An Efficient Service Propagation Scheme for Large-Scale MANETs

Choonhwa Lee, Sungshick Yoon
College of Information and Communications
Hanyang University
Seoul, Korea
lee@hanyang.ac.kr

Eunsam Kim, Abdelsalam (Sumi) Helal
Dept. of Computer and Info. Science and Engineering
University of Florida
Gainesville, FL, USA
{eunkim,helal}@cise.ufl.edu

ABSTRACT
This paper proposes a new scheme for efficient dissemination and discovery of service information in mobile ad hoc networks. By extending Bloom filter-based service summarization scheme, our discovery middleware layer also encodes distance to services in summarization filters. By having the distance information gradually attenuated during its propagation, our proposal enables faster discovery of services with a limited amount of advertisement traffic.

Categories and Subject Descriptors
C.2.1 [**Computer-Communication Networks**]: Network Architecture and Design – *wireless communication, network communications*

General Terms
Algorithms, Design.

Keywords
Ad hoc service discovery, Bloom filter, service information summarization, distance-aware discovery

1. INTRODUCTION
The potential of mobile ad hoc networks as a means to enable spontaneous networking with little or no manual configuration efforts has actively been explored over the past years. Among others are the prominent advances in ad hoc routing protocols, including flat (reactive and proactive), and hierarchical, and geographic location-aware routing protocols [2]. Along with the advancements in routing protocol research, service discovery problem also received much attention from the research community. It has been widely recognized as an integral part of ad hoc collaboration networking scenarios such as battle fields, rescue operations, and conference sites. Even if basic IP connectivity may be supported by the ad hoc routing protocols, ability to discover desired services or resources on a strange network is a must to realize the vision of ad hoc collaboration on the spot.

MPAC'06, November 27-December 1, 2006 Melbourne, Australia.

Dynamic service discovery can be defined as a process which let services and devices spontaneously become aware of the presence of the peers on the network without explicit administration [17]. In fact, the need of dynamic service discovery was first seen for LAN environments by mobile computing research community in the late 90s. Since then, a variety of service discovery protocols have been proposed, including Jini, SLP, UPnP, Salutation, UCB SSDS, and MIT INS. The research was later extended to cover both extremes of target environments, i.e., WAN and MANET. A rich set of service discovery protocols is found in the literature [1] [14] [17].

In this paper, we propose a new service discovery service for efficient dissemination and discovery of service information in mobile ad hoc networks. By extending Bloom filter-based service summarization scheme [18], our scheme enables faster service discovery for over medium size networks. Before presenting our scheme, we first review the state of the art for ad hoc service discovery problem in Section 2. Our idea of extending the Bloom filter scheme for the purpose of efficient service information dissemination and discovery is presented in Section 3. Section 4 briefs our on-going simulation study, and concludes the paper.

2. AD HOC SERVICE DISCOVERY PROTOCOLS
The peculiarity of ad hoc networks regarding network topology, node, and environments poses new challenges for the service discovery problem. The network is characterized as being far more dynamic and heterogeneous. It comprises a wide range of wireless and mobile devices, though resource constraints on them are assumed in general. Also, communication failures due to environmental effects and limited battery sources are norm in the network. Together with topology instability caused by node mobility, it contributes to the dynamism of mobile ad hoc networks.

The dynamism and heterogeneity of ad hoc network mandates no fixed infrastructure support for service discovery. That is, we can not assume the availability of a stable node in a MANET with moderate resource enough to act as a central directory node as in traditional LAN and WAN environments. Therefore, the discovery protocol should operate in a decentralized manner. It should be energy-aware, requiring minimal network transmission.

Additionally, the protocol itself should be lightweight and adaptive to highly changing environments. While meeting these requirements, it should be designed to support various application needs.

A variety of existing MANET service discovery protocols can be grouped by different classification criteria. First, we can classify them into network layer protocols vs. application layer protocols by the layer where service discovery functionality is supported. In the former group, service discovery protocols are often coupled with underlying routing protocols. That is, ad hoc routing protocols are extended to convey service query and response messages piggybacked on the underlying routing protocol packets. Examples include AODV-SD [4], ODMRP-SD [5], M-ZRP [6], and LSD [7]. The second group protocols, including DEAPspace [9], Konark [11], GSD [10], SSD [12], and SANDMAN [13], support service discovery at application layer, to which services naturally belong, on top of the IP routing layer.

