Service Discovery using OLSR and Bloom Filters

Joakim Flathagen
Norwegian Defence Research Establishment
BX 25, N-2027 Kjeller, Norway
Email: joakim.flathagen@ffi.no

Knut Øvsthus
Bergen University College
BX 7030, N-5020 Bergen, Norway
Email: knut.ovsthus@hib.no

***Abstract*—Automatic discovery of services and resources is a crucial feature to achieve the expected user-friendliness in Mobile Ad-hoc Networks. Due to limited computing power, scarce bandwidth, high mobility and the lack of a central coordinating entity, service discovery in these networks is a challenging task.**

In this paper, we develop a service discovery protocol (Mercury) utilizing a combination of different optimization techniques: The performance is increased using cross-layer interaction between the application layer and the routing layer. The service information is described using Bloom filters and distributed using Optimized Link State Routing (OLSR). A caching regime is implemented to obtain further reductions of both overhead and latency.

The analysis and simulation results show that our service discovery proposal induces very low overhead to OLSR and is superior to application layer solutions. The proposal is implemented as a plugin to the OLSR implementation *olsrd* for real-world deployments.

***Index Terms*—MANET, OLSR, Service discovery**

I. Introduction

A Mobile Ad-hoc NETwork (MANET) is a collection of mobile nodes connected by wireless links able to dynamically form an arbitrary multihop network—without the use of any pre-existing infrastructure. In order to enable communication between any two nodes in such a network, a special routing protocol is employed. The IETF MANET working group mainly considers two routing approaches: *Reactive routing* such as AODV [18] and *Proactive routing* such as OLSR [5].

However, there is a need for a *service discovery protocol* to discover applications, services and resources in the network. There has been much research activity in the field of service discovery by several consortiums, companies and organizations. This research has produced service discovery mainly for fixed local area networks. Examples include Service Location Protocol (SLP) [9], Simple Service Discovery Protocol (SSDP) [8], Jini [20] and DNS Service Discovery (DNS-SD) [4]. However, the overall Internet community has not yet reached a consensus on one particular service discovery protocol. Moreover, none of the above solutions are applicable to MANETs without adaptations as these networks have less computing resources, lower network bandwidth, higher mobility and more heterogeneity.

Service discovery (SD) mechanisms for MANETs are divided in two groups: (1) mechanisms *independent* of the underlying routing protocol, and (2) mechanisms *integrated* with the routing protocol, be it either *reactive* or *proactive*.

Most of the MANET SD proposals belong to the first category and solves the SD at a layer above routing—referred to as *application layer service discovery*. Examples include SLPManet [1], PDP [3] and Konark [10], which all rely on multicast support on the network layer. The performance of such SD protocols is therefore bound to the chosen multicast protocol. Further, multicast in MANETs is still at the research stage (no standard is defined) and is hence an open issue.

A better and more optimized approach is therefore to implement the SD protocol in a *cross-layer* fashion, and exploit the routing layer for efficient dissemination of service control messages. SEDRIAN [16] and the work by Engelstad et al. [6] propose cross-layer service discovery utilizing AODV. Jodra et al. [11] and Lightweight Service Discovery (LSD) [12] are examples of cross-layer service discovery using OLSR.

Differing from previous work on cross layer service discovery based on OLSR, this paper focus to support low-bandwidth environments and investigates an efficient way to describe services using Bloom filters combined with service caching. The analysis and simulation results show that our optimized SD proposal named Mercury, induces very low overhead to OLSR and outperforms application layer SD solutions. The proposal is implemented for real-world deployments [7].

The remainder of this paper is organized as follows: Section II presents Mercury service discovery protocol in detail. Section III describes the real-world implementation. Section IV and V presents and discusses the simulations. Finally, the paper is concluded by section VI.

II. Our Service discovery design

A. Overview

To successfully create service discovery for bandwidth-constrained environments, we envision several combined optimizations. For this purpose, we propose a new SD solution, Mercury. Mercury describes the service descriptors efficiently as *Bloom filters*, performs service dissemination by *piggybacking* service information on OLSR routing messages and utilizes *caching* of service advertisements.

