

Service Discovery Using Bloom Filters

Patrick Goering

Geert Heijen

Department of Computer Science,
University of Twente,
P.O. Box 217, 7500 AE Enschede, The Netherlands
patrick.goering@utwente.nl geert.heijen@utwente.nl

Keywords: Service Discovery, Bloom filters

Abstract

A protocol to perform service discovery in ad-hoc networks is introduced in this paper. Attenuated Bloom filters are used to distribute services to nodes in the neighborhood and thus enable local service discovery. The protocol has been implemented in a discrete event simulator to investigate the behavior in case of a multihop mobile ad-hoc network with nodes that all have services to offer. Methods to optimize the used bandwidth, which is a scarce resource in wireless networks, are investigated. Experiments performed with the simulator suggest that the proposed service discovery system enables users to find local services in a multihop ad-hoc network efficiently. The costs for advertising can be kept low, whereas the additional costs for queries set due to so-called false positives are moderate.

1 Introduction

Nowadays small devices are getting more and more popular. In the near future many people will carry multiple devices like a phone or a PDA with connected peripherals. People want to be able to find and connect these devices together as they may need information or services from one of their devices. Further people want to be able to find devices of other users or organizations in the vicinity that offer services to the public that they can make use of. Several solutions exist for service discovery, but most have disadvantages and cannot be used as such in the context of a wireless multihop ad-hoc network as described here.

This paper introduces a service discovery protocol for local ad-hoc networks based on the use of attenuated Bloom filters. Our solution focuses on service discovery in the local area, where a large number of nodes are mostly connected through a wireless technology in an ad-hoc multihop fashion. The amount of

bandwidth used by sending advertisements depends on the number of nodes in the network and the distance in which services can still be found. Another important aspect of our protocol is the time it takes for a service to propagate through the network. Further we want to reduce the number of advertisement messages needed to propagate a service change through the network. Finally we want to know the effect of changing the Bloom filter width with respect to the query load caused by so-called false positives in the network.

This paper is organized as follows. Section 2 describes other work related to service discovery in ad-hoc networks. Section 3 provides an overview of Bloom filters, attenuated Bloom filters, and a method to use those in our service discovery solution. Section 4 introduces our protocol and discusses some alternative approaches. Section 6 gives simulation results of the protocol, and Section 7 presents the conclusions and future work.

2 Related Work

For service discovery in computer networks several protocols have been developed, each with their own strengths and weaknesses in different areas. We can distinguish between centralized and distributed solutions. Centralized protocols use a central node, a directory, which stores all services available, while in a distributed system all nodes keep information about a part of the available services.

Directory based service discovery systems like SLP [1, 2] most of the time assume a connection to some infrastructure. Instead of a single directory server there could be multiple directory servers available for redundancy. A hierarchy of directory servers can be used to make the system more scalable. Although this does not need to be a single point of failure, all devices in the network have to communicate with a directory server. The communication path and

thus the nodes in the communication path towards a directory server will likely be loaded more than the rest of the network. This is undesirable for mobile devices in ad-hoc networks that have limited network capacity and power. Moving nodes make it more difficult to keep a stable communication path, as these protocols were not designed for these kind of networks. Furthermore the directory server itself might disappear or get out of range for some nodes.

A distributed system has some advantages in a mobile ad-hoc network [3] and can be proactive or reactive. In a proactive system services are announced through broadcasts, while in a reactive system queries for services are propagated through the network.

Zeroconf [4], e.g. implemented as Apple Bonjour, is an IETF protocol that enables the discovery of services on a local area network. A usable IP network is automatically created without the need for configuration or special servers, but it is limited to a single subnet.

A peer to peer (P2P) based solution has the advantage of being distributed over a larger number of nodes in the network. E.g. Chord [5] can be used to distribute objects evenly over a large number of nodes, but the location of a service description can be placed anywhere in the network. It looks more efficient and robust to have services and descriptions at least close to each other. When a network cluster gets disconnected from the infrastructure, all local services should still be available to the nodes in that cluster. Further a group or cluster of nodes has to be established before the system can be used. Also when all nodes have knowledge about all available services there will be problems with scalability. In the Intentional Naming System (INS) [6, 7] this is solved by separating sets of nodes in virtual spaces. Nodes are only aware of all services in their virtual space and have to rely on a directory server entity to find services in other virtual spaces.

