

# Context Discovery Using Attenuated Bloom Filters in Ad-hoc Networks

Fei Liu and Geert Heijenk

**Abstract**—A novel approach to performing context discovery in ad-hoc networks based on the use of attenuated Bloom filters is proposed in this paper. A Bloom filter is an efficient space-saving data structure to represent context information. Attenuated Bloom filters are used to advertise the availability of context information multiple hops away, and to guide queries to discover it. In order to investigate the performance of this approach, a model has been developed. This paper describes the model and its validation. From the results obtained with the model, using attenuated Bloom filters appear to be a very promising approach for context discovery in ad-hoc networks. Compared to conventional solutions, the traffic load of our approach is an order of magnitude lower in practical situations.

**Index Terms**—Context discovery, Bloom code, ad-hoc networks.

## I. INTRODUCTION

Ad-hoc networks are non-infrastructure wireless networks in which most of the terminals are both mobile and power-consumption constrained. When one needs to obtain a service or context information from other devices, querying and determining the location of the service or context information source might generate a lot of traffic. In a network with a high query rate, such traffic can be rather heavy. As a result, the terminals consume quite an amount of power and bandwidth for querying. An efficient context discovery mechanism needs to be developed for such situations.

This paper extends and improves the work presented in [12]. It describes the development of a discovery mechanism for networks that are context aware. These networks utilize context information to improve their operation, or to enrich the services provided to users. We propose a novel approach to discover context information sources in an ad-hoc network based on the use of attenuated Bloom filters, which allow for a decentralized space-efficient discovery method. Instead of broadcasting full information about the type and location of context information, nodes send attenuated Bloom filters which contain context type information for all the reachable nodes up to a certain number of hops away. Queries are only forwarded in the directions which possibly contain the required information. Due to the highly compressed representation of context type data, Bloom filters have a

special feature of false positive probability, which leads to probabilistic querying. Our analysis reveals that this type of probabilistic discovery method can substantially reduce the network load compared to discovery using traditional approaches.

This paper is structured as follows. Section 2 will introduce our novel approach for performing context discovery using attenuated Bloom filters and related work. In Section 3 we will describe performance models for the transmission costs of our method for two different network structures. Further we compare them with context discovery without attenuated Bloom filters. In Section 4 we will provide numerical results. Finally, in Section 5, we present our conclusions and propose future work.

## II. CONTEXT DISCOVERY USING ATTENUATED BLOOM FILTERS

### A. Related Work on Context Discovery

Context discovery has a lot of resemblance to service discovery. In computer science, context refers to the circumstances under which a device is being used, while service can be any software or hardware entity that a user might be interested to utilize [19]. Both context and service can be described by a certain format or template. The discovery of both context and service can be understood as an action of looking for requested context/services in the network. Service Discovery Protocols (SDPs) can be classified into centralized and decentralized architectures. In ad-hoc network environment, nodes are both mobile and mostly battery-powered. Those characteristics fit well with some features of decentralized architectures. The choice for a proactive or reactive SDP in decentralized architectures depends substantially on the network and service context and on the interaction with the underlying routing protocol [9].

Service descriptions are different for various SDPs. The most popular format is the attribute-value structure [14]. For instance, the Service Location Protocol (SLP) [8] uses service templates which predefine the attributes in a template document readable by humans and machines. Service agents (SAs) advertise the location of one or more services; directory agents (DAs) store service location information centrally. Whenever necessary, user agents will look for the required services at SAs and DAs. The Bluetooth Service Discovery Protocol (SDP) [3] defines a service record consisting of the entire list of attributes, which is then stored in the SDP server. Clients will send requests to the SDP server to obtain the required services.

Further, hierarchical attribute-value pairs, which mostly rely on eXtensible Markup language (XML), are also used

Manuscript received August 16, 2006; revised on November 10, 2006. This work was supported in part by the Freeband AWARENESS project (<http://awareness.freeband.nl>).

F. Liu and G. Heijenk are with University of Twente, P.O. Box 217, 7500 AE Enschede, the Netherlands. (Phone: +31-53-4893095; fax: +31-53-4894524; e-mails: fei.liu@utwente.nl, geert.heijenk@utwente.nl).

in some protocols, such as Global Service Discovery Architecture (GloSev) [1] and Group-based Service Discovery protocol (GSD) [5]. GloSev is proposed for worldwide and local area network usage. Services are described and categorized hierarchically by using the Resource Description Framework (RDF) which is based on Uniform Resource Identifiers (URI) and XML. The hierarchical service architecture is similar to the DNS domain name architecture. GSD is a distributed service discovery protocol for Mobile Ad-hoc NETWORKS (MANETs). Services are described based on DARPA Agent Markup Language (DAML+OIL). Advertisements are sent periodically to nodes within a maximum number of hops. Each node has peer-to-peer caching to keep a list of local and remote services that a node has received from advertisements. Services are also grouped to ease service discovery by selectively forwarding queries.

In some protocols, such as Jini [18], attributes are described as Java objects. Service objects are registered in service registries, which are also used to look up services. A client needs to download the service object and invoke it to access the service.

Among the protocols mentioned above, GloSev was developed for wide area networks. SLP and Jini were designed for local area networks. GSD and Bluetooth SDP are specifically for MANETs. Further, it is clear that whatever method is used to describe services, sending the complete service attributes causes heavy traffic. This is inefficient in a high-density mobile ad-hoc network with many services to be advertised and/or queried. Due to the limited battery power of terminals, a simple, efficient context description and discovery mechanism is required. Clearly, a mobile ad-hoc network is less suitable for a centralized structure due to the mobility of the nodes. Nodes should not depend on other specific nodes to reach the required context information. Context discovery using attenuated Bloom filters can solve these problems in ad-hoc networks.

### B. Brief Introduction to Attenuated Bloom Filters

A Bloom filter [2] is a data structure for representing a set in order to support membership queries. It can denote a set simply and efficiently, with a small probability of false positives. Bloom filters can be used in various network applications, such as distributed caching, P2P/overlay networks, resource routing, packet routing, and measurement infrastructure [4]. Bloom filters were also proposed to be used as an efficient approach for lossy aggregation and query routing for a Secure Service Discovery Service in [6]. Recently, researchers have explored the application of Bloom filters to ad-hoc networks, such as speeding-up cache lookups [17], group management [13], hotspot-based trace back [10], and neighbor solicitation [15].