Mercury handles requests and advertisements from two entities: (1) Local applications on the node and (2) foreign nodes through the ad hoc network (Fig. 1). Each node uses a set of repositories to store the information (Fig. 3): **Advertised services** contains the different services offered by the node itself. The services persist in this list until an upper layer application withdraws the service. Advertisements are sent

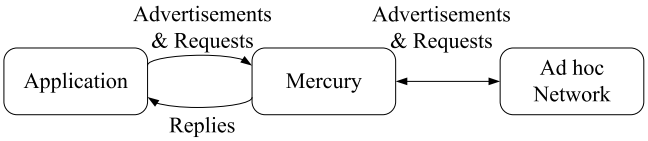

Fig. 1. Mercury connects users and applications to services in the Ad hoc network using service advertisements and service requests.

both when a service is first registered and upon an external request. All the service descriptors in this list are included in advertisemens encoded as one single Bloom Filter. In **Foreign services cache**, all the services offered by other nodes are stored. Each entry in the list consists of the Bloom Filter advertised by the foreign node and its current IP address. The last repository contains the **Requested services** which stores all the services requested—awaiting an incoming advertisement.

All incoming advertisements are immediately stored in the cache. Upon a request from an upper layer application, the cache is first requested. If an entry is found, the application is immediately notified. Otherwise, a service request is sent.

B. Protocol format

OLSR communicates using a unified packet format for all data [5]. Using this format the OLSR standard provides extensibility of the protocol without breaking backwards compatibility. This feature gives a unique possibility to disseminate different kinds of information through intermediate nodes even if the nodes do not support the specific extension.

We take advantage of the extensibility feature of the OLSR format, and introduce a new message, namely the Mercury service discovery message (MSD). MSD messages are sent as the data-portion of the general message format with the message type set to MSD_MESSAGE. The MSD message has the format specified in Fig. 2 when piggybacked to an OLSR header. The Mercury part consists of four fields including a **Spare** field for future use. The **Type** field indicates whether the message is a service request or a service reply. The **Service Filter** field contains the filter describing the services to be requested or advertised encoded as a Bloom filter (described subsequently). The **Filter Length** gives the size of the filter.

C. Distributing service descriptors

Many service discovery protocols use XML to describe the service information, such as in [10]. However, XML requires considerable bandwidth, which is sparse in ad hoc networks. An alternative is to map a predefined set of keywords, or service descriptors, to integers to save bandwidth as proposed in [11]. This solution indeed saves bandwidth. However, it is not very flexible nor is it scalable, as it requires maintenance on every node in the network when new service categories are added.

The proposed solution in this paper is therefore to distribute a summary of the available services as a vector described as a *Bloom filter* [2]. A Bloom filter is a data structure that allows data representation in a simple and space-efficient manner.

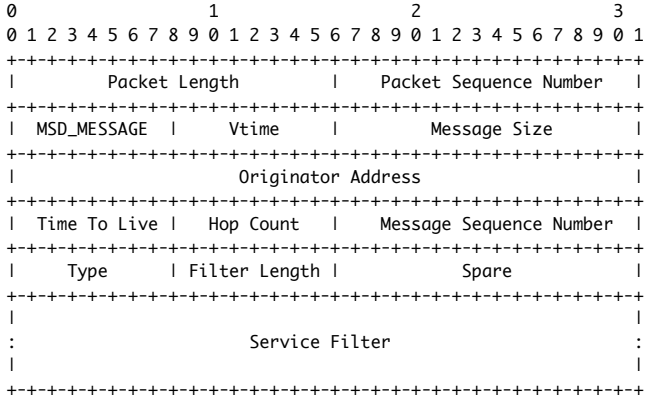

Fig. 2. Mercury service discovery format as an extension to the OLSR message format [5]

The filter is created by hashing service descriptors to a size-defined bit array. The size limitation may cause the filter to indicate that a service descriptor is in the filter even though it is not—referred to as a *false positive*. The implementation of the Bloom filter is hence a trade off between the size of the filter and the probability of a false positive request to the filter. Our Bloom filter is implemented using k independent hash functions to hash each service descriptor to the array. Given the number of service descriptors n and the filter width m, the probability of a false positive lookup can be given as:

$$P_n = \left(1 - \left(1 - \frac{1}{m}\right)^{kn}\right)^k \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \qquad (1)$$

In order to minimize the false positive rate, the filter width should mathematically be as large as possible. However, the feasible size is limited by computation time, OLSR packet size and memory consumption. The optimal value of k can be calculated by taking the derivate of equation 1. We then find that the derivate is 0 when $k = \frac{m}{n} \ln 2$, hence yielding the optimal number of hash functions for a given filter width. By having a thorough understanding of the target application the parameters k and m can be set to minimize the probability of false positive. In Mercury, the parameters are adjustable, however the default values are $k = 4$ and $m = 128$.

