Scalable Service Discovery in Mobile Ad hoc Networks

Uday Mohan[1], Kevin C. Almeroth[1], and Elizabeth M. Belding-Royer[1]

Department of Computer Science, University of California, Santa Barbara,
CA 93106-5110, Ph: 805-893-7520, Fax: 805-893-8553
{udaym, almeroth, ebelding}@cs.ucsb.edu

Abstract. Recent advances in hardware technology and wireless networking have made it possible to deploy large scale ad hoc networks[1]. As these networks begin to grow in size, an efficient mechanism is needed to locate services distributed within them. Protocols such as the Service Location Protocol (SLP)[2] and Universal Plug and Play (UPnP)[3] exist for service discovery in static networks. These protocols are based on a centralized server where services can register themselves and clients can query for them. However, maintaining such a server in an ad hoc network is difficult as nodes randomly join and leave the network. In this paper we study the problems associated with service discovery by first simulating two well known service discovery techniques and investigating their limitations for large network sizes. We then combine the best features of each approach to present an innovative, scalable mechanism. We simulate and analyze this mechanism and show it to scale well for large network sizes without increasing the latencies in locating a service.

Keywords: Ad hoc network, Service Discovery, Simulations

1 Introduction

A mobile ad hoc network (MANET) is a network formed by a group of wireless devices with limited power and transmission range[4]. These networks do not need any existing infrastructure but can form a network on the fly, with each device acting as a relay to forward packets for other nodes. With recent advances in hardware technology and wireless networking, it is now possible to deploy large scale MANETs[1]. As these networks begin to grow in size, an efficient mechanism is needed to locate services distributed within them. A *service* may be a computation, storage, a communication channel to another user, a software filter or a hardware device[5] that can be used by a person or a software program.

There are currently a number of existing protocols for service discovery [2, 6, 3, 7, 8]. These protocols are centralized, registration-oriented protocols, with an assumption that a centralized database of services can be maintained and accessed by every node. However, these existing strategies do not work well for MANETs because of several reasons. First, in such networks a centralized server is difficult to maintain as the nodes can join or leave the network at random and therefore no node is part of the network permanently. Second, because of the dynamic nature of the network, every time a service leaves or joins the network it has to inform the centralized server about its presence and this presents a scalability issue. Considering these and other factors, a decentralized approach to service discovery is desirable in ad hoc networks.

Several intuitive ways to solve this problem come to mind. A straightforward solution is a Push-based solution in which *services* advertise themselves in the network by periodically broadcasting packets. A second method to locate services is a Pull-based method where *clients* actively broadcast requests into the network. In this paper we evaluate these two well known discovery techniques for MANETs. We then propose a model that uses the best features of each, and add mechanisms to provide scalability.

The rest of the paper is organized as follows. In Section 2 we describe a motivating scenario for service discovery in MANETs. Section 3 describes the push-based and pull-based techniques to discover services in a MANET and identifies their limitations. In Section 4 we present our Adaptive Service Discovery Model. Section 5 presents simulation and analysis of our model and Section 6 offers our conclusions.

2 Motivation

To understand resource discovery and its requirements, we now discuss in detail a motivating example. Consider a shopping mall, where a MANET is formed by people carrying mobile devices. Some of those devices may be in need of traditional services such as printing, scanning or access to the Internet, while others may be looking for new services such as spare CPU cycles. There will be companies advertising their services for which people may want to pay, e.g. McDonalds offering their lunch time specials or banks offering stock quotes. Some people may be running a peer-to-peer system such as Gnutella[9] to exchange files or multimedia content. There may be teenagers wanting to locate friends to chat with, or parents monitoring their children. In essence, there is a dynamic network with a wide selection of resources available, and there are devices that constantly need to locate these resources. As the number of devices entering the network grows, an efficient mechanism will be required to discover these resources. In such a network,there is a high probability that a large percentage of the nodes will take part in service discovery.

A number of existing protocols for resource discovery exist like the Service Location Protocol(SLP)[2, 6], Jini[7], Salutation Consortium[8], and Universal Plug and Play (UPnP)[3]. All these protocols have similar architectures, the common feature being a centralized database that keeps track of all available services in the network. This feature works well in static networks where a centralized database can be maintained. However, it does not work well in MANETs because of several reasons.

