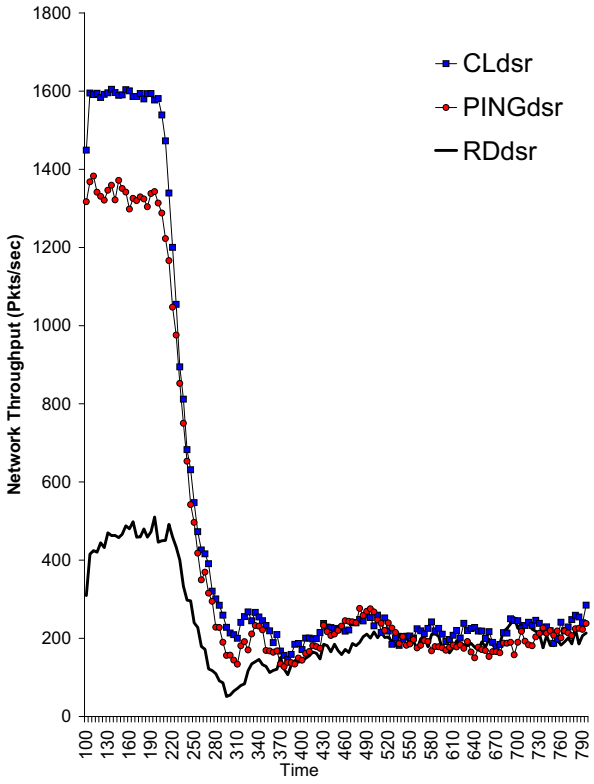

Fig. 4. **[DSR]** Mobility degrades service selection optimality, and in turn, overall network throughput (4 servers, 50 clients)

C. Experimental Results

Figure 4 plots the network throughput over time after the 100-second initialization phase for CL_{dsr} , RD_{dsr} and PING_{dsr} for 4 servers and 50 clients. Nodes move with a maximum speed of 20 m/sec. The static phase of the simulation (up to the 200 sec. mark) shows that service selection has a critical effect on overall network performance, with both CL_{dsr} and PING_{dsr} achieving more than 3 times the network throughput of RD_{dsr}. There is a strong correlation between the average path length between clients and servers and network throughput. The average route lengths for CL_{dsr} PING_{dsr} and RD_{dsr} are 2.67, 3.02, and 5.02 hops, respectively. The average optimal route length (defined as the shortest route between a client and the closest server) is 2.47. CL_{dsr} picks

[2]Whereas the SLP standard includes support for multicasting, we opted instead for an implementation based on application-layer broadcast to avoid the high cost associated with maintaining multicast groups in MANETs [10].

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

[4]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.

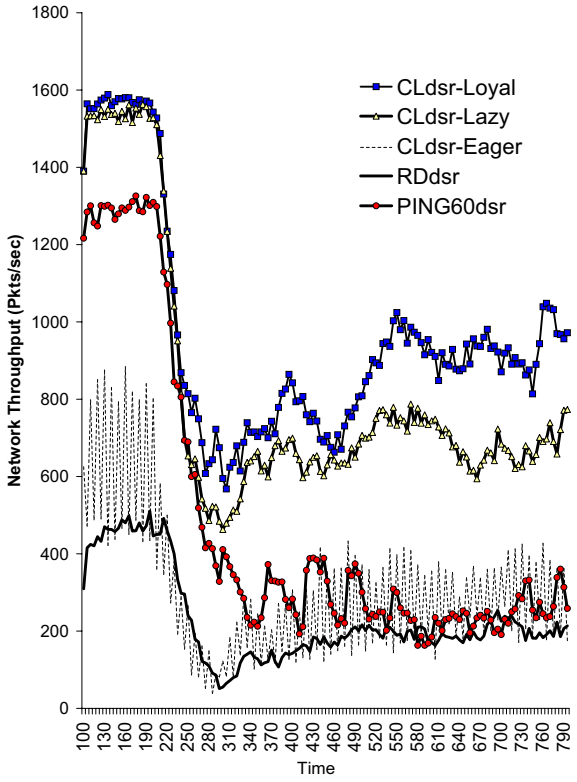

Fig. 5. **[DSR]** Effectiveness of reevaluation policies (4 servers, 50 clients).