In Mercury the filter is created using the message digest function MD5 [19]. MD5 is a cryptographic hash function that hashes arbitrary length strings to 128 bits. The k hash functions can then be constructed from k groups of r bits each out of the 128 bit hash. The Bloom filter in Mercury is implemented as shown in algorithm 1.

Algorithm 1 is used both when services are advertised and requested. An example usage of Bloom filter based SD is shown in Fig. 3. Each node advertises two services and employs three hash functions to describe the services. After performing service requests, the descriptors are stored in the local cache of the other nodes. The cache consists of one Bloom filter for each of the cached nodes (i.e. attenuated Bloom filter).

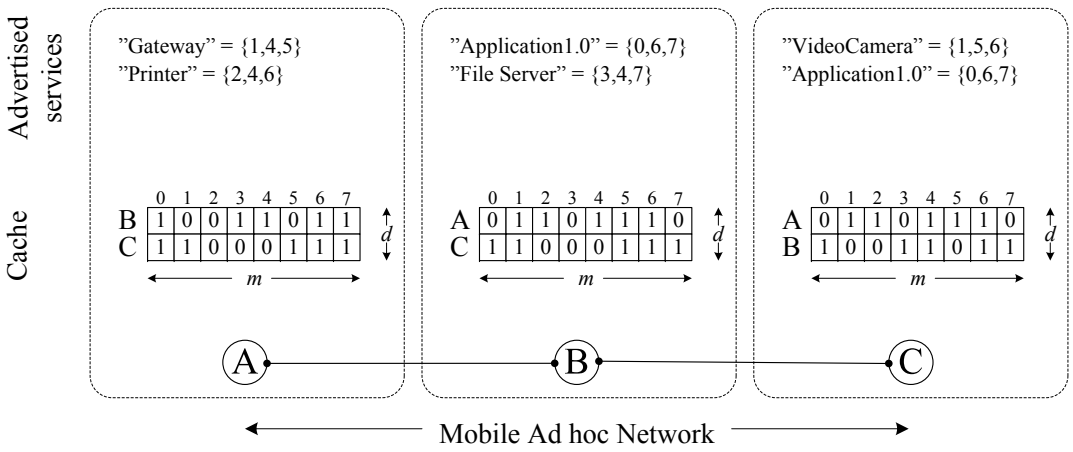

Fig. 3. A Mobile Ad hoc Network consisting of three nodes. Each node use three hash functions to create the Bloom filter and employs two repositories: One repository store the local services advertised, and one repository—implemented as an attenuated Bloom filter of depth d—serves as a cache storing advertisements received from foreign nodes.

Algorithm 1 Calculate the Bloom filter v for service x

Require: $x \neq 0$

1: $a \Leftarrow MD5(x)$
2: $r \Leftarrow 128/k$
3: **for** $i = 0$ to k **do**
4: $\quad f \Leftarrow subbits(r * i, (r * (i + 1)) - 1, a)$
5: $\quad v[f \mod m] = 1$
6: **end for**

D. Caching

Caching is employed to save network bandwidth. Caching may however, lead to false positive replies to the overlying application (Fig. 1) if the advertised service exists in cache even if the node with the advertised service is—due to network clustering—not available anymore. The cache cleanup timeout is therefore a trade-off between fast service queries and the false positive rate. To reduce the amount of false replies to the application, we propose a path-aware approach that consults the local routing table for the availability of the nodes in the cache. If a service exists is in the cache even if the node is not available, Mercury removes the cache entry and performs a new service discovery in order to find relevant nodes offering a similar service.

III. Implementation and use

The Mercury SD proposal is implemented as an extension to the UniK OLSR implementation (*olsrd*) [17]. Olsrd supports the loading of dynamically loaded libraries for auxiliary functions using a generic plugin interface [21]. Here, the Mercury plugin is briefly described and example usage is given. The code is available at [7] for further reference.

In order to allow communication between the plugin and user applications, a simple Inter-process communication (IPC) function is enabled via TCP/IP. Using IPC, services are *requested*, *advertised*, and *withdrawn* using a set of simple commands. By using Mercury and by adding only a few code lines, any distributed application can be extended to facilitate SD—regardless of programming language.

Peers [14] is a minimal SIP user agent (UA) written in Java. It enables Voice over IP services by allowing a user to call another user in the MANET using SIP. Using standard Peers, the caller is required to enter the IP address belonging to the node which it wants to call. By adding a few code lines, the application can utilize Mercury service discovery to detect the IP address of other SIP UAs automatically.