First, maintaining a centralized database is difficult in MANETs because no node is a permanent member of the network. Nodes join and leave the network at random. There may be no pre-existing base stations, no well-known servers and no guaranteed Internet connectivity. This makes the centralized approach difficult.

Second, even if such a centralized database is somehow maintained, the nodes are mobile and the same service may enter and leave the network multiple times.

Every time it disconnects and then connects, it has to contact the database and send updated information. Therefore, the centralized database has to keep track of every service in the network and has to do it every time the service joins or leaves the network. As the number of services in the network increases, this method does not scale well.

Third, several nodes may be providing the same service in the network and these nodes may enter at different times. An existing node might need updated information of all services in the network at all times. For example, there may be several nodes in a network providing Internet connectivity, with different costs associated to them. A node looking for an Internet connection will prefer to be connected to the server with the least cost of service. Hence, this node has to periodically poll the centralized database for updated information about any new server with better costs that might have entered the network at a later time. This continuous polling creates a serious scalability problem.

3 Architectural Considerations

We have described several scenarios where resource discovery is critical. The network used in each of these scenarios can be reduced to a generic architecture. This architecture is shown in Figure 1 and consists of mobile devices that fall into three categories.

1. Mobile devices that can provide services or have content that can be exchanged. We call these devices *servers*.
2. Mobile devices looking for such services. We call these devices *clients*.
3. Nodes that are not part of this resource discovery architecture but just members of the MANET helping in routing packets. We call them *forwarders*.

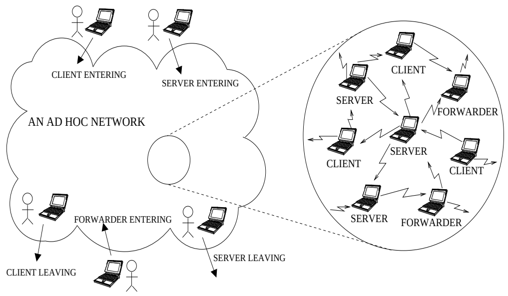

Fig. 1. A generic architecture for a Mobile Ad hoc Network.

There are two intuitive methods to locate resources in the above architecture and are discussed next.

3.1 The Push and Pull Models

A simple way to enable service discovery is for services to advertise their presence in the network (Push model)[5]. Another method to locate services is for clients to actively query for them (Pull model)[5]. To understand these two models and create the motivation for our better, more scalable solution, we next simulate them and study their characteristics and performance impact on the network.

Simulation of the Push and Pull models Simulations were performed using the GloMoSim Network Simulator[10] developed at UCLA. In particular, we wanted to study the traffic generated and the latencies in locating services when the number of servers and clients varies.

To simulate a server or a client entering the network, we start it at a random node. A server provides only one service. Each client requests only one service. There can be several servers providing the same service and several clients interested in the same service. We keep the total number of nodes fixed to 400 but vary the percentage of servers and clients.

To simulate the **Push Model**, each server advertises its service by broadcasting packets containing the *source address*, the *advertised service type*, and a *sequence number*. Each client passively waits for a particular service to be advertised. Upon receiving the advertisement, it can determine the address of the service and can contact the service directly. The latency is the difference between the time a client starts looking for a service and the time it receives the first advertisement packet from a server.

For the **Pull model**, the servers listen passively for service requests. When a client wants a particular service, it periodically broadcasts a request packet containing the *node address*, the *required service*, and a *sequence number*. When a server receives a packet with a matching service type, it broadcasts a reply back to the client with its address. In our model, we assume that several clients may be interested in the same service and accordingly a broadcast is preferable over unicast. The latency here is the difference between the time a client sent out its first query packet and the time it got the first packet with the service address.

An important point to note in the Pull model is that a client may broadcast request packets even after it has located a server. It does this because several nodes may be providing the same service, and may join and leave the network at different times. By continuously broadcasting query packets, the client has up-to-date information about all services in the network. An optimization here could be to reduce the rate of querying after locating the first server. In our model, we have implemented a simplified approach in which we do not reduce the querying rate but keep it the same even after locating a service.

