Service Discovery for Delay Tolerant Networks

Zijian Wang, Eyuphan Bulut, and Boleslaw K. Szymanski

***Abstract*—Service discovery is an essential step in deploying many wireless network applications. The design of service discovery protocols is particularly challenging for mobile wireless networks because of their dynamic and unstructured nature. Most of the previously proposed protocols are based on the assumption that there exists an end-to-end connection from the query source node to the destination node, the assumption that rarely holds for mobile wireless networks. In this paper, we propose a novel service discovery protocol for delay tolerant networks in which all node connections are intermittent. To describe services through a fixed size string, we use Bloom filter that leads to efficient service announcement and search. The service queries are dispatched into the network and forwarded by random walk search using node meeting history as hints. Our preliminary simulation results show that the proposed protocol achieves good performance as measured by the service discovery success rate, delay and overhead.**

***Index Terms*—service discovery, Bloom filter, random walk search, delay tolerant networks**

I. Introduction

THE growth of wireless devices ranging from PDA to mp3 player makes many new applications available on top of the mobile wireless networks formed by these devices. Due to limited capabilities of a single device, these applications often need to utilize services provided by other devices. Hence, the performance of a service discovery scheme can greatly affect the performance of the entire application. However, lack of infrastructure and unpredictability of mobility patterns make service discovery very difficult in such mobile wireless networks.

Most of the previous service discovery approaches for mobile wireless networks can be classified into two categories: directory-based and directory-less. For the former, service providers register and update their services in specific service directories and service requestors send service queries to these directories to get the services available in the network. In [1], a subset of nodes is selected to constitute a relatively stable virtual backbone to store service registrations. In [2], the whole network is divided into hexagon grids, mobile nodes in each grid are grouped into clusters while a gateway in each cluster is used as directory for service discovery. Klein et al. [3] used a service ring to group devices that are both physically close to each other and offer similar services. Each ring possesses a service directory which knows a summary of all services offered within. In these papers, if service queries do not find matches in local directories, they are forwarded to other directories. Instead of forwarding them blindly to all directories, Seada et al. [4] use distributed hash tables along with node position information to determine a mapping between services and directories. Since topology changes frequently, service directories need to be assigned to nodes and maintained there dynamically, which means that directory-based approaches cause additional communication overhead.

For the second category of approaches, there is no service directory in the network and the researches mainly focus on two problems: service advertisements and service query. In [5], each node providing services periodically broadcasts service advertisements to its one-hop neighbors. Those advertisements contain services provided by the node itself and by the node's neighbors. To lower the overhead introduced by such broadcasting, service advertisements are multicast to a fixed multicast group in [6]. Chakraborty et al. [7] used an advertisement range measured in number of hops specifying when the advertisement will be dropped to restrict the service advertisement area. In [8], the service query is integrated with the routing protocol. Such integration can improve performance, but there is still a need for network wide flooding searches. Mian et al. [9] used random walk based search to find the service in the network. The service query packets are forwarded to the neighbor nodes selected based on hints which reduce the number of needed hops and limit the area traveled by the query packet.

All the literature mentioned above targets the wireless networks in which the node density is high enough to guarantee the existence of an end-to-end connection from the source to the destination. Thus, they are not suitable for delay tolerant networks, where the end-to-end connections are hard to maintain due to low node density and unpredictable node mobility. In [10], Maheo et al. proposed a two layered middleware for service discovery in delay tolerant networks. Using DoDWAN [10] protocol, the service requestor first sends a discovery query and waits for a discovery reply. DoDWAN relies on a gossip-like communication in which nodes exchange

Zijian Wang, Eyuphan Bulut and Boleslaw Szymanski. Authors are with the Department of Computer Science, Rensselaer Polytechnic Institute, Troy, NY 12180 USA. (e-mail: {wangz, bulute, szymansk}@cs.rpi.edu).