A Bloom code can represent a set of context information types. Here, we assume that a context information type is denoted by a string of arbitrary length. Each context type will be coded by using  $b$  independent hash functions agreed upon globally over the range  $\{1 \dots w\}$ , where  $w$  is the width of the filter. The default value for each bit in the Bloom code is 0. The bits of positions associated with the hashes

will be set to 1. When querying for a certain type of context information, the same hash functions are performed. If all positions in a Bloom filter indicated by one of the hashes contain a 1, the presence of the queried context type is likely (but not certain). Otherwise the context type is not present. The use of these Bloom filters introduces the possibility of having false positives, which will be resolved during a later stage of the context discovery process.

For example, let us assume a 6-bit Bloom filter with  $b$  equal to 2. If location information is hashed into  $\{2, 4\}$  and temperature information is mapped into  $\{3, 6\}$ , we obtain the filter shown in Fig. 1 with bit position 2, 3, 4, and 6 set to 1.

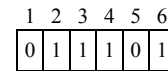


Fig. 1. A simple 6-bit Bloom filter.

Let us now discuss the querying of the example Bloom filter in Fig. 1. The filter definitely contains location and temperature information, because bit positions 2 and 4, respectively 3 and 6 have been set to 1. It definitely does not contain presence information, which is hashed into  $\{1, 4\}$ , because bit position 1 has not been set. It might be concluded that the filter contains humidity information  $\{2, 6\}$ , whereas it does not. This situation is termed false positive [4].

Our approach uses attenuated Bloom filters, each of which consists of  $d$  multiple layers of basic Bloom filters. The first layer of the filter contains the context type information for the current node, while the second layer contains the information about the nodes one hop away, and so on. In other words, a node can find the context type information  $i$  hops away in the  $i^{\text{th}}$  layer. By using attenuated Bloom filters consisting of multiple layers, context sources at more than one hop distance can be discovered, while avoiding saturation of the Bloom filter by attenuating (shifting out) bits set by sources further away.

An essential operation when using attenuated Bloom filters is context aggregation. It combines attenuated Bloom filters from neighboring nodes and a basic Bloom filter of locally available context information types to produce an attenuated Bloom filter that can be sent to neighbors. Fig. 2 presents the pseudo-code for the operation. Its arguments are the local filter ( $filter\_local$ ), received filters from neighbors ( $filter\_in[1,..]$ ,  $filter\_in[2,..]$ , ...,  $filter\_in[k,..]$ ), and their depth  $d$ . We number the layers of the filters from 0 to  $d-1$ . The first layer (layer 0) of the resulting filter ( $filter\_out$ ) contains the local basic Bloom filter ( $filter\_local$ ). Subsequent layers of the resulting filter are constructed by shifting the incoming filters one layer down, and applying an OR operation on corresponding bits. Note that the last layer of the incoming filters is not used. As a result, the first layer of  $filter\_out$  (layer 0) represents the local information, the second layer contains the information from direct neighbors, and the third layer covers the information two hops away which can be reached via direct neighbors, etc. Note that the bits of the local basic Bloom filter are set in all layers. These bits would probably be set in all layers anyway due to broadcasts that are looped back.

```

1 aggregation (filter_local, filter_in[1,..], filter_in[2,..], ..., filter_in[k,..],
  d) {
2
3 // The first layer of the outgoing filter is the local filter
4 filter_out[0] = filter_local;
5
6 // Subsequent layers of the outgoing filter are constructed by shifting
7 // the incoming filters one layer down, and applying an OR operation
8 // on corresponding bits.
9 for i = 1 to (d-1)
10 filter_out[i] = filter_local | filter_in[1,i-1] | filter_in[2,i-1] | ... |
  filter_in[k,i-1];
11 end;
12
13 return filter_out; // return the new filter
14 }

```

Fig. 2. Context aggregation pseudo code.

Fig. 3 exemplifies the context aggregation operation for a node with two neighbors.

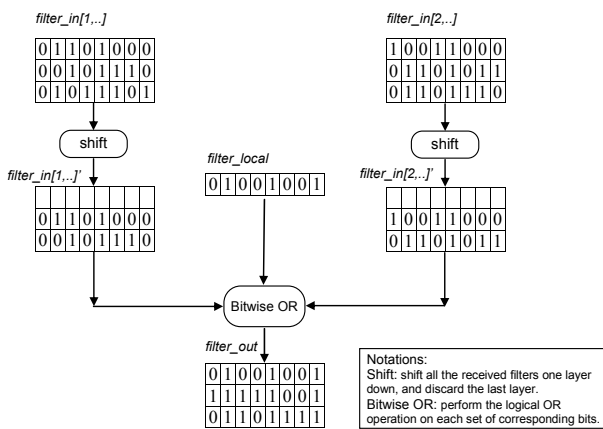


Fig. 3. An example of context aggregation.

### C. Context Discovery with Attenuated Bloom filters

Context discovery by using attenuated Bloom filters is a method combining proactive and reactive discovery mechanisms. Nodes broadcast attenuated Bloom filters to their neighbors. By receiving and storing their neighbors' attenuated Bloom filters, nodes can obtain an overview of available context types and their distribution. When a node wants to use a specific context type, it will generate a query, containing the Bloom code of the required context type. When this Bloom code matches with a layer in one of the received attenuated Bloom filters, a query message will be sent to the neighbor that broadcasted the Bloom filter. This neighbor will return a positive response if it supports the context type itself. It can also forward the query message to one (or more) of its neighbors, based on matches with the attenuated Bloom filters it has stored. As can be observed above, the proposed service discovery protocol consists of two major parts; context advertising and context querying. Both will be discussed in more detail below.

Context advertising is done as follows. When a node enters a new environment, it first sends a Bloom filter with its own context information types to its neighbors. When neighbors receive this filter, they will store it, aggregate it with their other stored filters and their local filter, as explained in Section II.B and broadcast the updated filter. The new node stores the incoming attenuated Bloom filters

separately for each neighbor, and also generates a new filter aggregating all the reply filters with the local one. This new filter will be sent to the neighbors in the next advertisement. The attenuated Bloom filters will be exchanged periodically, or when they change.

Context querying can be done in several ways, exploring possible positives either in parallel or sequentially. Fig. 4 presents pseudo-code for parallel querying. The function *receive\_query* is called when a query packet is received from a local application or from one of the neighboring nodes. Its arguments are the sender of this received packet (*received\_from*), and the query packet itself (*query\_packet*). The query packet contains at least an identification (*id*), a basic Bloom filter *bf* representing the queried context information type, and a hop count (*hop\_count*). When a query is generated by an application, the hop count value is set the same value with the depth of the filters *d*. First of all, the node receiving a query message will check whether this query has been received before or not. If so, the query packet will be dropped. Otherwise, the local basic Bloom filter will be checked for a match. If there is a local match, a response packet will be sent back to the sender of the query packet. Nodes will store path information, so that responses can be sent back along the same path. Next, the Bloom filter of the query packet will be compared with the stored attenuated Bloom filters from each of the neighbors. If there is a match at any of the *hop\_count-1* top layers in any of the stored filters, the query packet will be forwarded to the neighbor from whom the corresponding attenuated Bloom filter was received. The hop count of the query packet is decreased by 1. If no match is found, e.g., because of a false positive match in an earlier node, the query will be discarded.