The number of nodes used for the simulations was 400. These nodes were initially placed randomly within a fixed-size of a 1500 x 1500 m^2 field. We choose this field size as it could be a typical shopping mall or a disaster rescue area. The random waypoint mobility model[11] was choosen with speeds between 0 and 5 meters/second and a rest period from 0 to 10 seconds. The range of each communicating node is 250 meters and the link bandwidth is 2 Mbits/second. The protocol used is UDP with each advertisement or query packet having a size of 512 bytes. We ran a number of simulations by varying the values of each parameter. Had we chosen a higher or lower value, the shape of the graphs would be slightly different but would still yield the same conclusion. Hence, the results using the above parameters are representative of the entire set of simulations we conducted.

Evaluation Figure 2 shows the latency and overhead of the Push and Pull models. The left side of the x-axis has 0% servers and 100% clients and it varies linearly to 100% servers and 0% clients. The total number of nodes is kept constant at 400.

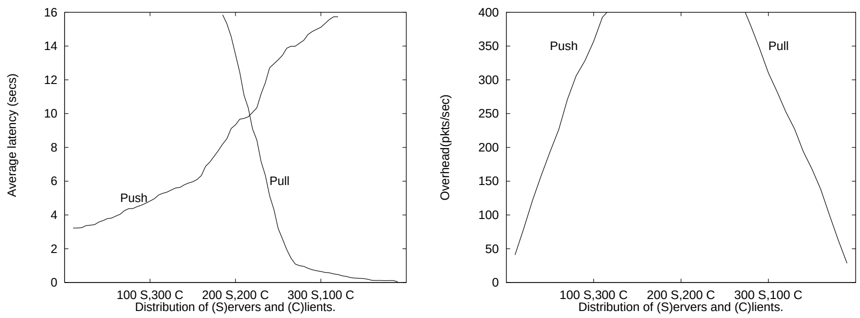

Fig. 2. Average latency and overhead for the Push and Pull models with varying distributions of Servers and Clients.

From Figure 2 it can be observed that the average latency in the Push model increases as the number of servers in the network increases. This can be explained by the overhead graph in Figure 2. As the number of servers increases, there is a corresponding rise in advertisement traffic. When the traffic increases beyond a certain limit, collisions start to occur and packets are lost. This results in the observed rise in latency to locate services. For example, suppose a server advertises a service every 10 seconds. If a packet is lost due to congestion, a new client entering will hear another advertisement only after 20 seconds.

One consideration then is how often should services advertise their presence. Advertising too often just increases the network traffic and redundant advertisements are flooded through the network. Advertising too infrequently results in high latencies. There is a fine balance between the number of services in the network and their frequency of advertisement so as to keep the number of collisions minimum. Achieving this balance is nearly impossible because the topology and membership of the network is dynamic.

Similarly, from Figure 2 it can be observed that the average latency in the Pull model increases as the number of clients in the network increases. This can also be explained from the overhead graph in Figure 2. When the number of clients in the network increases, the control traffic increases proportionately. This rise in control traffic causes congestion, and control packets are lost. This results in latencies becoming higher. For example, suppose a client queries the network every 10 seconds. If a packet gets lost due to congestion, the client will not get a reply. Clients then have to issue a request multiple times before they receive a response.

Like in the server Push case, the challenge is finding the appropriate interval for clients to query the network for a service. After sending out a query packet, how should the client decide when to query again? Sending it after a long interval of time will result in high latencies. Querying too often will result in more traffic being generated. There is a fine balance between the number of clients and their frequency of querying so as to keep the number of collisions minimum.

The simulations so far show that neither of the above methods is scalable to large networks. We need a mechanism that combines the good features of both methods, is scalable, and can limit the control traffic. The next section presents our scalable model for resource discovery.

4 Adaptive Service Discovery Model

Our model for resource discovery uses a combination of the Push and Pull methods plus additional mechanisms for scalability. A key feature in our model is that the percentage of bandwidth to be utilized for resource discovery can be set so as not to exceed a given upper bound.