Research was sponsored by US Army Research Laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defence, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

packets according to their respective interest profiles. Each node periodically broadcasts its interest profile and a list of service descriptors of the packets it currently carries and when any of the receivers of this broadcast is either interested in the broadcasted interest profile or can provide the service matching the service description, the data are exchanged. After receiving a discovery reply, the service requestor sends an invocation request to the service provider and waits for the real service data. To the best of our knowledge, this is the only paper that targets service discovery in delay tolerant networks. Unfortunately, the paper does not discuss how the interest profiles are created. With all nodes having interest profiles covering all services, both query and service packets are using expensive epidemic spreading, while with all nodes having interest profiles equal to services they provide, the protocol enables service discovery when the requestor meets the service provider. The latter case is termed Proximity-based method, which has low overhead but also low service discovery rate. Even in this case, the overhead (periodic broadcasts of service request) is dependent on frequency of the broadcast and on the length of the cutoff time for the service discover, none of which was defined in [10].

In this paper, we proposed a new service discovery protocol designed for delay tolerant networks. We used a technique based on Bloom filters [11][12] to describe services. We restrict the service advertisements to one-hop distance. The spray and wait routing algorithm [13] and random walk based search method [9] are combined to forward service query and service data packets. Like in [14] [15], history of last meetings between nodes is used as hints to aid the packet forwarding, but we used this information in a different way. Some of the techniques also apply to other environments (such as ad hoc networks), but as far as we know, they are combined to be used in DTN for the first time.

The remainder of the paper is organized as follows. We describe the network model and our assumptions in Section II. In Section III, we introduce our service discovery protocol. Section IV presents the simulation results. Finally, we provide conclusions and outline the future work in section V.

II. Network Model and Assumptions

The network consists of N nodes initially deployed randomly with uniform distribution over a finite, two-dimensional planar region. Each node repeatedly moves to a randomly selected point in the network with a random speed between 0 and a maximum speed V_{max}. Each node has the same maximum transmission range R and a unique ID. The buffer space in each node is assumed large enough so that no messages are ever dropped because of the buffer overflow. Each node has a local clock and clocks of all the nodes are not required to be synchronized. We assume that each node can provide at least one service and each of the services can be represented by some key words.

III. Service Discovery Protocol for Delay Tolerant Networks

Our service discovery protocol is a directory-less method, which uses periodical broadcast to announce service advertisements. To reduce service advertisement packet size and memory usage, Bloom filter is used to represent services with a fixed size string. The service query packets are sprayed into the network and relayed to neighbor nodes that meet the destination node providing the desired service recently.

A. Service Advertisements

Service description and service match

As mentioned in section II, each service can be represented by some key words. However, using key words to describe service makes message exchange and storage utilization inefficient, especially when a node provides multiple services. It is also not convenient for service matching during the service query procedure. We use Bloom filter to compress service key words into a fixed size string (called the service code) to represent services.

The Bloom filter consists of a vector v of m bits (used as service code) and a set of k independent hash functions, h_1, h_2, …, h_k, each with an output range $\{1, \ldots, m\}$, which are known to all the nodes in the network. Assume a node can provide a set of n services $S=\{s_1, s_2, \ldots, s_n\}$, where s_i is the key words representing the corresponding service. The construction of the service code that represents the set of services S using Bloom filter is done as follows.

For each service $s_i \in S$, the bits at positions $h_1(s_i)$, $h_2(s_i)$, ..., $h_k(s_i)$ in service code are set to one. Thus a bit at position p in the service code is one if there is at least one service s_x in S such that $h_i(s_x) = p\,(1 \le i \le k)$. All other bit positions are set to zero. We show an example of service code generation in Fig. 1, where one node provides two services: computing and printing. k is set to 4 and m is set to 16.

To start a search for a service represented by key word s_t, the node generates a target service code by setting bits at positions $h_1(s_t)$, $h_2(s_t)$, ..., $h_k(s_t)$ to one. The target service code is used to match the service codes at searched nodes. Only if all bits set to one in the target service code have the corresponding bits in the service code of the searched node also set to one, the required service may, but does not have to, be provided by the searched node. This is not certain because some of the bits may be set by other services causing a false positive match. However, by appropriately choosing the number of hash functions k and the size of service code m, the probability of false positive matches can be made very small (the technique of doing so is well known, see [11] or [12] for example, so for the sake of brevity we omit its details here).

The advantage of this solution is that, each node can describe a number of services using service code of constant size. Consequently, the service description overhead increases linearly with the number of nodes in the network, but only logarithmically with the number of services per node (probability of false positives decreases exponentially with the

ratio of the size of service code m to the number of services per node [12]). Such an approach keeps the required storage and the size of service announcement packet small.