Another classification is also possible by service caching strategies: no caching, non-cooperative distributed caching, and cooperative distributed caching. Unlike LAN environments where service information is usually cached in a centralized directory node, we can not assume such a luxury in ad hoc networks. Therefore, we may have to resort to the on-demand multicasts of discovery request messages that would trigger subsequent reply messages (no caching). In distributed caching cases, service information is distributed over the network. They can further be divided into non-cooperative and cooperative as to whether or not caching nodes cooperate with one another to resolve a service query. Despite subtle and minor differences when it comes to the details of caching strategies, by and large, individual nodes locally cache information about services incidentally overheard. A node consults its service cache, when it receives a service query. In the case of non-cooperative caching [4] [5] [7] [11] [8] [9] [6], the query is simply broadcast to its neighbors, if not found in its local cache. In the case of cooperative caching strategy [10] [12], each node propagates information about services seen nearby to its neighbors, so that queries unresolved by a local cache can be intelligently and selectively forwarded to a neighbor node which seems to have a clue on where to find the service in quest.

3. BLOOM FILTER-BASED SERVICE DISCOVERY

Bloom filter is a method to strike balance between storage and computation. In other words, it is a summarization technique to compress information into a less amount at the cost of some possible loss. It is actually a vector v of m bits initially set to 0. k independent hash function h_1, h_2, ..., h_k are used to indicate bit positions to be turned on. More specifically, given an input w, the designated bit positions are set to 1, and others remain zero.

$$v[h_i(w)] = 1, \text{ where } i = 1, 2, .., k. \qquad (1)$$

3.1 Bloom Filter-Based Discovery Protocols

In addition to others [15], Bloom filter has also found its place as a means for efficient information propagation for dynamic service discovery problem [18] [12]. Figure 1 illustrates how Bloom filter is used for the discovery purposes. For a service description word w, bit positions pointed to by hash functions are set to 1. Given a service attribute, *printer*, four hash functions indicate bit position 2, 4, 7, and 13 to be turned on. This way service information is compressed in the form of bit stream. Similarly, a service word, *fax*, sets bit 6, 10, 11, and 13. When given a query of *printer* later on, the Bloom filter can determine whether the word is encoded in or not by simply looking at the corresponding bits' value.

Note that, since a single bit array is used for all inputs, a bit position can be set multiple times by more than one input. This can causes *false hit* in which case the filter says "yes" misleadingly, when a query word is actually not included in the filter.

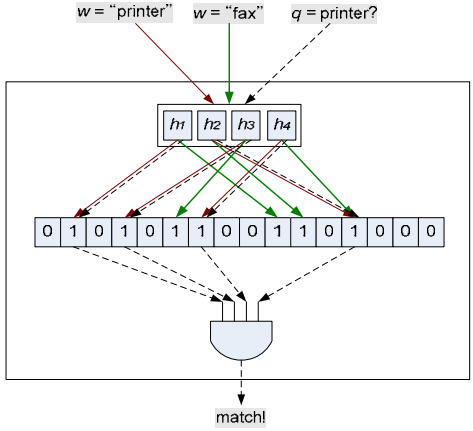

Figure 1. Bloom filter-based service matching.

When aggregating service information using Bloom filter, we have to consider every possible combination of service attributes as input w to the hash functions. In other words, all descriptive attributes and their possible combinations must be encoded, in advance, into the filter for later retrievals using the same words as a query. This is the way multi-key search is supported by Bloom filter-based service discovery protocols.

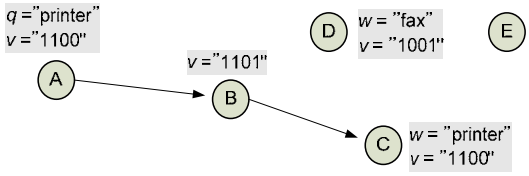

Figure 2. Service discovery using Bloom filter.

Figure 2 illustrates how Bloom filter can be used to support service query routing. Each node, in the figure, represents a directory node which summarizes information about services available in its vicinity, including services offered by itself as well as nearby non-directory nodes. The nodes can be administratively organized in a hierarchical fashion in the case of fixed networks [18], or they can form a virtual backbone network elected and deployed on the fly for MANETs [12]. Each directory node exchanges its own Bloom filter with its neighbors for selective query forwarding later on. Based on neighbor filters, a node can figure out where to forward a service query that is not

resolved by itself. Suppose that node *C* and *D* compute their Bloom filter as *1100* and *1001*, respectively, for a *fax* service and a *printer* service around them as depicted in Figure 2. From its neighbor filters from node *C* and *D*, node *B* produces an aggregate filter (*1101*) as its own by performing a bitwise OR operation of the two filters. Then, it is propagated to node *A*. By knowing that node *B*'s filter is *1101*, node *A* can forward to node *B* a query looking for a printer service. This is because the hashed printer query, *1100*, is matched against node B's filter *1101* that node *A* knows of. This Bloom filter-based query forwarding is repeated at node *B*, which results in the query routed once again and finally matched with a service instance at node *C*.