In this way, query messages will be filtered out as early as possible, depending on the stored attenuated Bloom filters. The application in the originating node will set up a connection to the destination node based on the best response to the query. Note that a hop counter is used to restrict the query range. Queries will only be sent a limited number of hops away, based on the depth of the Bloom filters. If no response is received by the originating node within a time-out period, it understands that the required information is not available in the range of *d* hops.

In case of sequential querying, the originating node checks the stored attenuated Bloom filters on a layer-by-layer basis. If a match is found in a certain filter at layer *i*, a query message is sent to the corresponding neighbor with hop count *i+1*. Only if no response is received within a certain time-out period, lower layers of the stored attenuated Bloom filters will be checked, and possibly additional queries will be sent. Compared to parallel querying, sequential querying can reduce the amount of traffic sent at the cost of a potentially larger query time. This is a trade off between bandwidth and response time. The exact implementation of sequential querying is for further study. In this paper, we are considering parallel querying.

Note that recording the path of the query message requires maintaining (soft) state routing information in the nodes. This can put a burden on these nodes. Alternatively, the return path for responses can be stored in the query messages. This will result in increased transmission costs.

```

1  receive_query (received_from, query_packet) {
2
3      if is_in_id_cache(query_packet.id) == FALSE           // if query has not been received before
4          add_to_id_cache(query_packet.id);
5          if (query_packet.bf & filter_local) == query_packet.bf // if there is a local match
6              response_packet = make_response(query_packet); // send response packet to neighbor
7              send_response(received_from, response_packet); // this query packet was received from
8          end;
9          query_packet.hop_count--;
10         for k = 1 to number_of_neighbors                    // check all the neighbors
11             if k <> received_from
12                 for i = 0 to query_packet.hop_count - 1
13                     if (query_packet.bf & filter_in(k,i)) == query_packet.bf // there is match in link k, layer i
14                         add-to-route-cache(query_packet.id, received_from); // record path of query packet
15                         send_query(k, query_packet); // forward query to neighbor k
16                         break; // stop processing for neighbor k
17                 end;
18             end;
19         end;
20     end;
21 end;
22 }

```

Fig. 4. Context parallel querying pseudo code.

As a third alternative, our system can rely on an external routing protocol for returning responses to the originating node.

### III. PERFORMANCE MODELING

Queries due to false positives can potentially contribute significantly to the costs of context discovery. In order to reduce the number of unnecessary queries, we have to reduce the ratio of false positives. However, the minimum false positive ratio, which is by definition equal to 0, will result in large Bloom filters to be broadcasted. We believe there is a balance to be struck between a reasonable false positive ratio and the size of Bloom filters to achieve an optimal network cost. The optimal size of Bloom filters depends on their depth, the rates at which advertisements are sent and queries are generated, and the cardinality of the represented set, i.e., the number of services or context information type to be advertised. The optimum can be found by tuning the filter width and the number of hash functions properly. In this section, we will present a model to find the optimum balance between those parameters to minimize the network cost. Further, comparisons between context discovery with and without Bloom filters will be made.

#### A. Network structures and related vital parameters

To be able to analyze the performance of the proposed system, we model two typical network structures: grid structure and circle structure. A grid structure represents a simple and uniform network, while a circle structure shows an ideal network structure for a fully distributed mobile ad-hoc network.

Before discussing the network structures, we first introduce some related vital parameters. In general, we assume that each node has the same number of context information types,  $s$ . Further all context types are supposed to be unique, and taken out of an infinitely large set of possible context types. The same width,  $w$ , and depth,  $d$ , of attenuated Bloom filters, and the same  $b$  hash functions are used throughout the entire network. Queries are forwarded by at most  $d$  hops, based on the depth of Bloom filters. General notation is listed in the Table 1. Further notation is

introduced below.

TABLE I  
NOTATION

General		Bloom Filter	
Notation	Description	Notation	Description
$s$	number of services (context information types) per node	$w$	the width of the filter
$\mu$	advertisement (update) rate	$d$	the depth of the filter
$\lambda$	query rate	$b$	number of hash functions
$n$	network density (nodes/m <sup>2</sup> )		
$r$	communication range (m)		

#### 1) Grid structure

The first network structure we use for modeling the performance of the proposed system is a grid structure. In such a structure, each node has 4 direct neighbors within communication range. The structure is shown in Fig. 5a.

For deriving some useful properties of a grid structure, we consider a central node, and the nodes that can be reached in  $i$  hops. Let  $numofNewnodes_i$  denote the number of nodes that can be reached in  $i$  hops from the central node, but not in  $i-1$  hops. By definition, we set

$$numofNewnodes_0^g = 1. \quad (1)$$

Here the superscript  $g$  denotes the grid structure. For larger  $i$ , it can easily be seen from Fig. 5a. that  $4i$  nodes become reachable when increasing the maximum number of hops from the central node from  $i-1$  to  $i$ . So,

$$numofNewnodes_i^g = 4 \cdot i \quad (i > 0). \quad (2)$$

This value increases linearly with the distance (the number of hops). The total number of nodes reachable in at most  $i$  hops, including the central node can be easily found by summation:

$$Totalnumofnodes_i^g = \sum_{j=0}^i numofNewnodes_j^g = 1 + 2i(i+1). \quad (3)$$

#### 2) Circle structure

Ideally, the two dimensional radio coverage of a mobile node is a circle. Therefore, a circle structured network is

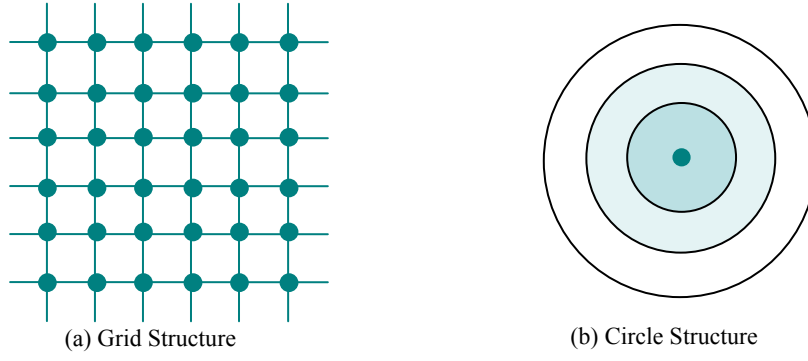


Fig. 5. Network structures: (a) grid structure; (b) circle structure.

also assumed in our modeling. The communication range of nodes will be generalized as a set of concentric circles, as shown in Fig. 5b. Nodes located in the inner circle are the ones reachable within one hop from the center node. Nodes in one ring outside the inner circle are the ones reachable in two hops, and so on.