In [8] the newscast epidemic protocol is used to provide a robust overlay network that adapts to large changes in a dynamic network. It uses quite some bandwidth to accomplish this, which makes it less suitable for wireless networks. For service discovery in ad-hoc networks, where we want to discover services located nearby, we need a fully distributed system, suitable for multihop networks. Many nodes in this ad-hoc network will be mobile with a wireless network interface. Furthermore the system should work as soon as a new node arrives, without the need to pre-establish a cluster or group.

Attenuated Bloom filters are used in [9] for context discovery. There, an analysis is done on the false positive chance and the size and depth of Bloom filters.

3 Service Discovery Using Attenuated Bloom Filters

3.1 Overview Bloom Filters

Bloom filters were introduced in [10] as a hash coding technique that provides a trade-off between space usage or hash size and time needed to test the membership of a text string in a given set of strings. Several strings are represented in one array of bits. A small chance of false positives is allowed, that is a string is not a member of the given set while the system claims it probably is. A Bloom filter consists of an array of w bits, initially all set to 0. A number of b independent hash functions is used to map a text string to the Bloom filter. For every string represented by the Bloom filter b bits are set as specified by the hash functions. A false positive appears when a string is represented by bits already set by one or more of the other strings represented in the Bloom filter. For each hash result, there may be a hash of one or more other strings that give the same result.

A false positive might not be a big problem as long as the chance of it occurring is small enough. When an application tries to contact a resource that does not exist, because of a false positive, the application will find out and try another resource that also matches the query. As we want to use Bloom filters in a distributed manner, these queries generate network traffic. In wireless ad-hoc networks bandwidth and battery power are scarce resources, therefore we want to prevent unneeded traffic in the network as much as possible. At the same time the delay to receive a response back for a query should be small, even when some nodes are mobile.

Table 1: Bloom filter example

0	1	2	3	4	5	6	7
1	1	0	1	0	1	1	0

In the example Bloom filter given in Table 1 several services are represented. When a user wants to test whether a color printer is one of these services, a hash function will be used on the string "Color Printer". Suppose this hash function returns (0, 3, 6). This means the color printer is probably represented in this filter as the bits 0, 3 and 6 are all enabled. When the hash function on the string "Camera" returns (1, 4, 5) this signifies that the camera service definitely is not a member of the set of service in the Bloom filter, as bit number 4 is false. As strings are added to the Bloom filter, more bits in the filter get set. Also the possibility of overlap in the bits that are set for specific services will grow with the number of strings the filter represents.

There is a trade-off between the probability of false positives, the size of the filter, and the time needed

to compute the hash values. In our system computing these hash values is an operation that needs to be done only once per query in the node that wants to send the query and once per service in a node that advertises services. We do not consider this hashing to be a bottleneck in our system. The bandwidth usage can be optimized by choosing an appropriate filter size, because in case of a false positive more packets need to be sent for the same query. Bloomfilters have some interesting properties: they are simple, efficient in testing whether a string is represented and efficient with space. The number of bits needed to represent a string is low, thus allowing for an efficient transmission of queries. The computation needed to compare a filter with a query is a comparison whether all bits are set. Also several filters can be combined in one filter by using the OR operator on the original filters. As more services are represented in the Bloom filter the chance of a false positive gets larger. The intersection of filters, as a measure of similarity, can be taken by the AND operator. This can be used to check whether a service is represented by the Bloom filter.

Although Bloom filters are already quite small, it is possible to compress them before transmission [11]. The Bloom filter width can be increased while the packet size stays the same, thus allowing a lower false positive probability. Or for the same false positive probability the bandwidth used can be decreased. The disadvantage is that it takes time and computing power to compress and decompress the Bloom filters, especially on some of the simple devices that also need to be supported.