3.2 Fisheye Bloom Filter

There has been a proposal to enhance the Bloom filter protocol. By associating a small counter with each bit position of the filter, it can immediately be determined whether or not a bit should be cleared, when it is pointed at by a service being removed [12]. This allows us to avoid re-computing the entire Bloom filter even for a single service removal, but at the cost of increased space due to an additional dimension to hold the counter values. It must be guaranteed that the occurrence counter does not exceed *log n*, where *n* is the number of times a bit is repeatedly set. Otherwise, the whole idea of using the counter will be defeated.

```
fillin(v0, 0)  // v0 initialized to zero.

for every hash function, hi, and every service attribute
      w of services in a local area
{
        v0[hi(w)] = 1
}

for each element position p from 1 to m
{
       if (v0[p] == 0)
             v0[p] = MIN(MIN(v1[p], v2[p], ..., vn[p]) + 1, 2^d)
}
```

Figure 3. Fisheye Bloom filter algorithm.

Our idea is to encode into the filter distance to service instances instead of the occurrence counter to keep track of how many times a bit is set. We have extended the one dimensional array by adding the second dimension to indicate how far services might be away. In other words, the filter has now changed into a vector *v* of *m* elements of *d* bits to hold a directory node hop counter to services in question. The counter is incremented by one at each hop, as our extended Bloom filter ripples over directory nodes. In the same way, *k* independent hash function $h_1, h_2, ..., h_k$ are used to determine which element value needs to be adjusted. More specifically, a non-zero value can either be incremented or overwritten by a pointer to closer service instances (i.e., a lower value.) Figure 3 summarizes our algorithm whereby a directory node calculates a new Bloom filter (v_0), based on the filters (v_1, v_2, ..., v_n) from its neighbor nodes.

Figure 4 shows Fisheye Bloom filters above each directory node in a network where two *printer* services are advertised around directory node *C* and *F*, and a *fax* service is located near node *E*. Service information aggregated by the algorithm in Figure 3 propagates throughout the network, which is indicated by arrows. Given a Fisheye Bloom filter *v*, we know that the distance to services in question is at least MAX($v[h_1(w)]$, $v[h_2(w)]$, ..., $v[h_k(w)]$). Suppose that node *A* later receives a query looking for a printer service. By passing through the same hash functions, it translates the query into *1100*. By consulting node *B*'s Bloom filter (*2203*), it knows where to forward the query (i.e., to node *B*) and an instance may be found in two hops (from the first and second 2 in the filter.) On receiving the passed-on query, node *B* now figures out that one may be found in the direction of node *D*, but it should be at least 3 hops away from the first and second digits of node D's filter *2302*. Also, node *B* knows that a closer one is located in the other branch from node *C*'s filter *1100*. This is the way our Fisheye Bloom filter scheme supports distance-aware service discovery.

In sum, our approach is to embed distance information into Bloom filters by which closer instances can be favorably discovered. As shown in Figure 4, further details on which branch to follow are revealed, as we get closer to services being sought. It is also important to know that total service traffic is contained to some extent (i.e., bound by the depth *d* bits.) This is similar, in spirit, to FSR (Fisheye State Routing) protocol [16] which maintains more accurate route information about closer nodes but less accurate for farther nodes. The protocol enables efficient yet correct routing, while being able to reduce route traffic. Our approach is based largely on the same idea: getting a clearer view, as we near a target node. This is why we named our approach as Fisheye Bloom filters.

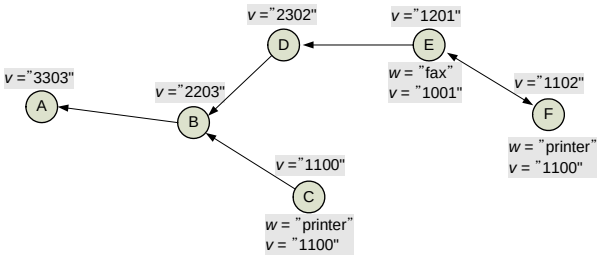

Figure 4. Service discovery based on Fisheye Bloom filter.

A few intuitive insights into our extension to the base Bloom filter can be summarized as follows.

- The cascaded propagation of Bloom filter changes in both cases of service additions and removals is limited at most to 2d hops, which should set an upper bound of service update traffic. However, it does not affect the correctness of the protocol operation, i.e. its ability to discover services present somewhere in the network.
- Service information fades out gradually, as Fisheye Bloom filter travels away from its advertising source node. This is a good compromise between service traffic and query routing performance especially for a large-scale ad hoc network. In other words, our scheme scales to large size networks without incurring excessive service advertisement traffic.

4. CONCLUDING REMARKS

Our simulation study is underway to evaluate the performance of Fisheye Bloom filter scheme versus its base case. We have developed an agent module that implements Fisheye Bloom filters on top of ns-2 simulator (version 2.26) with MAODV (Multicast Ad hoc On-Demand Distance Vector) extension [19]. Multi-hop multicasts are necessary for service advertisements and neighbor directory node discovery. As the simulation workload, we use an English dictionary. An English word is used for service name, and its definition for descriptive service attributes. We are planning on measuring such performance metrics as query routing path length and latency, discovery traffic, and service advertisement traffic against different network configurations and service densities..