In our model, a server and a client both actively try to locate each other. The goal of a server is to periodically advertise its services while the goal of the client is to query for a needed service. A client can locate a service in two ways. It can either receive a periodic advertisement by a server, or can receive a response to a query. In either case, this is done in an adaptive manner in order to keep the control traffic below a threshold limit. Every time a server is scheduled to advertise, it first listens for other control packets in the network. It listens for a time interval to estimate the number of members that advertised or queried within that period. If this number is below a threshold, the server sends an advertisement; otherwise it exponentially backs off. This is shown in Figure 3. At the end of the backoff period, it again senses the network and repeats the process. If it has backed off a maximum number of times, it ceases backing off and advertises *forcefully*. This forceful advertisement is special in that it forces others to back off, and lets it advertise. This is used to prevent starvation. This happens because other members also sense the traffic before advertising, and hear the forceful advertisement and back off themselves.

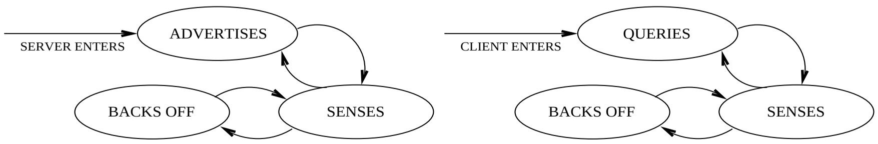

Fig. 3. State diagram for the server and client.

Along with advertising its presence, a server also listens for queries from clients. If it hears a query and can provide the requested service, it sends a reply immediately. This reply could be unicast to the client or could be a general broadcast into the network. In our model, the server broadcasts a reply to the client. This is because we assume that several clients may be interested in locating the same service and so a broadcast is preferable. To limit response scaling problems, a server only replies to a query if it has not heard from the client in one scheduled advertisement interval[1].

[1] This, and other security concerns will be dealt with in Section 5.3.

Similarly when a new client wants to locate a service, it first transmits a query packet to announce its presence. Before it broadcasts the next packet into the network, it listens to the control traffic and estimates the number of members in the network. It accordingly queries for services or backs off similar to the way a server adapts its rate. This is shown in Figure 3.

We now describe our model more formally. The parameters used are described in Table 1. The method to select an appropriate value for each parameter will be discussed in Section 5.2 and 5.3.

Table 1. Parameters used in our simulations.

$\mathcal{T}$	is the period of time that a server or client listens for other control packets in the network. This should be at least equal to the NET_TRAVERSAL_TIME[12], which is a conservative estimate of how long it should take a message to traverse the entire MANET.
$\mathcal{N}_{Threshold}$	is the maximum number of members that can advertise or query in $\mathcal{T}$ time. The amount of bandwidth taken up by the resource discovery protocol depends on this variable.
$\mathcal{S}_{Advertisement}$	is the number of advertisements packets a member hears from unique sources.
$\mathcal{C}_{Query}$	is the number of query packets a member hears from unique sources.
β	is the number of times a member has backed off.
β_{Max} & β_{Min}	is the maximum and minimum values of the backoff counter.

Model

1. When a new server (or client) enters the network, it advertises (or queries) once to announce its presence in the network[2]. It sets β to β_{Min}.
2. It listens for a time $\mathcal{T}$ to the control traffic to estimate the number of members $\mathcal{N}$ that advertised within that time period. $\mathcal{N}$ is calculated as

$$\mathcal{N} = \sum(\mathcal{S}_{Advertisement} + \mathcal{C}_{Query})$$

   - IF $\mathcal{N} <= \mathcal{N}_{Threshold}$ then
     Advertise (or Query).
     Set β to β_{Min}
     ELSE
     IF $\beta < \beta_{Max}$
     Increment β by one.
     Backoff for 2^{β} x $\mathcal{T}$ seconds
     ELSE
     Advertise(or Query) forcefully.
     Set β to β_{Min}
3. Goto step 2.

[2] The problem of traffic explosion, i.e. when a large number of members suddenly enter the network at the same time, may occur but the impact is not significant. The reason for this will be explained in Section 5.2.

5 Simulation and Analysis

We simulate our Adaptive model to achieve several goals. First, we compare the performance of our model with both the Push and Pull models. Through this comparison we evaluate the improvements provided by our model. Second, we investigate the performance of our model in different scenarios to understand its characteristics and determine possible optimizations.