As shown, first the IPC socket is initialized. Then, two objects are created to communicate with the socket:

```
mySD = new Socket("localhost",port);
out = new PrintWriter(mySD.getOutputStream(), true);
in = new BufferedReader(new InputStreamReader(
          mySD.getInputStream()));
```

After initialization, the service "SIP" is *advertised* to inform other SIP-clients in the ad-hoc network about the existence of the UA by advertising itself (ADVR):

```
out.println("ADVR SIP");
```

The application then immediately *requests* for all other SIP UAs using the code word RQST:

```
out.println("RQST SIP ALL");
```

The application will now receive the IP addresses of all the other SIP enabled clients—immediately as they connect—via the IPC Interface (`in`). The successful discovery of other clients can then be parsed using a simple string tokenizer. By using only a few code lines, the Peers SIP software is changed to automatically detect other SIP UA in the Ad hoc network. Other existing distributed applications such as file sharing, instant messaging, whiteboard sharing, may use the same technique.

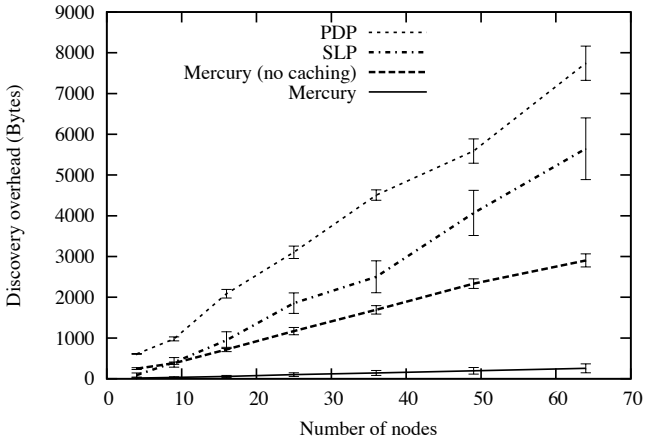

Fig. 4. Overhead using Mercury compared with SLPManet and PDP.

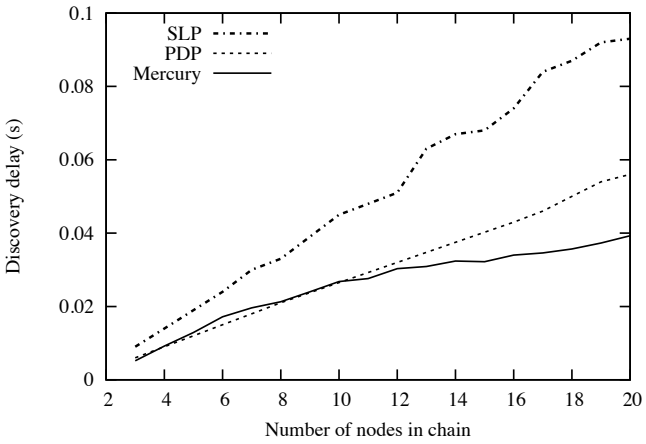

Fig. 5. The service discovery delay using Mercury compared with SLP and PDP.

IV. Performance evaluation

A. Simulation setup

The proposed service discovery mechanism is implemented in ns-2.31 [22] as an extension to UM-OLSR [23]. The transmission range is set to 100m and default OLSR parameters according to [5] are used. For Mercury, the Bloom filter size is set to 128 bits. All measurements are done after topology convergence.

To make a qualitative benchmark of the overhead induced by the service discovery process and the average time consumed when requesting a service, the Mercury protocol is compared with two widespread service discovery protocols, PDP [3] and SLPManet [1]. As both PDP and SLPManet require an underlying multicast routing protocol, our simulation of PDP and SLPManet used *nrlolsr* [15] for ns2 with the extension Simplified Multicast Forwarding (SMF) [13] used in S-MPR mode. Mercury used OLSR MPR message forwarding.

B. Overhead

To measure the overhead, we used static square topologies consisting of 4 to 64 nodes. The network had two services, located on node 0 and 1. The services were randomly requested by the other nodes with 5s intervals during the 1500s run. For each static topology, 20 simulations were run and the 95% confidence interval was estimated and presented in the figures.

Fig. 4 shows average network traffic induced by one single service discovery with increasing network size. Compared to its counterparts, the service discovery overhead is reduced by a factor of 20 when using Mercury.