We assume that the communication range of a node is denoted as  $r$ . The density of the network is  $n$  nodes/m<sup>2</sup> on average. Again, by definition,

$$\text{numofNewnodes}_0^c = 1. \quad (4)$$

Here the superscript  $c$  denotes the circle structure. Below, the superscript will be omitted to obtain generalized results. In a circle with radius  $r \cdot i$  around the central node, an average number of nodes of  $n\pi(r \cdot i)^2$  can be found. We take the simplifying assumption that these nodes are exactly the ones reachable in at most  $i$  hops. Note that this assumption is more accurate for dense networks and relatively smaller value of  $i$ . So, we get for the total number of nodes, reachable in at most  $i$  hops, including the center node,

$$\text{Totalnumofnodes}_i^c = 1 + n\pi r^2 i^2. \quad (5)$$

From this formula, it is easy to derive the number of nodes that are reachable in  $i$  hops, but not in  $i-1$ :

$$\begin{aligned} \text{numofNewnodes}_i^c &= n\pi r^2 (i^2 - (i-1)^2) \quad (i > 0). \\ &= (2i-1)n\pi r^2 \end{aligned} \quad (6)$$

Compared to the grid structure, this circle structure has the advantage that we can vary the node density, and as a result the average number of neighbors of a node. To ease comparison, we are interested in the node density at which the circle structure most closely resembles the grid structure. Therefore, we are looking for the density where  $\text{Totalnumofnodes}_i^g$  equals  $\text{Totalnumofnodes}_i^c$ . So we have

$$1 + 2i(i+1) = 1 + n\pi r^2 i^2 \Rightarrow \quad (7)$$

$$n\pi r^2 = \frac{2i(i+1)}{i^2} = 2 + \frac{1}{i} \quad (i > 0). \quad (8)$$

If  $i$  is large, the circle structure corresponds with the grid structure for

$$n\pi r^2 \approx 2. \quad (9)$$

### B. Cost Functions

In our performance analysis, we aim at an expression for the total costs of transmissions in bits per second per node.

We define two types of cost in the network: cost for successful querying and overhead cost. Cost for successful querying is caused by a query with positive results. Overhead cost is induced by the advertisements and false positive queries. The total cost for a node is defined as the sum of the successful querying cost ( $C_{\text{successfulquerying}}$ ) and overhead cost ( $C_{\text{overhead}}$ ). Overhead cost is the sum of advertisement cost ( $\text{adcost}$ ) and false positive cost ( $\text{fpcost}$ ). Note that there is no false negative in Bloom filters.

$$\text{cost} = C_{\text{successfulquerying}} + C_{\text{overhead}}, \quad (10)$$

$$C_{\text{overhead}} = \text{adcost} + \text{fpcost}. \quad (11)$$

In our analysis, we will focus on the overhead cost.

We assume that advertisements are broadcasted periodically at a constant rate. Therefore the advertisement cost can be defined as:

$$\text{adcost} = \mu \cdot \text{adpack}, \quad (12)$$

where  $\mu$  is the advertisement (update) rate, and  $\text{adpack}$  is the advertisement packet size.

The false positive cost,  $\text{fpcost}$ , represents the transmission cost for false positive queries incurred by a query initiated in the node under consideration. Transmission of such query messages can take place on all links up to  $d$  hops away from the node under consideration. Thus, we can denote the false positive cost as

$$\text{fpcost} = \lambda \cdot \sum_{i=1}^d \text{cost}_{\text{fp},i}. \quad (13)$$

We assume queries are performed at a certain rate  $\lambda$ , i.e.,  $\lambda$  queries will be initiated per second per node.  $\text{cost}_{\text{fp},i}$  denotes the total cost of all false positive queries transmitted to nodes  $i$  hops away from the node under consideration by nodes  $i-1$  hops away.

In order to obtain this false positive query cost to the  $i^{\text{th}}$  hop, we have to count the possible number of query transmissions sent by nodes  $i-1$  hops away to their neighbors,  $\text{numofTransmission}_{\text{fp},i}$ . Such a transmission is indeed done, with a packet size  $q\text{pack}$ , if the attenuated Bloom filter received from the intended receiver of the query (the node at  $i$  hops) by the node at  $i-1$  hops gives a false positive in layer  $d-i$ . This false positive will occur with probability  $P_{\text{fp},d-i}$ , where  $P_{\text{fp},j}$  is defined as the probability of a false positive occurring in layer  $j$ . Note that a false positive at any of the layers  $j < d-i$  automatically implies a false positive at layer  $d-i$ , since bits set in layer 0 are duplicated to all layers (see Fig. 2). So, false positives at

layers  $j < d-i$  do not have to be accounted for. A false positive in a layer  $j > d-i$  does not result in a query to be transmitted, because the final destination of that query would be more than  $d$  hops away from the node originating the query. Finally, note that the query to be sent to the node at  $i$  hops will reach the node at  $i-1$  hops for certain, since the false positive at layer  $d-i$  (as received by the node at  $i-1$ ) will also appear at the query originating node at layer  $i-1$ . The resulting false positive query cost to the  $i^{\text{th}}$  hop can be given as:

$$\text{cost}_{fp,i} = P_{fp,d-i} \cdot \text{numofTransmission}_{fp,i} \cdot \text{qpack}. \quad (14)$$

For determining  $\text{numofTransmission}_{fp,i}$ , we use the results from Section 3.1, omitting the superscripts for grid or circle structure to generalize the results. The possible number of transmissions sent by nodes  $i-1$  hops away to their neighbors,  $\text{numofTransmission}_{fp,i}$ , equals the number of new nodes reached in the  $(i-1)^{\text{th}}$  hop times the number of neighbors each new node has to forward to. For  $i=1$ , this is

$$\text{numofTransmission}_{fp,1} = \text{numofNewnodes}_1. \quad (15)$$

For larger  $i$ , we have to take into account that one of the neighbors of the node at  $i-1$  hops is the neighbor the query packet was received from. So the query will be forwarded to at most  $\text{numofNewnodes}_{i-1}$  neighbors:

$$\text{numofTransmission}_{fp,i} = \text{numofNewnodes}_{i-1} \cdot (\text{numofNewnodes}_i - 1) \quad (i > 1). \quad (16)$$

If one of the nodes receiving the query has received it before, it will discard it (see Fig. 4). That is why in the above formula only new nodes at  $i-1$  hops are taken into account.