3.2 Overview Attenuated Bloom Filters

In [12] attenuated Bloom filters have been introduced as a method to optimize the performance of location mechanisms especially when objects to be found are located nearby. An attenuated Bloom filter is an array of standard Bloom filters of depth d . Every row in the attenuated Bloom filter represents objects at different distances. The further away an object is located the more the respective filter is attenuated. Thus local replicas of an object are given a higher priority. Every outgoing link will have a separate attenuated Bloom filter. This makes it possible to select a link where an object most likely can be found.

Table 2: Attenuated Bloom filter example

	0	1	2	3	4	5	6	7
First Hop	1	1	0	1	0	1	1	0
Second Hop	1	0	0	0	1	0	0	1
Third Hop	1	1	1	1	0	0	1	0

In the attenuated Bloom filter of Table 2 there are three layers. E.g. the service "Color Printer" (0, 3, 6) is available both one and three hops away, while the service "speakers" (0, 4, 7) can be found in two hops.

3.3 Service Discovery With Attenuated Bloom Filters

Every node has an attenuated Bloom filter for each of its neighbors. In this context a link refers to a direct connection with a neighbor, so there are as many attenuated Bloom filters as there are neighbors. When a query arrives at a node, the node will check its attenuated Bloom filters to find an outgoing link that is a likely direction where the requested service can be found. The number of layers per attenuated Bloom filter is the same for all nodes in the network. The first level of the attenuated Bloom filter contains only the services that are one hop away. The second filter can contain services that are two hops away and so on. This can be used to give an advantage to services that are nearer in the overlay network. The larger the distance from a node, the more services will be contained in the corresponding attenuated Bloom filter. This means a larger chance of false positives. The Bloom filter can be seen as a way to summarize available services. This could especially be useful when many services are available. Closer to the destination more accurate information will be available to better match the query. When we have an attenuated Bloom filter of depth d we can discover services that are located up to d hops away.

4 The Protocol

4.1 Overview

There are many nodes in the neighborhood, connected in an ad-hoc manner. Nodes will announce the services they have to offer through local broadcast messages, see section 4.2. All nodes will keep information about what services can be reached through their direct neighbors in attenuated Bloom filters. A node will store d attenuated Bloom filters per node it receives a broadcast from, which represent the number of hops away the service might be located. A broadcast message will contain the Bloom filter that represents services offered by the node itself and the $d-1$ attenuated Bloom filters that represent services from other nodes. The d^{th} attenuated Bloom filter is not sent as it would be discarded by the next hop. Broadcast messages will be sent in several cases: first when there is an addition or deletion of a service in a node. Second when a broadcast packet that contains new information is received. Third when a node detects a new neighbor we want to announce the services reachable through that node. Fourth there will be periodic broadcasts to allow nodes that move into range to discover the services and to keep the information up to date. The periodic broadcasts are also used to clean up services that have not been announced for several periods, e.g. because the service is no longer available or the node has moved away.

When a node wants to find a specific service it will look in the information it has of the neighborhood and send a query message when the service is likely to be present, see section 4.3. These query messages will be forwarded toward the destination as long as every node on the path has information about the direction the service can likely be found. Note that there could be multiple directions with a match for the service. When a node that has the service receives the query message, it will send back a response message to the node that originated the query, see section 4.4.

4.2 Sending broadcasts

Sending of broadcasts will be done at different occasions as explained in Section 4. When there is a change in the network, it is likely a node receives broadcast packets from multiple nodes that saw this change. To limit the amount of broadcast packets, sending of packets is delayed. We define a broadcast window that starts when a broadcast packet is received. If during this window more broadcast packets are received, we collect the information in all of these packets. The node will send a broadcast packet after this window only if there were changes with respect to the previous broadcast sent by this node.