5.1 Simulation Environment

The Adaptive model is simulated in GloMoSim and works as described in Section 4. Servers and clients enter the network at random times advertising or querying for a service. The advertisement or query packets contain fields similar to the fields used in the Push and Pull models. Each server (client) then listens for $\mathcal{T}$ seconds to sense the channel. It advertises (queries) if it senses the number of members below $\mathcal{N}_{Threshold}$; otherwise it backs off exponentially. Every server (client) backs off a maximum of β_{Max} times. After these many unsuccessful attempts, it forcefully broadcasts a packet to tell others to back off and let it advertise (query).

The range of parameters used for the simulation of the Adaptive model is given in Table 2. The nominal values are the default value used in the simulations when the particular parameter is not varied.

Table 2. Range of parameters for the Adaptive model.

Parameter	Min	Max	Nominal
Number of nodes	10	400	400
$\mathcal{N}_{Threshold}$	2	10	5
$\mathcal{T}$(ms)	150	1000	1000
β_{Min}	0	0	0
β_{Max}	3	6	5

5.2 Results

Our simulations begin by comparing the *latency* and *overhead* of the Adaptive model to the Push and Pull models. We then fine tune our model to determine the optimum range of values used for the various parameters.

To compare the latency and overhead, we run two different sets of simulations. In the first set, we keep the total number of nodes fixed at 400 but vary the percentage of servers and clients in the network. In the second set, we run simulations by varying the number of nodes from 10 to 500 and observe how overhead increases for the three models.

Figure 4 shows the same result as in Figure 2 but with additional lines for the Adaptive model. The distribution of clients and servers varies as explained in Section 3.1.

The Adaptive model performs significantly better in terms of latency and bandwidth utilization. Due to better bandwidth utilization, collisions are minimized and this results in lower latencies to locate services. However, this comes at a certain cost. The tradeoff is that the overhead in using our model for a 400

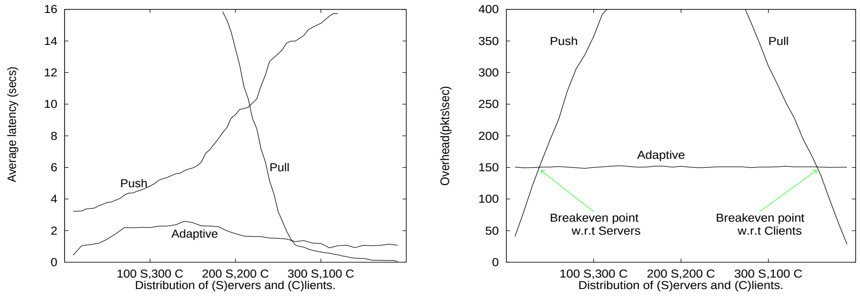

Fig. 4. Average latency and overhead for the Push, Pull and the Adaptive models with varying distributions of Servers and Clients.

node network is greater than the push model if the number of servers is less than 40 servers. Similarly the overhead in our model is greater than the pull model if the number of clients is less than 35. This happens because the traffic is low when there are few clients in the Pull model and few servers in the Push model and there is a lot of free bandwidth available. In the Adaptive model, servers and clients both advertise and query, hence more traffic is generated.

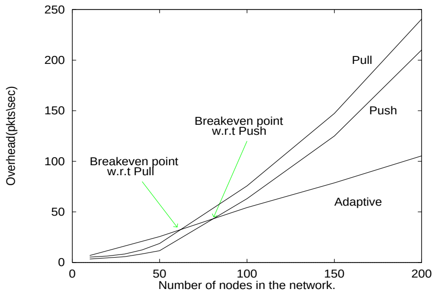

Fig. 5. Overhead of the Pull, Push and the Adaptive models with varying number of nodes.

Figure 5 shows the overhead as the number of nodes in the network is increased. The x-axis shows the number of nodes where each run consists of 50% servers and 50% clients. We varied the number of nodes to 500; however Figure 5 only presents our results till 200 nodes. This is because nothing unusual is observed after 200 nodes and the trend continues. We wanted to emphasize the breakeven points and hence, we only present simulation results till 200 nodes. Initially the overhead for the Adaptive model is slightly greater than for the Push and Pull model. However, when there are more than 60 nodes, the overhead for the Pull model becomes greater than the overhead for the Adaptive model. This is because when the number of nodes in the network exceeds 60, the overhead for 30 servers advertising *every 10 seconds* becomes higher than 60 members advertising or querying *adaptively*. Similarly, for the Push model this limit is 81 nodes.