Let us finally determine  $P_{fp,j}$ , the probability of generating a false positive in layer  $j$  as a result of a query. In this paper, we assume that the hash functions we choose are perfectly random. Let the random variable  $m_j$  denote the number of bits actually set in the layer  $j$ . Clearly,  $1 \leq m_j \leq \min(b \cdot x_j, w)$ , where  $x_j$  denotes the number of services (or context information types) represented in layer  $j$ . We can obtain  $P_{fp,j}$  by conditioning on the number of bits set:

$$P_{fp,j} = \sum_{k=1}^{\min(b \cdot x_j, w)} P\{\text{false positive} | m_j = k\} \cdot P\{m_j = k\}. \quad (17)$$

The false positive probability if the number of bits set is known to be  $k$  can be expressed as:

$$P\{\text{false positive} | m_j = k\} = \left(\frac{k}{w}\right)^b. \quad (18)$$

The probability that exactly  $k$  bits are set in the Bloom filter, given that a bit is set  $b \cdot x_j$  times, is the quotient of

- the product of
  - the number of ways of partitioning a set of  $b \cdot x_j$  elements into  $k$  non-empty subsets;
  - the number of permutations of a subset of  $k$  elements out of  $w$  with replacement;
- the number of ways to choose  $b \cdot x_j$  elements out of  $w$  with replacements.

This can be denoted as:

$$P\{m_j = k\} = \frac{S(b \cdot x_j, k) \cdot \frac{w!}{(w-k)!}}{w^{b \cdot x_j}}, \quad (19)$$

where

$$S(b \cdot x_j, k) = \frac{1}{k!} \cdot \sum_{l=1}^k (-1)^{k-l} \cdot \binom{k}{l} \cdot l^{b \cdot x_j}, \quad (20)$$

is called a Stirling number of the 2<sup>nd</sup> kind.

This results in the following equation for the false positive probability in layer  $j$ :

$$P_{fp,j} = \sum_{k=1}^{\min(b \cdot x_j, w)} \left(\frac{k}{w}\right)^b \cdot \frac{S(b \cdot x_j, k) \cdot \frac{w!}{(w-k)!}}{w^{b \cdot x_j}} \quad (21)$$

$$= \frac{1}{w^{b \cdot (x_j+1)}} \cdot \sum_{k=1}^{\min(b \cdot x_j, w)} k^b \cdot S(b \cdot x_j, k) \cdot \frac{w!}{(w-k)!}.$$

The above formula is computationally quite complex, especially for large values of  $b$  and  $w$ . Therefore, we seek to approximate (21) by a simpler formula. If the probability density function of  $m$  is relatively highly concentrated around its mean, we can approximate (17) by taking the false positive probability at the expected value of  $m_j$ , instead of conditioning on  $m_j$ . Experiments show that this is the case if  $b \cdot x_j$  is small compared to  $w$ :

$$P_{fp,j} \approx P\{\text{false positive} | m_j = E\{m_j\}\}. \quad (22)$$

The expected number of bits set at layer  $j$  of the Bloom filter can be expressed as:

$$E\{m_j\} = \left[1 - \left(1 - \frac{1}{w}\right)^{b \cdot x_j}\right] \cdot w. \quad (23)$$

Finally, since  $(1-1/w)^{b \cdot x_j}$  can be approximated by  $e^{-b \cdot x_j / w}$  when  $b \cdot x_j$  is small compared to  $w$  [4], we can substitute (18) and further approximate the false positive probability as follows:

$$P_{fp,j} \approx \left[1 - \left(1 - \frac{1}{w}\right)^{b \cdot x_j}\right]^b \approx \left(1 - e^{-b \cdot x_j / w}\right)^b. \quad (24)$$

Note that the first approximate step in (24) is given as if it is exact in some earlier publications (e.g., [2], [4], [10], [12], and [15]). Experiments reveal that for optimized parameter values the outcome of (24) is always within 0.5% of the outcome of (21).

Formula (24) shows that the false positive probability depends on the width ( $w$ ) and number of hash functions ( $b$ ) of the Bloom filters, and the number of services contained inside the filter. We define the number of services in the  $j^{\text{th}}$  hop ( $j \geq 0$ ) for the grid and circle structures as:

$$x_j = s \cdot \text{Totalnumofnodes}_j. \quad (25)$$

The assumption is that all services that are reachable in  $j$  or fewer hops are represented at the  $j^{\text{th}}$  layer. A service that is represented at a certain layer is also assumed to be represented at all layers below. This holds if the local filter of a node is duplicated to all layers of its advertised attenuated Bloom filter, as is done in the pseudo code of

Fig. 2. Even if duplication would not be applied, this will often be the case, because a service at  $i$  hops away can probably also be reached via an alternative path of  $j > i$  hops length.

To complete the performance model, we have to specify the size of the packets for advertising and querying. We assume that the context discovery protocol using Bloom filters is running on top of UDP. For both advertisements and queries, besides the header of the Bloom filters protocol advertisement (AD) and query (Q) messages, the headers of the UDP, IP, and MAC layer will be attached. The advertisements and queries packet size are defined as follows:

$$\begin{aligned} adpack = & \text{header}_{MAC} + \text{header}_{IP} + \text{header}_{UDP} \\ & + \text{header}_{AD} + w \times d \end{aligned} \quad (26)$$

$$qpack = \text{header}_{MAC} + \text{header}_{IP} + \text{header}_{UDP} + \text{header}_Q + w \quad (27)$$

Note that the size of the advertisement packet depends on both  $w$  and  $d$ , as the complete attenuated Bloom filter has to be transmitted. The possibility of applying compression to the filter is beyond the scope of this paper. The size of the query packet only depends on  $w$ , since only a basic Bloom filter representing the query is to be transmitted. Note that instead of a basic Bloom filter, the query string could also be sent in the query message.

### C. Two Extreme Cases

To evaluate the performance of context discovery using attenuated Bloom filters, we compare it with two alternative discovery solutions: complete advertisement and no advertisement.