In the case of multiple interfaces in one node we might be able to prevent some unnecessary traffic. A node can transmit broadcasts only to nodes that do not have the updated information yet. In the case of a fixed network this can be done by sending a broadcast further only on other interfaces and exclude the interface the update was received from. In the case of ad-hoc networks it is more complicated. There is one wireless interface through which other nodes can be reached, but it is unknown whether the nodes received each others messages. Thus some of those nodes might be unaware of the updated service information received in the broadcast message. This occurs for nodes that are out of reach of the original node that sent the broadcast. Some nodes might have more than one wireless interface using different technologies. When only one node can be reached from a wireless interface over which a broadcast was received it is certainly not necessary to send a broadcast using that interface.

4.3 Sending queries

There are several methods to send queries. All of them have advantages and disadvantages for different kinds of scenarios. Below we will describe parallel, sequential, and a hybrid form of sending queries. Queries are uniquely identified with a query identification (Q_ID) to prevent loops. When a node sees a query it has processed before, it can just drop the query. A problem present for all three methods is when a client wants to get a different result back for

the same query, as it is possible that two or more services match the query. We define a freshness identifier for every packet to detect whether a client wants a different result for the same query with the same Q_ID. A client that wants a different result will keep the Q_ID but increase the freshness. If a node detects a query with a freshness that is the same as a previous message with the same query identification, the message is a duplicate and can be dropped. If the freshness in the packet is higher than the freshness of any previous messages with the same query identification the client requests a different service. The node will forward the query to a different path than previously, when another match exists. We are still investigating how to select the best alternative path and which node should select this path. Nodes will have to keep some state information about queries. For every query that arrives it has to keep the Q_ID, the freshness, and the node the query was received from. We also define a hop count to prevent query messages travelling too far. From the attenuated Bloom filter we know the number of hops in which a service can be found and thus set the hop count accordingly.

4.3.1 Parallel querying

In this alternative all query messages are sent in parallel. When a query message arrives at a node, the node will check all Bloom filters for all directions. The query message will be forwarded for each direction where a match is found, as long as the hop count is not exceeded. This may result in multiple replies to one query.

4.3.2 Sequential querying

The difference with parallel querying is that a query message is sent only to the direction with the best / first match. The best / first match could be determined by first looking at the top layer in the Bloom filter for all interfaces. When a match is found in the direction of one or more interfaces, one link will be selected. Multiple matches on the same level means the same service, or better said multiple services that match the same query where found. We then have to select one, but we cannot distinguish between them on grounds of the given query. We could however look at how close to saturation the Bloom filter is for every match and take the link with the least saturated Bloom filter. The chance for a false positive is smaller in this direction. If nothing is found in the first level, continue with the second level and apply the same algorithm. This means for every query some state has to be kept, as it should be possible to also find services that are not the best match in any intermediate node. A strategy would be to follow a path of best matches until a node with a matching service or a dead end

is reached. A dead end can occur in the case of a false positive in a node. A node further along the path will not have any matches. To solve the situation we can use traceback. This means following the path the original query took to the node where the dead end was discovered in reverse direction. We go back to a previous node, which is probably the node with the false positive, to try a second best match there. We continue doing tracebacks until we find the final destination or we are back at the client. If we are back at the client that means there is no service found to match the query, only false positives. It should be noted that for a given link there might be multiple matches in different layers of the Bloom filter for one specific link. Although the link was already tried, we might want to try again with a higher maximum number of hops to forward the query.

4.3.3 Hybrid querying

There is a tradeoff between parallel and sequential sending of queries. The parallel method will have a smaller round trip time for a query, it takes less time to find out whether a service is not available, as all possible services that match are returned without waiting for previous queries. The downside of the parallel method is the bandwidth usage. It will be higher even without false positives. One destination might be reachable over multiple paths with partially overlapping nodes. This is also good for robustness as some paths might not be available all the time. Then as long as there still is a path the service will be found quickly. The sequential method could take longer and eventually use the same amount of bandwidth or even more. If the false positive rate is small, as it should be, the first match in every intermediate node should be correct and a matching service will be found by following only one path. This results in a reduced bandwidth usage. In the case of only false positives there is the same amount of traffic as for the parallel case, with the addition of the traffic for tracebacks and negative responses. As both methods have some advantages, we can try to combine them.