This paper presents an efficient service discovery scheme named as Fisheye Bloom filter. The key idea is to embed distance information into Bloom filters by which closer instances can be favorably discovered. A primary design goal of the scheme is scalability without excessive service discovery traffic for large-sized networks. Our performance evaluation via ns-2 simulations is under way to prove the intuition behind the protocol design. Besides, a thorough analysis of the trade-off of faster discovery and space increase remains our future work..

5. ACKNOWLEDGMENTS

This work was supported by grant No. R01-2005-000-10267-0 from the Basic Research Program of the Korea Science & Engineering Foundation.

6. REFERENCES

[1] F. Zhu, M. Mutka, and L. Ni, “Service Discovery in Pervasive Computing Environments,” IEEE Pervasive Computing, vol.4, no.4, pp.81-90, October-December 2005.

[2] X. Hong, K. Xu, and M. Gerla, “Scalable Routing Protocols for Mobile Ad Hoc Networks,” IEEE Network, vol.16, no.4, pp.11-21, July-August 2002.

[3] E. Guttman, C. Perkins, J. Veizades, and M. Day, “Service Location Protocol Version 2,” IETF RFC 2608, 1999.

[4] R. Koodli and C. E. Perkins, “Service Discovery in On-Demand Ad Hoc Networks,” IETF Internet Draft - work in progress, 2002.

[5] L. Cheng, “Service Advertisement and Discovery in Mobile Ad Hoc Networks,” Proc. of the ACM 2002 Conf. on Computer Supported Cooperative Work Workshops, November 2002.

[6] C. Ververidis and G. Polyzos, “Routing Layer Support for Service Discovery in Mobile Ad Hoc Networks,” Proc. of the 3rd Int’l Conf. on Pervasive Computing and Communications Workshops, pp.258-262, March 2005.

[7] L. Li and L. Lamont, “A Lightweight Service Discovery Mechanism for Mobile Ad Hoc Pervasive Environment Using Cross-Layer Design,” Proc. of the 3rd Int’l Conf. on Pervasive Computing and Communications Workshops, pp.55-59, March 2005.

[8] U. Kozat and L. Tassiulas, “Network Layer Support for Service Discovery in Mobile Ad Hoc Networks,” Proc. of IEEE INFOCOM, pp.1965-1975, March 2003.

[9] M. Nidd, “Service Discovery in DEAPspace,” IEEE Personal Communications, vol.8, no.4, pp.39-45, August 2001.

[10] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, “GSD: A Novel Group-Based Service Discovery Protocol for MANETs,” Proc. of the 4th IEEE Conf. on Mobile and Wireless Communication Networks, pp.140-144, September 2002.

[11] S. Helal, N. Desai, V. Verma, and C. Lee, “Konark - Service Discovery and Delivery Protocol for Ad Hoc Networks,” Proc. of IEEE Wireless Communications and Networking Conf., pp.2107-2113, March 2003.

[12] F. Saihan and V. Issarny, “Scalable Service Discovery for MANET,” Proc. of the 3rd Int’l Conf. on Pervasive Computing and Communications, pp.235-244, March 2005.

[13] G. Schiele, C. Becker, and K. Rothermel, “Energy-Efficient Cluster-Based Service Discovery for Ubiquitous Computing,” Proc. of the 11th ACM SIGOPS European Workshop, September 2004.

[14] S. Helal, “Standards for Service Discovery and Delivery,” IEEE Pervasive Computing, vol.1, no.3, pp.95-100, July-September 2002.

[15] L. Fan, P. Cao, J. Almeida, and A. Broder, “Summary Cache: A Scalable Wide-Area Web Cache Sharing Protocol,” IEEE/ACM Transactions on Networking, vol.8, no.3, pp.281-293, June 2000.

[16] G. Pei, M. Gerla, and T.-W. Chen, “Fisheye State Routing: A Routing Scheme for Ad Hoc Wireless Networks,” Proc. of IEEE Int’l Conf. on Communications, pp.70-74, June 2000.

[17] W. K. Edwards, “Discovery Systems in Ubiquitous Computing,” IEEE Pervasive Computing, vol.5, no.2, pp.70-77, April-June 2006.

[18] S. Czerwinski, B. Zhao, T. Hodes, A. Joseph, and R. Katz, “An Architecture for a Secure Service Discovery Service,” Proc. of the 5th Int’l Conf. on Mobile Computing and Networks, pp.24-35, September 1999.

[19] Y. Zhu and T. Kunz, “MAODV Implementation for ns-2.26,” Technical Reprot SCE-04-01, Systems and Computing Engineering, Carleton University, Canada, January 2004.