Complete advertisement floods all network nodes within  $d$  hops with a complete description of all context information. Nodes have the complete map of the network, which indicates how nodes can send queries directly to the destination. It is a proactive protocol. The advertisement cost is the main concern in this situation. We assume that each context information type can be presented in  $c$  bits. We assume each node up to  $d-1$  hops away to broadcast the advertisement, so that we have:

$$\begin{aligned} cost_{compl\_ad} = & \mu \cdot Totalnumofnodes_{d-1} \\ & \cdot (\text{header}_{MAC} + \text{header}_{IP} + \text{header}_{UDP} + \text{header}_{AD} + s \cdot c). \end{aligned} \quad (28)$$

In the no advertisement case, nodes do not advertise context information types. When a query comes, nodes forward it to all the neighbors. It is a reactive protocol. Nodes do not have any idea about the network. The queries

are spreading around the whole network, up to  $d$  hops from the originator. There is no way to stop forwarding queries, even though the query node has already received an answer. The cost for querying is counted as the cost for sending queries uni-directionally to all nodes in the network. All nodes up to  $d$  hops away will receive such a query, apart from the node where the query was originated. So, the number of transmissions also equals the total number of nodes within  $d$  hops minus 1:

$$\begin{aligned} cost_{no\_ad} = & \lambda \cdot (Totalnumofnodes_d - 1) \\ & \cdot (\text{header}_{MAC} + \text{header}_{IP} + \text{header}_{UDP} + \text{header}_Q + c). \end{aligned} \quad (29)$$

## IV. EXPERIMENTAL RESULTS

The model described above has been implemented in Matlab 6.5. Using the model, two basic sets of experiments (experiment 1 and 2) have been done for both network structures; three extra experiments (experiment 3, 4, and 5) have been implemented for the circle structure. Next to the model presented in this paper, a detailed implementation of the proposed system has been made in the simulation package OPNET [7] and [16]. The results for the grid model have been checked against simulation results. They turn out to be within the 90% confidence interval, which was smaller than 1% of the mean value. Note that the model is built based on the approximation formula (24).

For all experiments, we focus on the overhead of the proposed system and its alternatives. Therefore, we assume that the services that are queried for are not present in the network, i.e.,  $C_{successfulquerying} = 0$ .

The following tables and figures show the results of the experiments. Experiment 1 is used to achieve the optimal cost by choosing the proper width,  $w$ , of the Bloom filter and the number of hash functions,  $b$ , with given depth,  $d$ , of the filter, query rate,  $\lambda$ , and number of services per node,  $s$ . Experiment 2 shows the influence of the query rate,  $\lambda$ , on the network cost for given  $d$  and  $s$ . The influence of the query range,  $d$ , is evaluated in experiment 3. In experiment 4, we show the impact of density of services,  $s$ , in the network. The final experiment presents performance for varying network density,  $n$ , in the circle structure.

In all the experiments below, we assume that the advertisement rate  $\mu = 0.1$  and that each context information type can be represented in 32 bits, i.e.,  $c = 32$  bits. The sizes of headers are assumed as follows [11]:  $header_{MAC} = 160$  bits;  $header_{IP} = 320$  bits (assuming the use of IPv6);  $header_{UDP} = 64$  bits;  $header_{AD} = 32$  bits;  $header_Q = 192$  bits.

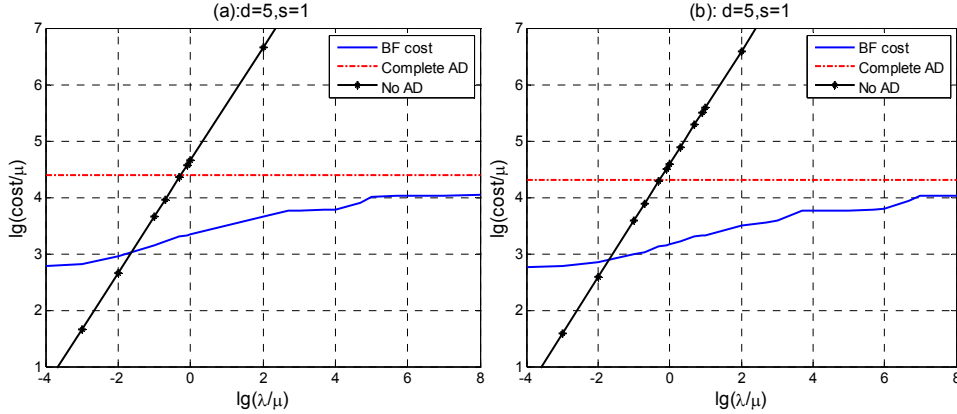
TABLE II  
OPTIMAL BF COST COMPARED WITH COMPLETE AND NO ADVERTISEMENT (GRID STRUCTURE)

$d$	$w$ (bit)	$b$	BF cost (bit/s)	Complete Advertisement (bit/s)	No Advertisement (bit/s)	Maximum number of services in BF
3	96	5	97	790	1843	13
5	256	5	218	2493	4608	41
7	448	5	477	5168	8602	85
10	896	5	1277	11005	16896	181

TABLE III

OPTIMAL BF COST COMPARED WITH COMPLETE AND NO ADVERTISEMENT (CIRCLE STRUCTURE)

$d$	$w$ (bit)	$b$	BF cost (bit/s)	Complete Advertisement (bit/s)	No Advertisement (bit/s)	Maximum number of services in BF
3	64	5	79	547	1382	9
5	128	4	145	2006	3840	33
7	224	4	296	4438	7526	73
10	480	4	779	9910	15360	163

Fig. 6. Performance results for varying  $\lambda/\mu$  for (a) grid structure and (b) circle structure.

## A. Basic Experiments

### 1) Experiment 1

In this experiment, we assume a query rate of  $\lambda = 0.1$ , and a number of services per node of  $s = 1$ . For each given value of the filter depth,  $d$ , the experiment result shows that there exists a certain value of  $w$  and  $b$  which leads to the minimum network cost. The results are shown in Table 2 and Table 3 for the grid and circle network structures respectively. The results are also compared with the complete and no advertisement cases under similar situations.

As we see from Table 2 and Table 3, for each depth of the filter, the proper width and number of hash functions leads to a minimum network cost which is much lower than for the cases of “complete advertisement” and “no advertisement”. The difference becomes larger as the query range  $d$  increases. The last column shows the maximum number of services that can be represented in one Bloom filter for the corresponding depth of Bloom filter ( $x_d$ ). A significant difference between results for the grid structure and the circle structure can be observed. This is attributed to the difference in the number of nodes within  $i$  hops, for small  $i$ .

### 2) Experiment 2

Using Bloom filters, we can reduce the packet size by using simple and efficient coding. However, false positives also create redundant traffic. This trade-off heavily depends on the relation between the update frequency  $\mu$ , and the query rate  $\lambda$ . We expect that there exists a (high) value for  $\lambda$  at which the traffic generated due to false positives is more than the benefit of using Bloom filters. In contrast, if there are only few queries in the network, it does not pay to broadcast the context information to the entire network. A no advertisement protocol can perform better in this case. This experiment is going to discuss the suitable range of

using Bloom filters for context discovery to achieve the minimum network cost. The comparisons are done for  $d$  is a value from 3 to 10.