We can try layer by layer to follow the best matches in parallel and allow them to traceback when the service is not reachable. Each node will keep track of the links it forwarded a specific query to. Then the node can try the next best matches as with the sequential method. In case two queries with the same Q_ID arrive at one node, they can be merged. A message has to be sent to the previous node to signal the query was merged, so the node does not have to wait for a reply.

4.4 Sending responses

When a service has been found, that is a node received a query that matches one of its own services,

the originating node has to be notified. A response message is sent back to the originating node along the same path the query has used. We can do this in two different ways: store the information about the path taken in the nodes that forwarded the query message or store the information in the query message itself. When the information is stored in the query message, the packet size will increase by the number of hops away the service is located times the size needed to describe the node address. This will increase the cost of a false positive. When all intermediate nodes store information about the link they first received the query, they can forward the response message along this path. Nodes already need to store information about queries they receive to prevent loops and duplicate sending of queries. Therefore we chose to implement this last method as there is no additional cost in terms of bandwidth usage.

When the originating node receives the response message it can try to contact the service to determine if it really satisfies its needs. If it does not the node can try to refine the query to get a better match to the service it really needs. Note that the path from the querying node to the service is already known from the service discovery process. Instead of relying on a routing protocol to set up routes, we might set up the route ourselves when the response message is sent back from the node that contains the service to the node that originated the query.

4.5 Duplication

There are some problems related to this system of broadcasting. Multiple nodes will send each other updates of the same services and those services will be duplicated over multiple layers in the Bloom filter, see Figure 1. There are three nodes A, B, and C that all can reach each other and will broadcast all services they know about to each other. There are four layers in the attenuated Bloom filter, the first one containing the nodes own services. The arrows signify where service B is broadcasted to. The attenuated Bloom filter is updated according to: $A_i = B_{i-1} \vee C_{i-1}, \forall i > 0$. The consequence is the exchange of an unneeded number of broadcasts.

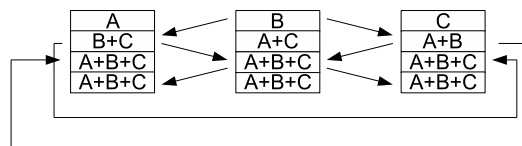


Figure 1: Duplication of service B.

It is possible to prevent this from happening between two nodes directly, but when there are more nodes in between this is difficult to avoid. To prevent all this excessive traffic it might be better to dupli-

cate the service information immediately in all lower layers, starting from the second layer, as this is the shortest loop possible.

4.6 Hash function

The hash function used in the Bloom filter service discovery is very important. It should perform a mapping from a string of characters of arbitrary length to bits in the fixed length Bloom filter. There are families of universal hash functions, which might give good results when used for Bloom filters. We are planning some experiments to verify the dependency on the hash functions. Nodes that want to exchange service-broadcasts should use the same set of hash functions that have to be defined beforehand. We could use different sets of hash functions per group of nodes that want to communicate. From the bits set in the Bloom filter it cannot be seen what services are represented. Any node cannot easily map queries and broadcasts back to service descriptions. A node that knows the set of hash functions used can do a bruteforce attack, but this is also difficult as the hash functions are a mapping from many to one. Nodes can only check whether a given service is represented in the Bloom filter.

5 Analysis

We are interested in the number of broadcast packets we can expect in the network. We assume the nodes in the network are structured in a grid, see Figure 2.

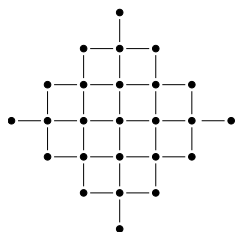


Figure 2: Grid structure

Every node can reach four neighbors, except the edge nodes. The radius r of the network is the number of hops from the center node to an edge node. We assume all services are uniquely represented, that is there are no overlapping bits for any of the services in the network and there is no duplication of services as discussed in Section 4.5. All nodes have a unique service, and the center node will start transmitting a broadcast message. All other nodes will only broadcast when they receive a broadcast and there was a change in their Bloom filter. Because we have a small delay, a node waits for other broadcast packets, we can see a pattern in the broadcasting nodes. An ap-

proximation of the total number of broadcast messages M sent in the network is given in Equation 1.