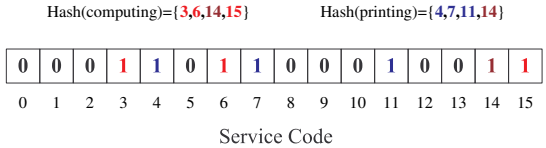


Fig.1 Example of service code generation

Service announcement

Each node broadcasts periodically information about its services through hello messages, which carry node ID and the service codes of all available services. Each node maintains also the list of the nodes that it has encountered. Each entry in the list contains the encountered node ID, service code and also the time of the meeting (denoted as t_m) as measured by the node's local clock. The meeting time is updated to the latest time if the node receives multiple hello messages from the same neighbor node. The meeting time can also be used to indicate whether the corresponding node is the currently active neighbor node. This can be done by comparing the difference between the current time and t_m with the predefined time gap threshold. The encountered nodes are maintained in the list even if they are no longer active neighbor nodes.

B. *Service Query and Reply*

Service query

When an application needs certain service which is not provided by the current node, it sends a service query using the service discovery protocol. The application will also authorize the total number of service query copies (denoted as L) that can be dispatched during the service discovery procedure. Only nodes that hold the service query copies can further dispatch or forward service query to other nodes.

After receiving the service query from the application layer, the query source node will generate first the target service code using key words passed from application layer. Then it will start a service query procedure by broadcasting a service match packet containing query node ID, target service code, and the time (denoted as t_q) that elapsed since it met the node (referred to as query destination node) providing the desired service. If the query node never met the query destination node, t_q is set to a very large number. At the same time, two timers are setup at the query node. One is called back-off timer which will expire after a predefined time t_b. The other is called time-over timer which will expire after a predefined time t_o ($t_o > t_b$).

The neighbor nodes receiving the service match packet will search its encountered node list for a match between the target service code and the service codes in the list (thus, our algorithm is linear in the number of the encountered nodes). For each match, the time t_n since the corresponding neighbor node met the query destination node is set to the difference of the current time and the corresponding t_m. If $t_n > t_q$, which means that this neighbor node met the query destination node earlier than the query node, the neighbor node will remain silent to save energy and bandwidth. Otherwise, it will send a packet with t_n back to the query node after waiting a back-off time uniformly randomly chosen between 0 and t_b. Since t_n is a difference of times measured by the neighbor node's local clock, our method does not require synchronization of the clocks in the network nodes. The query node stores the received information in a temporary cache. When back-off timer expires at the query node and there is at least one qualified information in its cache, the query node will first cancel the time-over timer and then it will dispatch n_i copies to the responding neighbor node i according to formula (1):

$$n_i = \frac{L_{left}}{2}\left(1 - t_i \Big/ \sum_{j=1}^{M} t_j\right) \tag{1}$$

where L_{left} is the number of service query copies left at the current query node, t_i is the time elapsed since neighbor node i met with the query destination node, and M is the number of responding nodes that are still active neighbor nodes of the query node.

If there is no qualified information in the cache, it means that no current active neighbor node has met the query destination node later than the query node. Thus, there is no advantage to dispatch service query copies to any of the neighbor nodes. However, it is not efficient for the query node to hold multiple service query copies for a long time just waiting to meet some node that has encountered the query destination node later than itself. Instead, spraying the service query copies to moving neighbors is much better, as it increases the chance of discovering desired service. Thus, in such a situation, the query node will wait until the time-over timer expires and then will dispatch one copy of the service query to each of the currently active neighbor nodes. If there are too many active neighbor nodes, only L_{left} /2 of them will be randomly selected to get the copies.

The service query dispatch information is organized into a list of number of service query copies authorized, indexed by the neighbor node ID. It is broadcasted to all the neighbor nodes at once. Each neighbor node receiving this query dispatch packet will check through the list. If its ID is on the list, it will hold the corresponding number of service query copies. Compared to sending dispatch information to each neighbor node individually, this solution reduces the energy costs and decreases the chance of packet collision.

Each node holding service query copies will repeat the procedure introduced above (starting with sending service match packet) whenever it encounters a new neighbor node not recorded in its list. When there is only one copy left, the query node will only forward this copy to the currently active neighbor node that met the query destination node later than itself.