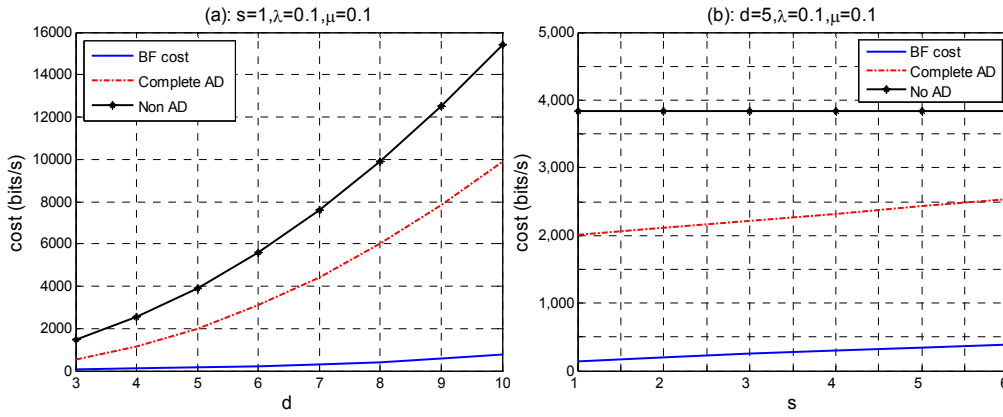
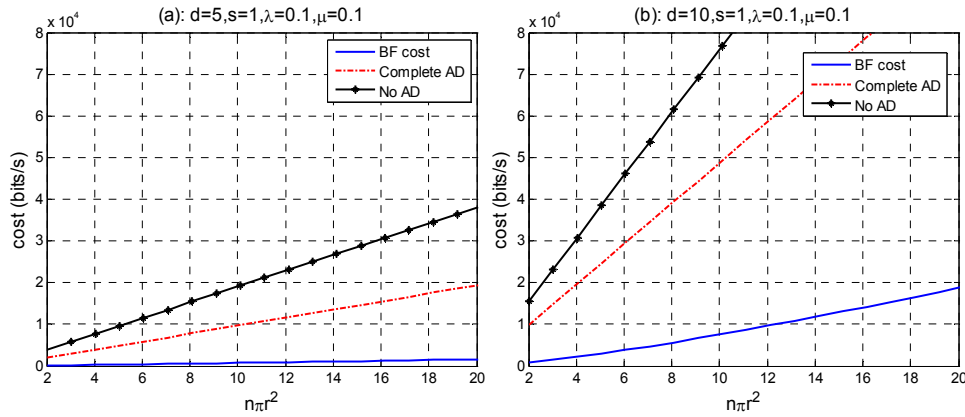
We set  $\mu$  as a reference, and change the value of  $\lambda$ . Here we talk about  $\lambda/\mu$ . The experiments show that the favorable range of  $\lambda/\mu$  decreases when the depth of the filter,  $d$ , increases. For both the grid and the circle structure, when each node has only one service ( $s = 1$ ), the Bloom filter context discovery algorithm performs better than the no advertisement algorithm when  $\lambda$  is at least 0.1 times  $\mu$ , i.e., if queries are generated at least once per 10 advertisement periods. Further, the Bloom filter algorithm performs better than the complete advertisement algorithm even if  $\lambda$  is  $10^8$  times  $\mu$ , when  $d$  is between 3 and 10. However, the upper bound of  $\lambda/\mu$  for better performance of Bloom filter discovery mechanism decreases, while  $d$  increases. Fig. 6a and 6b show the situation when  $d = 5$  for both network structures. Note that both axes have a logarithmic scale.

When each node has 4 services ( $s = 4$ ), the network requires larger Bloom filters to contain more information. The results show that for  $d = 5$ , the proper range of  $\lambda/\mu$  is (0.1, 500000); for  $d = 10$ , the proper range of  $\lambda/\mu$  is (0.1, 500). These results are not shown in any of the figures.

We found that in practical situations the Bloom filter algorithm has a better performance. If the query rate and the advertisement rate are about equal ( $\lg(\lambda/\mu)=0$ ), the context discovery system based on attenuated Bloom filters has an order of magnitude less network costs than alternative systems.

## B. Extensive Experiments

Because circle structures resemble the reality more closely, the following three experiments focus on such a structure. The sensitivity of our Bloom filter context discovery algorithm to several parameters is observed in this section.

Fig. 7. Impact of (a) Bloom filter depth,  $d$ , and (b) context type density,  $s$ .Fig. 8. Impact of network density for (a)  $d=5$  and (b)  $d=10$ .

### 1) Experiment 3

With a larger search range (larger  $d$ ), there are more context information types available within the range. On the other hand, a larger  $d$  also leads to larger size of the attenuated Bloom filter. In this set of experiments, we would like to see the impact of  $d$ .

We vary the depth of the Bloom filter,  $d$ , from 3 to 10, and compare the performance with different values of  $s$  and  $\lambda$  (fixed  $\mu = 0.1$ ). The results show that, in general, the Bloom filter algorithm has better performance than complete and no advertisement algorithms. There is a limit to the number of services and the query rate for which the algorithm has the best performance. When exceeding that limit, the performance of Bloom filter becomes worse. When the number of services within range (this depends on both  $d$  and  $s$ ) and the query rate is quite high, the cost of using the Bloom filter algorithm increases significantly. For instance, this happens when  $s = 4$ ,  $\lambda = 20$ , and  $d > 9$ . Fig. 7a shows the results when  $s = 1$  and  $\lambda = 0.1$ .

### 2) Experiment 4

From the experiment above, we find that the number of services per node also has some influence on the network cost. In this set of experiments, we would like to investigate it in detail. We do this for fixed  $d$  and  $\lambda$ . The results show that there is some influence from  $s$ , but not much. When  $s$  is increasing from 1 to 6, i.e., the number of context sources within range increases from 51 to 306, using Bloom filters still has by far the best result among three alternative algorithms. The network cost of using a Bloom filter increases only a little bit faster than the complete advertisement algorithm. We can expect the Bloom filter

algorithm to perform worse when  $s$  is really large, which will seldom happen in reality (for given  $d$  and  $\lambda$ ). Fig. 7b shows the results for three alternative algorithms when  $d = 5$  and  $\lambda = 0.1$ .

### 3) Experiment 5

In the previous experiments, we worked with a network density  $n\pi r^2 = 2$ , i.e., an average node has 2 neighbors. This made the circle structure correspond to the grid structure w.r.t. the total number of nodes within  $i$  hops, for large  $i$ . The influence of the network density is also very interesting, and in the model based on a circle structure we can vary this parameter. We observe the influence of network density in this experiment. The number of nodes within communication range is varied from 2 to 20, which implies that  $n\pi r^2 = 2$  to 20. The other parameters are set as follows:  $s = 1$ ;  $\lambda = 0.1$ ;  $\mu = 0.1$ .