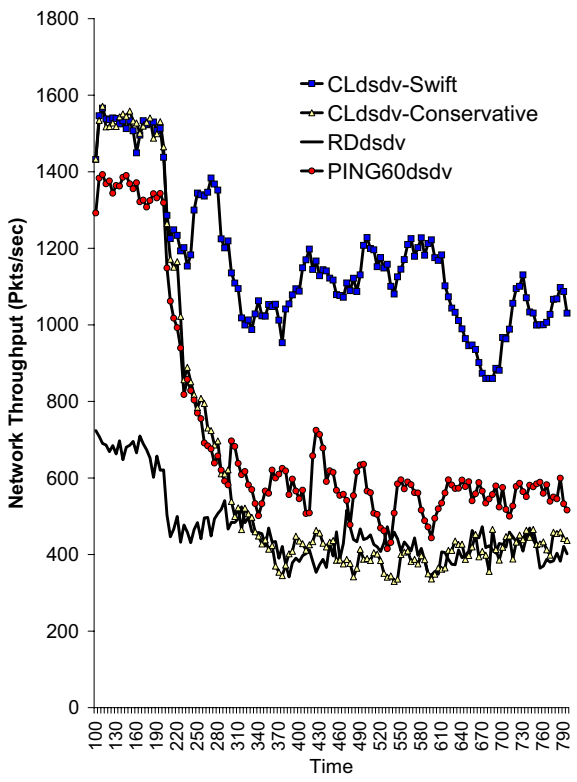

Fig. 6. **[DSDV]** Effectiveness of reevaluation policies (4 servers, 50 clients).

optimal routes most of the time. The difference that exists between CL_{dsr} and optimal route length results from packet losses due to collisions that lead CL_{dsr} to choose sub-optimal routes. PING_{dsr} has a longer average route because it performs service selection based on limited knowledge of the network topology. PING_{dsr} attempts to find the closest server by measuring the round-trip time between the client and all known servers. Unfortunately, the time to unicast a message from the client to a server and back varies greatly based on the availability of DSR routes. As a result, a remote server for which the client caches a valid route may reply ahead of a closer server for which the client will have to discover a route. The RD_{dsr} picks servers randomly and is obviously the worst case. Therefore, we conclude that near-optimal service selection based on route length clusters clients and servers together, localizing communication and increasing network capacity. Poor service selection, on the other hand, results in interference that severely limits network throughput.

Once nodes begin to move (200 sec. mark), however, the optimality of service selection degrades rapidly, which causes all protocols to converge to the same throughput. This is an expected result, as the initial service selection becomes irrelevant after a period of movement. Note that in this scenario, clients can still communicate with the servers, albeit less efficiently. Experiments with maximum node speed of 2 m/sec present a similar trend (not shown), 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. Experiments with CL_{dsdv} and SLP over DSDV present similar trends, and are therefore not shown.

Figures 5 and 6 compare the performance of the service selection reevaluation policies we implemented for CL_{dsr} and CL_{dsdv} to PING60 and RD. PING60 is a version of PING that pings all known servers every 60 sec., and selects the one with the shortest round-trip time. RD 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 maximum speed of 20 m/sec.

For CL_{dsr}, *Loyal* achieves the best performance, validating the DSR policy of trying all cached routes to a given node before issuing a route discovery. *Eager* suffers from network congestion as a result of sending discovery requests as soon as the active route breaks. *Lazy*, on the other hand, is too conservative and ends up trying (unsuccessfully) a large number of stale routes before finally sending a discovery request. *Lazy* also chooses longer routes over initiating a discovery request leading to poor locality.

For CL_{dsdv}, *Swift* achieves the best performance. Since no additional packets are being transmitted into the network, *Swift* does not incur any penalty for switching to a better server when one becomes available. *Conservative*, on the other hand, waits for a route break and thus misses opportunities of switching to better servers.

PING60 shows (in both its DSR and DSDV variants) that service selection based on timing measurements, such as pinging, increases network throughput. However, a substantial gap still remains between the best cross-layer reevaluation technique, and the application-layer implementation. Two fac-

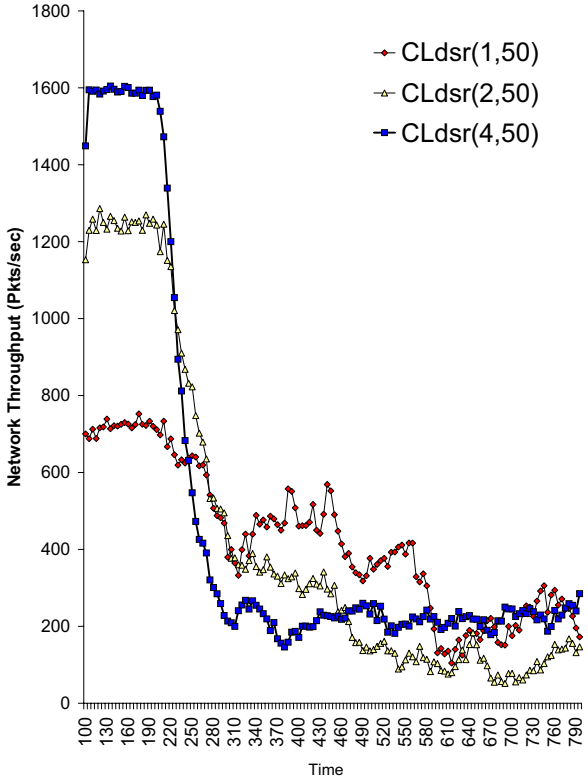

Fig. 7. **[DSR]** Effects of mobility on throughput with different number of servers (1, 2 and 4 servers, 50 clients)