Service reply

Once a node holding service query copies meets the query destination node, the service discovery procedure is finished and the service reply procedure starts. The query destination node will authorize sending L service reply packets, each containing the source query node ID, service code, and service data.

The dispatch and forward of service reply procedure is similar to service discovery except that it uses a time hint which is the time that has elapsed since the forwarding node encountered the query source node. When dispatching query and reply copies, query node copies just one query or reply packet to neighbor node together with the number of copies (e.g., *l*) that this neighbor is authorized to dispatch, instead of copying *l* query or reply packets.

When one of the service reply copies finally reaches the query source node, that node will start a network-wide broadcast of acknowledgements to stop all the nodes that still hold the query or reply copies from trying to dispatch or forward them.

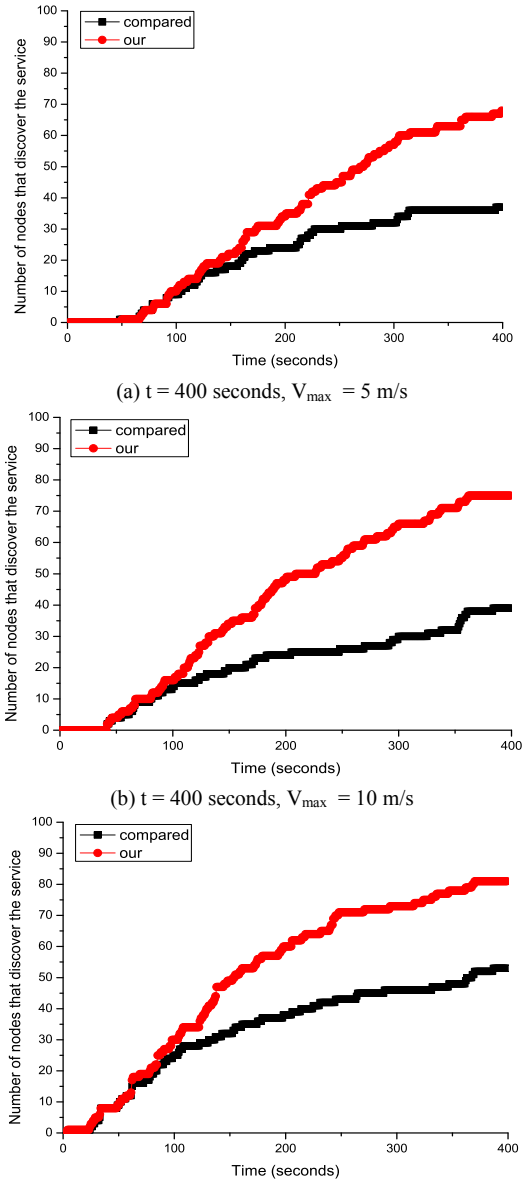
IV. SIMULATION

We used NS-2.33 simulator to evaluate the proposed scheme in terms of service discovery success ratio, service discovery delay and average service discovery overhead. We compared our method with Proximity-based method described in [10] in terms of the service discovery delay and the ratio of service discovery success, which is accomplished when the service match packet encounters the destination node providing desired service.

The average service discovery overhead is measured by the number of packets sent during the service discovery procedure. Most of these packets, sent periodically by each node, advertise node services to its neighbors. The number of such packets is proportional to the simulation time and number of nodes (we assume that both our scheme and the compared method generate those messages with the same frequency). For our method, we also computed the additional overhead resulting from specific service discovery packet sent from the time when the application starts the service query until the query source node gets the first query reply (so the packets sent during the acknowledgement are not included).

The simulated network is composed of 100 nodes initially deployed uniformly randomly within a 300 m by 300 m area. We used two-ray-ground propagation model and each node's maximum transmission range is 10 m (this is a typical DTN environment). Each node repeatedly moves to a randomly selected point in the networks with a random speed ranging from 0 to V_{max}. IEEE 802.11 is used as the MAC and physical layer protocol. Each node provides one service and we used 4 hash functions to generate the service code with length of 16. Under this configuration, the probability of false positive match is only 0.27% [12].