C. Delay

To measure the time delay when requesting a service, a static network of nodes was chosen, and the nodes were connected in chains of 2 to 20 nodes. The only service in the network was located on node 0 and was requested by the node on the edge of the chain with 10s intervals. The delay between a service request and the successful receipt was measured. Both Mercury and SLPManet utilize local caching with 300s timeout, which reduces the average time delay. The average delay results from all topologies are given in Fig. 5. For all topologies, Mercury performs better or equal than its counterparts.

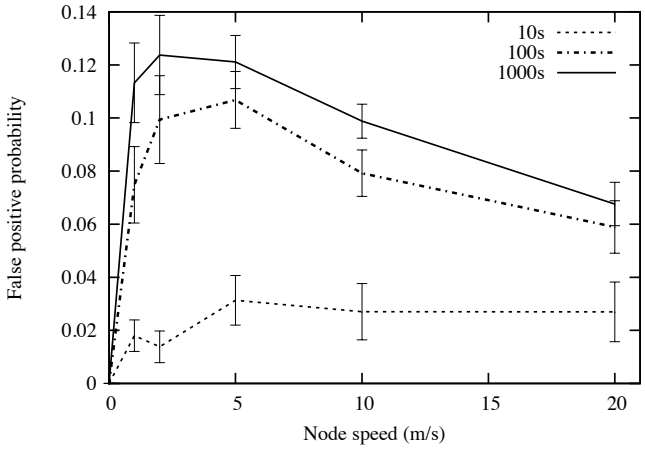

Fig. 6. False positive probability caused by caching in a dense network.

D. Path-aware algorithm

False positive replies as a side effect of caching cause unacceptable delays and reduces user satisfaction. The benefit using our path-aware caching algorithm is clearly showed by the simulations. We created two scenarios, one dense and one sparse. The dense scenario consisted of 22 nodes in a 250m x 550m area. The sparse scenario increased the area to 500m x 1000m. In both cases, the nodes followed the random waypoint model with constant speed. The nodes advertised one service each, which was randomly requested. 20 simulations were run for each combination of node speed and cache time and 95% confidence interval was estimated.

The results show the expected false probability using caching. We observe that an application requesting a service has a probability up to 12% of receiving a false positive reply when a cache timeout of 1000s is used (Fig. 6). The

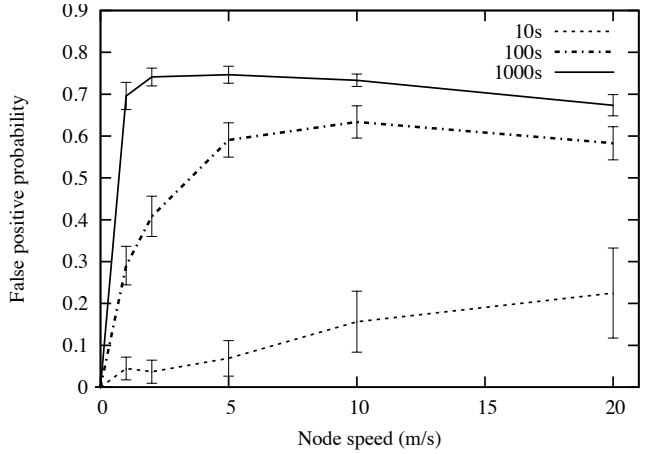

Fig. 7. False positive probability caused by caching in a sparse network.

astute reader may also observe that the false positive rate in some cases tend to decrease as the node speed increases. This phenomenon is caused by general increased node availability (more entries in the routing table) as node availability increase with increasing speed due to the nature of the random waypoint mobility model.

By examining the sparse setup (Fig. 7), we see that the false positive probability increases considerably. The false positive rate is effectively reduced using our algorithm since it verifies node availability by examining the routing table.

V. Discussion

The performance results reveal that Mercury is superior to SLPManet and PDP regarding overhead. The major overhead reduction is caused by caching. Service descriptor compression achieved from the Bloom filters (compared to transmitting the service descriptors as text), and piggybacking of the information in OLSR packets further reduce the overhead. Due to these optimizations, it is expected that Mercury outperforms other cross-layer SD proposals [11] and [12].

The time consumed to connect to the actual service is expected to be many times higher than the discovery delay found in the simulations. We therefore state that the service discovery delay is promising for all service discovery alternatives. However, caching is a way to achieve further reduction of the delay.

The proposed path-aware caching architecture, reduces the number of false positives and hence, increases application performance and user-friendliness. An amount (albeit relatively small) of false positive replies may still occur, as network mobility and routing protocol settings may lead to erroneous entries in the routing table.