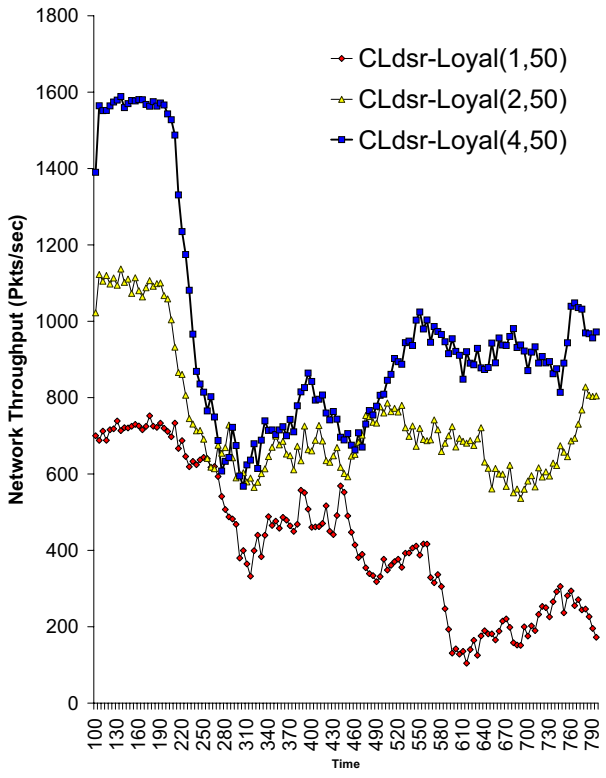

Fig. 8. **[DSR]** Effects of reevaluation on throughput with different number of servers (1, 2 and 4 servers, 50 clients)

tors account for PING60's 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 PING$_{dsr}$ to the reevaluation period, we varied the pinging period between 30 and 90 sec. While the 60 second pinging period achieves the best throughput, overall network throughput was not significantly affected.

Figure 7 presents an interesting result. The figure plots the network throughput over time for CL$_{dsr}$ for networks with 1,2, and 4 servers and 50 clients. 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. While the average path length to a server degrades with mobility to the same value independent of the number of servers, the increased number of servers in the network causes more interference between client-server pairs. Therefore, we conclude that for mobile networks where the service selection is not reevaluated (and where server processing capacity in not the bottleneck), decreasing the number of servers can actually increase network performance. This result is not specific to CL$_{dsr}$. The graphs for RD$_{dsr}$ and PING$_{dsr}$ (not shown) present a similar trend. It is compelling that as the number of servers in a network increases, the need to reevaluate service selection becomes more urgent.

Figure 8 plots the network throughput over time for CL$_{dsr}$-Loyal (the best-performing CL$_{dsr}$ variation) with the same network configuration as in Figure 7. In contrast to CL$_{dsr}$, 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.

V. Related Work

There has been significant research on application-layer service discovery solutions for MANET. Tchakarov *et al.* [14] propose a resource location protocol for multi-hop ad hoc networks that uses geographical information to reduce service advertisement and discovery traffic. The Intentional Naming System (INS) [15] 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 [16] have extended INS with context-sensitive service discovery and Bisdikian *et al.* [17] 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 [18] and Multi-Layer Clusters [19]. Azondekon *et al.* [20] 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.

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

Cross-layer design for MANETs has been an active topic of research in the recent years. Shakkottai *et al.* [5] suggest using cross-layer design to improve performance in MANETs. Conti *et al.* [4] propose a cross-layer design that maintains the layering principle. Carter *et al.* [21] show that the application-layer communication on top of routing protocols suffer from a large control message overhead and argue for a need of a routing-layer support for group communications. Kozat and Tassiulas [10] 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. Raman *et al.* [22] 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.* [23] propose extensions to MANET routing protocols to support service discovery. Cheng [24] 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 creation and maintenance of multicast groups requires significant control message overhead [10].

In contrast to our work, these efforts focus on initial service discovery, and do not consider the problem of reevaluating service selection, which we have shown to be fundamental for good performance in MANETs.

VI. Conclusions and Future Work

We presented a new cross-layer architecture that integrates service discovery and service selection functionality with existing routing protocols, thus allowing nodes to learn about available servers and routes to them simultaneously. Our cross-layer approach helps clients select the best possible server, and allows them to register call-back routines to be notified once a better server is available nearby.