We ran two groups of simulations, in which service discovery always started after 20 seconds of run. For the first group, we fixed the simulation time at 400 seconds but varied the maximum speed V_{max} of each node from 5 m/s to 15 m/s with increment of 5 m/s. For the second group, we fixed V_{max} at 5 m/s but varied the simulation time from 400 seconds to 1160 seconds with increment of 380 seconds. For each V_{max} and simulation time configuration, we generated five moving scenarios. For each moving scenario, we ran 20 service discoveries with random query source node and random target service, thus 100 random service discoveries for each configuration. The hello message interval was set to 1 second and the total number of service query copies was set to 12, i.e. L=12. t_b=1 second. Finally, t_o was set to 3 seconds. All simulation results are based on the average value of all simulations.



(a) t = 400 seconds, V_{max} = 5 m/s

(b) t = 400 seconds, V_{max} = 10 m/s

(c) t = 400 seconds, V_{max} = 15 m/s

Fig.2 Service discovery performance for simulation group one

Fig. 2 shows the number of nodes that discover the service against time for the first group of simulations. It also shows the service discovery success ratio which increases with time for both methods, but the one for our method is much higher than for the compared one. This is because we used service discovery procedure to find the desired service instead of just waiting for the source node to encounter the destination node. We can also see that the service discovery success ratio also increases as V_{max} grows from 5 m/s to 15 m/s. For our method, the success ratio raises from 68% to 81%, and is still much higher than the compared method (for which it raises from around 30% to 50%).

This is because as the node speed increases, nodes meet much more frequently during the constant service discovery time. Hence, the chance of meeting a new neighbor node that may be qualified to hold the service query or service reply data increases. Moreover, the chance that nodes holding service query will meet the query destination node during the query procedure as well as the chance that nodes holding service reply data will meet the query source node during the query reply procedure increase too.

The service discovery delay for both methods for the first group of simulations is shown in Table I. For each maximum node speed, we compare the average delay when certain percent of the services that are successfully discovered by both methods. It is clear that the average service discovery delay for our method is much shorter than the compared one for all situations. This is because we spray the service discovery packets into network, which increases the chance that the destination node will be encountered much earlier. We can also see that the delay decreases as V_{max} increases from 5 m/s to 15 m/s for both methods. For our method, this is because the nodes holding service query or service reply data will meet new neighbor nodes that are qualified to forward packet or meet the service query destination (or source) node much faster with the increased node speed. For the compared method, this is because the source node will meet with the destination node that provides the desired service faster with the increased node speed.

TABLE I
THE SERVICE DISCOVERY DELAY FOR SIMULATION GROUP ONE

	10%		20%		30%	
Speed	Our	Compared	Our	Compared	Our	Compared
5	95	103	140	160	173	300
10	67	80	110	150	132	227
15	51	51	78	82	98	131

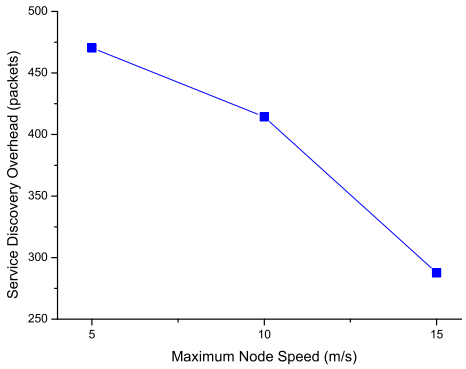
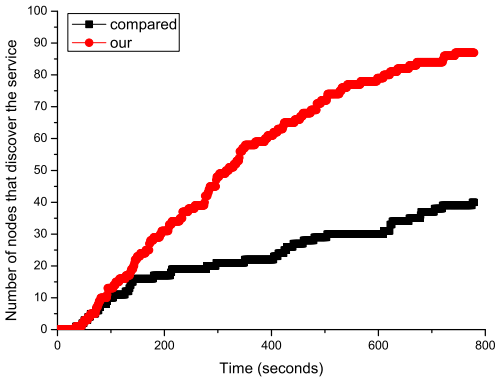
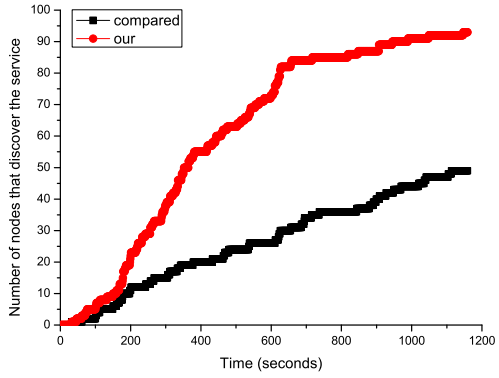