$$M \approx \sum_{i=1}^{r+1} i^2 + \lfloor \frac{d}{2} \rfloor r^2 + \lfloor \frac{d}{2} \rfloor (r+1)^2, \forall d \geq r \quad (1)$$

First only the center node will send. After that all neighbors of nodes that transmitted in the previous round will send a broadcast message. This will continue until a node filled all layers of its Bloom filters. The first term in Equation 1 represents the increasing number of nodes that send the next round until the edge of the network is reached. Then it will alternate between the nodes that are a even and uneven number of hops from the center node as expressed in the second and third term respectively. In Table 3 the number of broadcast packets for several d and r are given, scaled by the number of nodes in the network.

Table 3: Broadcast packets per node

Radius	d=4	d=5	d=6	d=7	d=8
1	2.2	3	3.2	4	4.2
2	2.4	3.1	3.4	4.1	4.4
3	2.6	3.2	3.6	4.2	4.6
4	2.7	3.3	3.7	4.3	4.7
5		3.5	3.9	4.5	4.9

When not the center node but a node closer to the edge starts transmitting, the number of broadcast packets in the network needed to reach a stable situation will be higher. The worst case will be when an edge node starts transmitting the broadcast message.

6 Simulation

6.1 Simulation Model

We implemented the protocol as described in Section 4 using Opnet modeler [13] as our discrete event simulator. We implemented a separate module that can be connected to the UDP module in any Opnet node. For our experiments we use an ad-hoc wireless node that uses AODV as a routing protocol, in this case a MANET node as included in Opnet.

6.2 Experiments

We connect all nodes in a grid structure, without any mobility. We need 61 nodes to be able fill five layers of the middle node with unique services. Every node announces one unique services to the network. Per node we gather statistics about the number of packets received and transmitted of all different message types in our protocol in different situations. The number of bits set per service $b = 3$ in all experiments. For the first two experiments the Bloom filter width is fixed at $w = 1024$ bits.

6.2.1 Experiment 1

In this experiment we try to find the number of broadcast packets sent in the entire network for several depths d and radius r of the network. We use the same assumptions as in Section 5: we have a grid network where the central node will initialize the sending of broadcast packets. This is for a startup situation, the nodes have no information at all about services available in the neighborhood or neighbors they have. The number of broadcast packets depending on the network size and depth of the attenuated Bloom filter is depicted in Table 4. For small networks this is exactly the same as in Section 5, but from a radius of three on there is a deviation as the formula does not take into account how often the nodes closer to the edge have to broadcast until they reach a stable situation without further changes. Larger networks still need broadcasts near the edge of the network while nodes near the center have all information already. In a real network the number of broadcasts might be lower as there can be false positives that cause a node to not transmit because it thinks there is no new information.

Table 4: Number of broadcast packets

Radius	d=4	d=5	d=6	d=7	d=8
1	2.2	3	3.2	4	4.2
2	2.4	3.1	3.4	4.1	4.4
3	2.7	3.6	4.2	4.2	4.6
4	2.9	3.5	3.8	5.1	5.6
5	2.9	3.7	4.5	5.2	5.8

Figures 3 and 4 show the effect of duplication in the startup phase. At a certain time the center node receives a broadcast messages. From then on information is exchanged between the center node and its neighbors until a stable situation is reached. This stable situation is reached when the nodes in the network have complete information about all neighbors upto d hops away. After a broadcast packet has been received a node waits for a random time between 1 and 1.5 seconds before it sends a packet with updated information. It can be seen that the number of broadcast packets the center node transmits and receives with duplication is a bit better than without duplication. For small networks duplicating is a huge improvement, but as the network radius is approaching the depth of the Bloom filter the difference is getting smaller. This can be explained because in bigger networks there is always a neighbor that has new information. Therefore in the case of duplication nodes will be transmitting almost the same amount of broadcast packets as without duplication.