We explored the benefits of cross-layer service selection over a typical application-layer implementation. The experimental results show that the cross-layer implementation succeeds in localizing communication in the presence of topology changes, and achieves up to five times higher network throughput than a typical application-layer solution.

We showed that periodic application-level timing-based mechanisms for service selection, such as pinging, add significant amount of overhead to an already congested network and fail to capture the rate of network topology change.

In the future, we plan to extend our technique to support other routing protocols (e.g., AODV), as well as session migration of stateful services.

References

[1] David B Johnson and David A Maltz, "Dynamic source routing in ad hoc wireless networks," in *Mobile Computing*, Tomasz Imielinski and Hank Korth, Eds., vol. 353, chapter 5, pp. 153–181. Kluwer Academic Publishers, 1996.

[2] Charles Perkins and Pravin Bhagwat, "Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers," in *SIGCOMM*, 1994, pp. 234–244.

[3] Charles E. Perkins and Elizabeth M. Royer, "Ad hoc on demand distance vector routing," in *WMCSA*, 1999.

[4] Marco Conti, Gaia Maselli, Giovanni Turi, and Silvia Giordano, "Cross-layering in mobile ad hoc network design," *IEEE Computer*, 2004.

[5] S. Shakkottai, T. S. Rappaport, and P. C. Karlsson, "Cross-layer design for wireless networks," *IEEE Communications Magazine*, Oct. 2003.

[6] Erik Guttman, "Service location protocol: automatic discovery of IP network services," *IEEE Internet Computing*, vol. 3, no. 4, pp. 71–80, July 1999.

[7] Richard A. Golding, "A weak-consistency architecture for distributed information services," *Computing Systems*, vol. 5, no. 4, pp. 379–405, 1992.

[8] Alex C. Snoeren and Hari Balakrishnan, "An end-to-end approach to host mobility," in *Proc. of MobiCom*, 2000.

[9] Jiang Mingliang, Li Jinyang, and Y.C. Tay, "Cluster-based routing protocol (cbrp)," IETF Internet Draft draft-ietf-manet-cbrp-spec-00.txt, Aug. 1999.

[10] Ulas C. Kozat and Leandros Tassiulas, "Network layer support for service discovery in mobile ad hoc networks," in *Proc. of INFOCOM*, 2003.

[11] "Wireless and mobility extensions to ns-2," Oct. 1999.

[12] Josh Broch, David A. Maltz, David B. Johnson, Yih-Chun Hu, and Jorjeta Jetcheva, "A performance comparison of multi-hop wireless ad hoc network routing protocols," in *Proc. of MobiCom*, 1998.

[13] Samir Ranjan Das, Charles E. Perkins, and Elizabeth E. Royer, "Performance comparison of two on-demand routing protocols for ad hoc networks," in *Proc. of INFOCOM*, 2000, pp. 3–12.

[14] Jivodar B. Tchakarov and Nitin H. Vaidya, "Efficient content location in mobile ad hoc networks," in *Proc. of MDM*, 2004.

[15] William Adjie-Winoto, Elliot Schwartz, Hari Balakrishnan, and Jeremy Lilley, "The design and implementation of an intentional naming system," in *Proc. of 17th SOSP*, 1999.

[16] Guanling Chen and David Kotz, "Context-sensitive resource discovery," in *Proc. of PerCom*, 2002.

[17] Chatschik Bisdikian, Isaac Boamah, Paul Castro, Archan Misra, Jim Rubas, Nicolas Villoutreix, Danny Yeh, Vladimir Rasin, Henry Huang, and Craig Simonds, "Intelligent pervasive middleware for context-based and localized telematics services," in *Proc. of Workshop on Mobile Commerce*, 2002.

[18] Dipanjan Chakraborty, Anupam Joshi, Yelena Yesha, and Tim Finin, "GSD: A novel group-based service discovery protocol for manets," in *Proc. of MWCN*, 2002.

[19] Michael Klein and Birgitta 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.

[20] Victor Azondekon, Michel Barbeau, and Ramiro Liscano, "Service selection in networks based on proximity confirmation using infrared," in *Proc. of ICT*, 2002.

[21] Casey Carter, Seung Yi, Prashant Ratanchandani, and Robin Kravets, "Manycast: Exploring the space between anycast and multicast in ad hoc networks," in *Proc. of MobiCom*, 2003.

[22] Bhaskaran Raman, Pravin Bhagwat, and Srinivasan Seshan, "Arguments for cross-layer optimizations in bluetooth scatternets," in *Proc. of SAINT*, 2001, pp. 176–184.

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

[24] Liang Cheng, "Service advertisement and discovery in mobile ad hoc networks," in *Proc. of CSCW*, 2002.