Fig.3 Service discovery additional overhead for simulation group one

Fig. 3 shows the additional service discovery overhead for the first group of simulations for our method[1]. We can see that the number of packets sent during the service discovery procedure decreases from nearly 500 to less than 300 as V_{max} grows from 5 m/s to 15 m/s. The overhead mainly consists of two parts: service match and service dispatch. As the node speed increases, nodes holding service query or service reply data will send more service match packets because they will meet more new neighbor nodes. They will send more service dispatch packets because they will meet more nodes that may be qualified to hold the service query or service reply data. Moreover, they will also be likely to meet the query destination (or source) node much earlier than in case of lower speeds and such a meeting will stop the growth of the overhead. The simulation results clearly show that the overhead is decreasing as the node speed increases, which indicates that the service query discovery is accomplished earlier cutting the growth of the overhead.



(a) t = 780 seconds, V_{max} = 5 m/s



(b) t = 1160 seconds, V_{max} = 5 m/s

Fig.4 Service discovery performance for simulation group two

Fig. 4 together with Fig. 2(a) gives the service discovery success ratio for the second group of simulations. In this case, the service discovery success ratio of our method increases from 68% to 93% as the service discovery time varies from 380 seconds to 1160 seconds. The compared method increases its success ratio from around 30% to 50%. This can be explained as follows. Increase in the service discovery time allows nodes to meet many more times even if the speed remains unchanged. Thus, the same behavior arises as when the node speed increases, while the service discovery time remains unchanged.

The service discovery delay for both methods for the second group of simulations is shown in Fig. 5. For each simulation time, we compare the average delay when certain percent of the services are successfully discovered for both methods. It is clear that the average service discovery delay for our method is much shorter than the compared one for all situations, especially when

[1] By definition, the compared method has no additional overhead. However, the additional overhead is a small percentage of the basic overhead resulting from periodic advertisement of services, which is the same for both methods.

the simulation time is long. The reason is the same as the first group of simulations.

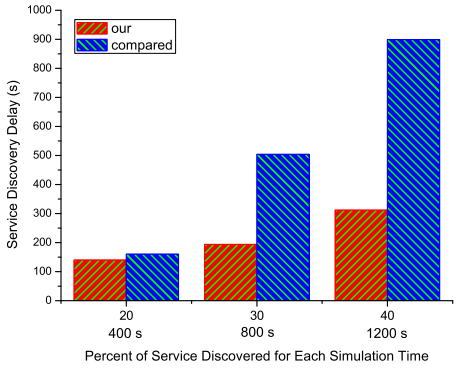


Fig.5 The service discovery delay for simulation group two

Fig. 6 shows the service discovery additional overhead for the second group of simulations. The plot shows that the number of packets sent during the service discovery procedure increases from less than 500 to nearly 600 as the service discovery time grows from 380 seconds to 1160 seconds. In this situation, nodes holding service query or service reply data will meet many more new neighbor nodes. Hence, the additional overhead will increase as well, as already mentioned above in the first group of simulations. The service discovery delay indicates that the service discovery procedures cannot finish quickly enough for additional overhead not to grow significantly. Thus, the additional overhead will increase with the service discovery time.

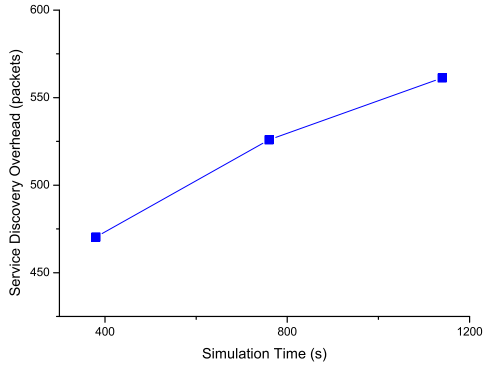


Fig.6 The service discovery additional overhead for simulation group two

V. Conclusion and Future Work

In this paper, we introduced a novel service discovery protocol designed for delay tolerant networks. The novel aspects of the work include the following. 1) Using Bloom filter technique to describe services through a fixed size code. Additionally, we restrict the service advertisements to one-hop distance. Thus, the introduced protocol is very efficient during service advertisement and search. 2) Keeping records in each node of nodes encountered and their services. Thus we can forward service queries and service reply data based on random walk search using node meeting history as a hint. 3) Using limited spray-and-wait mechanism for sending service query and replay packets.