We state that a combination of optimization techniques as presented by Mercury is inevitable in order to support service discovery in bandwidth-constrained environments.

VI. Conclusions and future work

In this paper, we have presented a method of service discovery using a combination of Bloom filters, the extensibility feature of the OLSR, and a path-aware caching regime. The false positive property of Bloom filters is evaluated and discussed. By simulation, we have demonstrated the performance gain by our cross-layer protocol compared to application layer service discovery alternatives. We also have provided an implementation for real-world usage available for download. Future work includes further optimizations and tests in real deployed networks focusing on bandwidth-constrained environments.

References

[1] M. Abou El Saoud, T. Kunz, and S. Mahmoud. SLPManet: service location protocol for MANET. In *IWCMC '06: Proceeding of the 2006 international conference on Communications and mobile computing*, pages 701–706, New York, NY, USA, 2006.

[2] B. H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.

[3] C. Campo, C. Garc'ia-Rubio, A. M. Lopez, and F. Almenarez. PDP: a lightweight discovery protocol for local-scope interactions in wireless ad hoc networks. *Comput. Networks*, 50(17):3264–3283, December 2006.

[4] S. Cheshire and M. Krochmal. DNS-Based Service Discovery, August. INTERNET-DRAFT draft-cheshire-dnsext-dns-sd-04.txt, Work in progress, 2006.

[5] T. Clausen and P. Jacquet. Optimized Link State Routing Protocol (OLSR). RFC 3626 (Experimental), October 2003.

[6] P. E. Engelstad, Y. Zheng, R. Koodli, and C. E. Perkins. Service discovery architectures for on-demand ad hoc networks. *International Journal of Ad Hoc and Sensor Wireless Networks, Old City Publishing (OCP Science)*, 2(1):27–58, March 2006.

[7] J. Flathagen. Mercury Service Discovery Plugin for OLSRd. (http://olsr-mercury.sourceforge.net), Accessed 2008.

[8] Y. Goland, T. Cai, P. Leach, and Y. Gu. Simple service discovery protocol/1.0. INTERNET-DRAFT draft-cai-ssdp-v1-03.txt, Work in progress, 1999.

[9] E. Guttman, C. Perkins, J. Veizades, and M. Day. Service Location Protocol, Version 2. RFC 2608 (Proposed Standard), June. Updated by RFC 3224, 1999.

[10] S. Helal, N. Desai, V. Verma, and C. Lee. Konark - a service discovery and delivery protocol for ad-hoc networks. *Proceedings of the Third IEEE Conference on Wireless Communication Networks (WCNC), New Orleans*, 2003.

[11] J. L. Jodra, M. Vara, J. M. Cabero, and J. Bagazgoitia. Service discovery mechanism over OLSR for mobile ad-hoc networks. *Advanced Information Networking and Applications, AINA*, 2:534–542, 2006.

[12] L. Li and L. Lamont. A lightweight service discovery mechanism for mobile ad hoc pervasive environment using cross-layer design. *Pervasive Computing and Communications Workshops*, pages 55–59, 2005.

[13] J. Macker. Simplified multicast forwarding for manet. INTERNET-DRAFT draft-ietf-manet-smf-05, Work in progress, 2007.

[14] Martineau, Y. Peers SIP User Agent. (http://peers.sourceforge.net/), Accessed 2008.

[15] Naval Research Laboratory. NRL-OLSR. (http://cs.itd.nrl.navy.mil/), Accessed 2008.

[16] A. Obaid, A. Khir, and H. Mili. A Routing Based Service Discovery Protocol for Ad hoc Networks. In *ICNS '07: Proceedings of the Third International Conference on Networking and Services*, 2007.

[17] olsr.org. The OLSR daemon. (http://www.olsr.org/), Accessed 2008.

[18] C. Perkins, E. Belding-Royer, and S. Das. Ad hoc On-Demand Distance Vector (AODV) Routing. RFC 3561 (Experimental), July 2003.

[19] R. Rivest. The MD5 Message-Digest Algorithm. RFC 1321 (Informational), April 1992.

[20] Sun. Jini. (http://www.jini.org/), Accessed 2008.

[21] A. Tønnesen, A. Hafslund and Ø. Kure. The Unik-OLSR Plugin Library. In *The OLSR Interop and Workshop*, 2004.

[22] University of California. ns2 Network Simulator. (http://www.isi.edu.nsnam/ns/), Accessed 2008.

[23] University of Murcia. UM-OLSR. (http://masimum.dif.um.es/), Accessed 2008.