6.2.2 Experiment 2

This experiment shows the effect of duplication when the center node in the network starts broadcasting a

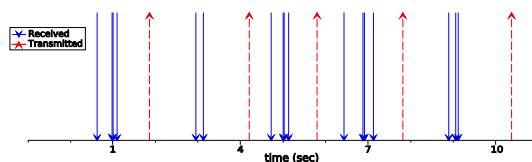


Figure 3: Broadcast packets without duplication.

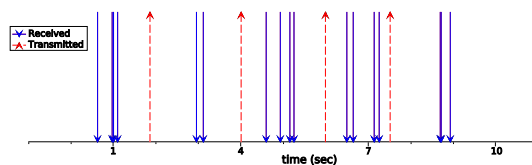


Figure 4: Broadcast packets with duplication.

changed service. All nodes in the network already know about all their neighbors and the services available in the vicinity. We use a grid network of 61 nodes and vary the depth of the filter, both with and without duplication. The effect of a service change can be seen upto d hops from the node with a new service. We consider both the cases where the center node adds and removes a service. In Tables 5 and 6

Table 5: Sent broadcasts without duplication: add

depth	Number of Hops						Total
	0	1	2	3	4	5	
4	2	8	8	12	0	0	30
5	3	8	16	12	16	0	55
6	3	12	16	23	16	20	90
7	4	12	24	24	32	20	116
8	4	16	24	36	32	39	151

Table 6: Sent broadcasts with duplication: add

depth	Number of Hops						Total
	0	1	2	3	4	5	
4	2	8	8	12	0	0	30
5	2	8	12	12	16	0	50
6	2	8	12	16	16	20	74
7	2	8	12	16	20	20	78
8	2	8	12	16	20	23	81

we show the number of packets sent a number of hops from the center node and in total in case of an added service to the center node. In this case duplication performs better, as the number of broadcast packets needed to reach steady state is smaller.

In case a service is removed from a node, it is better to not use duplication. Because the old service is still represented in the network nodes will keep broadcasting this old information until it is replaced in all layers. The center node will see some of its old service appear again in lower layers and sends another broadcast packet because of this change. It is clear that the larger the depth of the Bloom filter, the more duplication is an issue. Duplication can prevent

sending of unneeded packets when a service is added. When a service is removed it does not help, as information can only be added to Bloom filters and not removed. For removal the Bloom filter needs to be changed completely.

6.2.3 Experiment 3

In this experiment the effect of the Bloom filter width is demonstrated. We vary the width from 512 bits to 2304 bits in steps of 256 bits. We keep the filter depth constant at $d = 4$. We have 61 nodes in a grid structure with 4 unique services per node. On all nodes we generate randomly 1000 queries per period and exclude any query that could really be present in the system. We use parallel querying as described in Section 4.3 and measure how many queries are really sent as a result of the random query trials. This will only happen when there was a match in any of the Bloom filters representing a neighboring node. A local match will not cause a query packet to be sent. The number of queries sent from all nodes and the center node separately are scaled to the number of queries tried. This is shown in 5. Note however that we did not optimize the number of bits representing a service b .

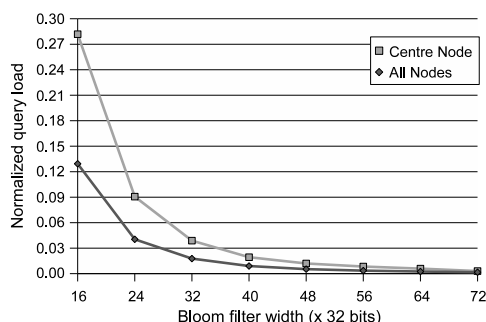


Figure 5: Normalized query load.

A query will only be sent in case of a false positive in the system. There could be a false positive in the center node as result of a query tried directly on this node, but there is also a possibility that a query packet sent by a neighboring node is sent to the center node and causes a false positive there too. The center node has a higher false positive probability than nodes closer to the edge of our grid network. This can be explained by the density of services present, which is the highest in the center node, and gets lower the closer we get to the edge of the network.