We have studied the performance of our protocol through simulations under different configurations. We observed that our protocol achieved good performance in service discovery success ratio and delay, much better than the one achieved by the compared method. Moreover, an additional overhead incurred by our protocol was very modest. The performance of our protocol is sensitive to maximum node moving speed and service discovery time.

We plan to further study setting of additional parameters (such as *L,* the maximum number of service discovery packets sent) in our protocol to understand their influence on the performance. Additionally, we are also interested in the service reply acknowledgement method and its influence to protocol's performance.

References

[1] R. A. Mallah and A. Quintero, "A light-weight service discovery protocol for ad hoc networks," Journal of Computer Science, vol. 5(4):330-337, 2009.

[2] J. Tyan and Q. H. Mahmoud, "A comprehensive service discovery solution for mobile ad hoc networks," Mobile Networks and Applications, vol. 10(4): 423-434, 2005.

[3] M. Klein, B. Konig-Ries and P. Obreiter, "Service rings - a semantic overlay for service discovery in ad hoc networks," Proceedings of the 14th International Workshop on Database and Expert Systems Applications, pp. 180-185, 2003.

[4] K. Seada and A. Helmy, "Rendezvous regions: a scalable architecture for service location and data-centric storage in large-scale wireless networks," Proceedings of the 18th International Parallel and Distributed Processing Symposium, pp. 218-226, 2004.

[5] M. Nidd, "Service discovery in DEAPspace," IEEE Personal Communications, vol. 8(4): 39-45, 2001.

[6] S. Helal, N. Desai, V. Verma and C. Lee, "Konark - a service discovery and delivery protocol for ad hoc networks," Proceedings of the 3rd IEEE Conference on Wireless Communication Networks (WCNC), pp. 2107-2113, 2003.

[7] D. Chakraborty, A. Joshi, Y. Yesha, and T. Finin, "Toward distributed service discovery in pervasive computing environments," IEEE Transactions on Mobile Computing, vol. 5(2): 97-112, 2006.

[8] A. Obaid, A. Khir and H. Mili, "A routing based service discovery protocol for ad hoc networks," Proceedings of the Third International Conference on Networking and Services, pp. 108-118, 2007.

[9] A. N. Mian, R. Beraldi and R. Baldoni, "Identifying open problems in random walk based service discovery in mobile ad hoc networks," Proceedings of the 6th International Workshop on Innovative Internet Community Systems, 2006.

[10] Y. Maheo, R. Said and F. Guidec, "Middleware support for delay-tolerant service provision in disconnected mobile ad hoc networks," IEEE International Symposium on Parallel and Distributed Processing, pp. 1-6, 2008.

[11] I. Sheriff, P. A. K. Acharya, A. Sampath, B.Y. Zhao and E. M. Belding, "Integrated data location in multihop wireless networks," Proceedings of the second International Conference on Communication Systems Software and Middleware, pp. 1-10, 2007

[12] L. Fan, P. Cao, J. Almeida, and A. Broder, "Summary cache: A scalable wide-area Web cache sharing protocol," in Proc. of ACM SIGCOMM Conf., Sept. 1998, pp. 254–265.

[13] T. Spyropoulos, K. Psounis and C. S. Raghavendra, "Spray and wait: an efficient routing scheme for intermittently connected mobile networks," Proceeding of the ACM SIGCOMM workshop on delay tolerant networking, pp. 252-259, 2005.

[14] D. F. Henri, G. Matthias and V. Martin, "Age matters: efficient route discovery in mobile ad hoc networks using encounter ages," Proceedings of the 4th ACM international symposium on Mobile ad hoc networking & computing, pp. 257-266, 2003.

[15] T. Spyropoulos, K. Psounis and C. S. Raghavendra, "Spray and focus: efficient mobility-assisted routing for heterogeneous and correlated mobility," Proceedings of the Fifth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, pp. 79-85, 2007.