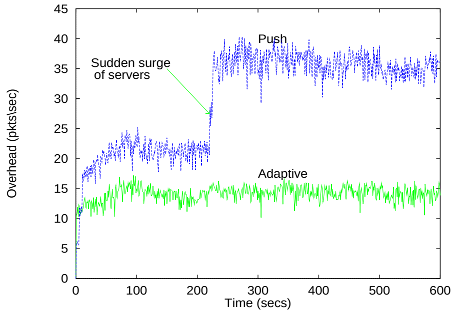

Fig. 6. Overhead with a sudden burst of servers at approximately 220 seconds.

Figure 6 shows the variation of overhead with time when a number of servers suddenly enter the network. In this simulation we start 50 servers at uniform times in the network. At around 220 seconds we simulate a sudden burst of 20 new servers entering the network. In the push model these servers immediately start advertising and hence there is a sharp rise in overhead. The adaptive model performs better under the same scenario as it is able to keep this overhead under control. This is because when new servers enter and advertise the first time, existing members in the network backoff to let these new servers advertise. This happens because in our model, every member listens for an interval $\mathcal{T}$ for advertisements or queries in the network. Existing members hear the new members and back off. If members suddenly enter the network within a time period of $\mathcal{T}$ seconds, there will be a noticeable rise in control traffic in the adaptive model. Hence, the value of $\mathcal{T}$ should be set to an interval such that the probability of several members suddenly entering the network within that interval is very low. Our model performs in a similar manner when there is a sudden burst of clients entering the network because clients follow the same mechanism to limit the overhead. Hence, our model is able to limit the overhead when there is a sudden burst of *servers or clients* entering the network.

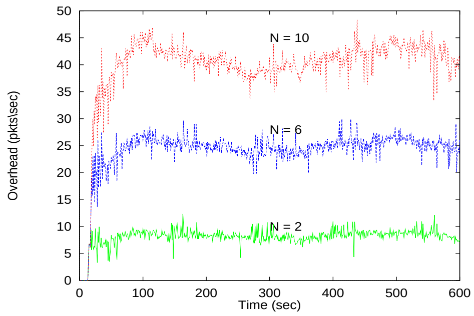

Fig. 7. Variation of overhead with time for different values of N=$\mathcal{N}_{Threshold}$.

Figure 7 shows the variation in overhead as $\mathcal{N}_{Threshold}$ is varied. $\mathcal{N}_{Threshold}$ is the number of members allowed to advertise or query in $\mathcal{T}$ seconds. Setting $\mathcal{N}_{Threshold}$ to a high value results in more members being able to advertise or query and hence more traffic is generated. As the value of $\mathcal{N}_{Threshold}$ is reduced, the control traffic decreases. The amount of bandwidth utilized by our resource discovery model depends on this variable and is an essential feature in our model.

5.3 Discussion

The Adaptive Service Discovery Model performs significantly better than the Push and Pull models. However, there are several practical considerations that need to be dealt with before our model can be deployed. First, how should parameters such as $\mathcal{N}_{Threshold}$ and $\mathcal{T}$ be communicated to new members entering the network? A simple mechanism to do this is to piggyback these two parameters in the dynamic address configuration packet which assigns an IP address to the node[13] when it first enters the network. Another method to set $\mathcal{N}_{Threshold}$ is to set it to be a fixed percentage of the maximum bandwidth available in the MANET[14]. This maximum bandwidth value will be unambiguous and known to every member in the network. To set the value of $\mathcal{T}$, it should at least be equal to the NET_TRAVERSAL_TIME [12] and can be experimentally determined by a node in the network.