7 Conclusions and Further Work

Attenuated Bloom filters can be used for local service discovery in ad-hoc networks. This paper shows the number of broadcast packets needed to keep all nodes up-to-date about services in their area when

using our protocol. Further some investigation is done to try to optimize the number of packets needed for the broadcasting of services, also when there are changes in the services offered.

The experiments with the simulator show the amount of bandwidth used for different network sizes and Bloom filter depths. As expected the average number of broadcast packets per node is larger for networks with more nodes or an increased Bloom filter depth. As soon as the radius of the network is approaching the depth of the Bloom filter the average number of broadcast packets per node converges to a stable value. In a network without prior knowledge about available services where all nodes start sending their known services to all neighbors, it is shown steady state is reached sooner in case we do use duplication. Changing a service in one node afterwards gives different results depending on the change. When a service is added to a node, duplication performs better with respect to the number of broadcast packets needed to reach steady state. When a service is removed from a node however it is shown the strategy without duplication performs better in this respect. Finally the query load caused by false positives is significantly lower as the Bloom filter width increases. Overall the experiments suggest the proposed service discovery system can be used efficiently in a multihop ad-hoc network. The bandwidth used by advertisements and queries can be optimized to a moderate level.

Further work includes charting the effect of sequential and hybrid methods for querying, as there might be many queries in the network and parallel querying is not the most bandwidth efficient in all cases. When there is node mobility, we expect the number of broadcast packets needed to go up compared to a static situation. We want to show this effect and find some effective methods to limit this. The hash functions used in this protocol have an influence on how good the protocol works. We want to find how much influence they have and how we can find more optimal hash functions for our protocol.

Acknowledgement

This work is supported by the Dutch Ministry of Economic Affairs under the Innovation Oriented Research Program (IOP GenCom, QoS for Personal networks at Home).

References

- [1] J. Veizades, E. Guttman, C. Perkins, and S. Kaplan. Service location protocol, rfc 2165, 1997.
- [2] S.E. Czerwinski, B.Y. Zhao, T.D. Hodes, A.D. Joseph, and R.H. Katz. An architecture for a se-

- cure service discovery service. In *Mobile Computing and Networking*, pages 24–35, 1999.
- [3] J. Hoebeke, I. Moerman, and B. Dhoedt. Analysis of decentralized resource and service discovery mechanisms in wireless multi-hop networks. In *Proc. WWIC 2005*, Xanthi, Greece, May 11–13 2005.
 - [4] S. Cheshire, B. Aboba, and E. Guttman. Dynamic configuration of ipv4 link-local addresses, rfc 3927.
 - [5] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In *Proc. of the 2001 ACM SIGCOMM Conference*, pages 149–160, 2001.
 - [6] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The design and implementation of an intentional naming system. In *Proc. 17th ACM SOSP Conf*, 1999.
 - [7] M. Balazinska, H. Balakrishnan, and D. Karger. Ins/twine: A scalable peer-to-peer architecture for intentional resource discovery. In *Proc. of Pervasive 2002*, 2002.
 - [8] S. Voulgaris and M. van Steen. An epidemic protocol for managing routing tables in very large peer-to-peer networks. In *Proc. 14th IFIP/IEEE International Workshop on Distributed Systems: Operations and Management, (DSOM 2003)*, 2003.
 - [9] F. Liu and G. Heijenk. Context discovery using attenuated bloom filters in ad-hoc networks. to appear in proceedings WWIC2006, 2006.
 - [10] B.H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7):422–426, 1970.
 - [11] M. Mitzenmacher. Compressed bloom filters. In *Proc. of the 20th Annual ACM Symposium on Principles of Distributed Computing*, IEEE/ACM Trans. on Networking, pages 144–150, 2001.
 - [12] S.C. Rhea and J. Kubiatowicz. Probabilistic location and routing. In *Proc. of INFOCOM 2002*, 2002.
 - [13] OPNET modeler software, available: <http://www.opnet.com/products/modeler>.