Second, there are several security concerns in our model. A server immediately replies with an advertisement when it receives a request packet. A malicious node can flood the network with query packets and force a server to keep replying, thereby causing a denial-of-service attack[15]. To prevent this, every server replies only to the first query it receives from a node within its scheduled advertisement interval. It ignores duplicate queries that it receives between two of its periodic advertisements. Another security concern is if a node cheats and advertises or queries at a higher rate. A node can do this by increasing its value of $\mathcal{N}_{Threshold}$. There has to be a mechanism for neighboring nodes to sense this discrepancy and suppress the additional traffic generated by the cheating node. Techniques such as *Watchdog* and *Pathrater*[16] are likely to be useful if customized for this scenario.

6 Conclusions

As MANETs begin to grow in size, an efficient mechanism is needed to locate services in them. There are currently a number of existing protocols for service discovery. However, these are centralized protocols and do not scale well for MANETs. In this paper we study the problems associated with service discovery for MANETs by simulating the Push and Pull models. We then combine the best features of each approach to present an innovative model. Simulation results show that this model scale wells for large network sizes without increasing the latencies in locating a service. A further improvement can be achieved by setting priorities for services. Generally, 10% of services are used by 90% of clients. This 90-10 rule can be used to further reduce overhead and latency by setting up higher priorities for those 10% services used more often. We also need a formal model to define services in a network in order to let clients accurately define the services they require. All of these are interesting issues that need further research.

References

1. Intel Team: Largest Tiny Network. http://today.cs.berkeley.edu/800demo/ (2001)
2. J. Veizades, E. Guttman, C. Perkins, S. Kaplan: SLP: Service Location Protocol. Internet Engineering Task Force : RFC 2165 (1997)
3. Consortium Members of the UPnP Forum: Universal Plug and Play Device Architecture. Version 0.91. http://www.upnp.org (2000)
4. S. R. Das, C. E. Perkins, E. M. Belding-Royer: Performance comparison of two on-demand routing protocols for ad hoc networks. In: Proceedings of the IEEE Conference on Computer Communications (INFOCOM), Tel Aviv, Israel. (2000)
5. L. Cheng: Service advertisement and discovery in mobile ad hoc networks. In: Workshop on Ad hoc Communications and Collaboration in Ubiquitous Computing Environments. (2002)
6. C. E. Perkins, H. Harjono: Resource discovery protocol for mobile computing. In: IFIP World Conference on Mobile Communications. (1996)
7. K. Arnold, A. Wollrath, B. O'Sulliva, R. Scheifler, J. Waldo: The Jini specification. Addison-Wesley,Reading, MA, USA (1999)
8. Members of the Salutation Consortium: Salutation Consortium Homepage. http://www.salutation.org (2000)
9. M. Ripeanu: Peer-to-peer architecture case study: Gnutella network. In: Proceedings of International Conference on Peer-to-peer Computing. (2001)
10. X. Zeng, R. Bagrodia, M. Gerla: GloMoSim: A library for parallel simulation of large-scale wireless networks. In: Workshop on Parallel and Distributed Simulation. (1998)
11. T. Camp, J. Boleng, V. Davies: A survey of mobility models for ad hoc network research. In: Wireless Communications and Mobile Computing (WCMC): Special issue on Mobile Ad Hoc Networking: Research, Trends and Applications. (2002)
12. C. E. Perkins, E. M. Belding-Royer, S. R. Das: IP broadcast in ad hoc mobile networks. IETF Internet Draft, draft-ietf-manet-bcast-02.txt,(Work in Progress) (2001)
13. N. H. Vaidya: Weak duplicate address detection in mobile ad hoc networks. In: The Third ACM International Symposium on Mobile Ad Hoc Networking and Computing(MOBIHOC). (2002)
14. H. Schulzrinne, S. Casner, R. Frederick, V. Jacobson: RTP: A transport Protocol for Real-Time Applications. Internet Engineering Task Force : RFC 1889 (1996)
15. F. Lau, S. H. Rubin, M. H. Smith, L. Trajovic: Distributed denial of service attacks. In: IEEE International Conference on Systems, Man, and Cybernetics. (2000)
16. S. Marti, T. J. Giuli, K. Lai, M. Baker: Mitigating routing misbehavior in mobile ad hoc networks. In: 6th Annual International Conference on Mobile Computing and Networking(MOBICOM). (2000)