The results show that with the increase of the number of direct neighbors in reach, the cost of using our Bloom filter method is increasing. However, the cost of using Bloom filters is still much less than using the other alternatives. Further, the increase is not as fast as for complete advertisement and no advertisement methods. The influence of the network density is relatively small compared to the influence of the ratio of query and advertisement rate. Fig. 8 illustrates the impact of network density when  $d$  equals 5 (a) and 10 (b).

## V. CONCLUSIONS AND FUTURE WORK

We have introduced a novel approach for performing context discovery in ad-hoc networks based on the use of attenuated Bloom filters. Attenuated Bloom filters are used

to advertise the availability of context information multiple hops away, and to guide queries to discover it. In order to investigate the performance of this approach, a model has been developed. The use of attenuated Bloom filters for advertising available context types in ad-hoc networks is very promising. We observed the performance of the model in two different network structures: a simple grid network and a circle network which represents a fully distributed ad-hoc network. Results show the combined cost of advertising and doing unsuccessful queries due to false positives. The same conclusion can be drawn from both structures, namely that there exists an optimum size of Bloom filters to achieve optimal network cost. The performance of Bloom filters highly depends on the ratio of query and advertisement rates, and on the query range of nodes. The density of context information sources and the node density also have some influence. Especially, in a fully distributed ad-hoc network in practical situations, this approach requires significantly less (up to an order of magnitude) traffic load than advertising a full map of all available context types, or broadcasting queries when no advertisements are used. As such, it is a very promising compromise between these two extremes.

Ongoing and future research includes further refinement of the ideas presented in this paper. Especially, the impact of node mobility on the network cost and query success ratio is a very important topic for further study. This is not addressed by the current model. Meanwhile, we are developing the idea further, by specifying a protocol, and testing this in a detailed, discrete event simulator and in a prototype. Security issues are also subject to future research. Finally, an interesting idea to explore is to use the broadcasting of attenuated Bloom filters to execute directed route requests (instead of undirected broadcasts) for ad-hoc routing protocols such as AODV. Such an approach would allow ad-hoc nodes to efficiently establish routes only to other nodes with relevant context information, rather than establishing multiple routes first, and then finding out where the relevant context information is.

#### ACKNOWLEDGMENT

We are grateful to Patrick Goering, who implemented the OPNET simulator model. We also thank Boudewijn Haverkort for his helpful comments.

#### REFERENCES

- [1] K. Arabshian and H. Schulzrinne, "GloServ: global service discovery architecture", in *Proc. 1<sup>st</sup> Annual International Conference on Mobile and Ubiquitous Systems, MobiQuitous 2004*, August 2004, Cambridge, USA.
- [2] B. H. Bloom, "Space/Time trade-offs in hash coding with allowable errors", *Communications of the ACM*, vol. 13, no. 7, July 1970, pp. 422 – 426.
- [3] Bluetooth Consortium, "Specification of Bluetooth system Core version 2.0: part C, service discovery protocol (SDP)", November 2004.

- [4] A. Broder and M. Mitzenmacher, "Network applications of Bloom filters: a survey", *Internet Math*, vol. 1, no. 4, 2003, pp. 485-509.
- [5] D. Chakraborty, A. Joshi, T. Finin, and Y. Yesha, "GSD: a novel group-based service discovery protocol for MANETs", in *Proc. 4<sup>th</sup> IEEE MWCN 2002*, Stockholm, Sweden, September 2002.
- [6] S. E. Czerwinski, B. Y. Zhao, T. D. Hodes, A. D. Joseph, and R. H. Katz, "An architecture for a secure service discovery service", in *Proc. 5<sup>th</sup> ACM/IEEE MobiCom*, Seattle, USA, August 1999.
- [7] P. T. H. Goering and G. J. Heijenk, "Service discovery using Bloom filters", in *Proc. 12<sup>th</sup> Annual Conference of the Advanced School for Computing and Imaging*, Lommel, Belgium, June 2006.
- [8] E. Guttman, C. Perkins, J. Veizades, and M. Day, "Service location protocol version 2", *RFC 2608*, IETF, June 1999.
- [9] J. Hoebeke, I. Moerman, Bart Dhoedt, and Piet Demeester, "Analysis of decentralized resource and service discovery mechanisms in wireless multi-hop networks", in *Proc. 3<sup>rd</sup> WWIC*, Xanthi, Greece, May 2005.
- [10] Y. Huang and W. Lee, "Hotspot-based traceback for mobile ad-hoc networks", in *Proc. 4<sup>th</sup> ACM Workshop on Wireless Security*, Cologne, Germany, September 2005.
- [11] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach featuring the internet*, Addison Wesley Longman Inc, 2001.
- [12] F. Liu and G. Heijenk, "Context discovery using attenuated Bloom filters in ad-hoc networks", in *Proc. 4<sup>th</sup> WWIC*, Bern, Switzerland, May 2006.
- [13] J. Liu, F. Sailhan, D. Sacchetti, and V. Issarny, "Group management for mobile ad-hoc networks: design, implementation and experiment", in *Proc. 6<sup>th</sup> IEEE International Conference on Mobile Data Management*, Ayia Napa, Cyprus, May 2005.
- [14] R. Marin-Perianu, P. H. Hartel, and J. Scholten, "A classification of service discovery protocols", Centre for Telematics and Information Technology, University of Twente, The Netherlands, Technical report nr. TR-CTIT-05-25, June 2005.
- [15] P. Mutaf and C. Castelluccia, "Compact neighbor discovery", in *Proc. IEEE INFOCOM 2005*, Miami, March 2005.
- [16] OPNET modeler software, available: <http://www.opnet.com/products/modeler>.
- [17] E. Papapetrou, E. Pitoura, and K. Lillis, "Speeding –up cache lookups in wireless ad-hoc routing using Bloom filters", in *Proc. IEEE PIMRC 2005*, Berlin, Germany, September 2005.
- [18] Sun Microsystems, "Jini architecture specification version 2.1", November 2005, [http://java.sun.com/products/jini/2\\_1\\_index.html](http://java.sun.com/products/jini/2_1_index.html).
- [19] Wikipedia, "Context" and "Service" — wikipedia, the free encyclopedia, <http://en.wikipedia.org/>, accessed in March 2006.



**Fei Liu** is a Ph.D candidate at the University of Twente, the Netherlands. She received her B.Sc. degree in Computer Science from Jiangsu Institute of Petrochemical Technology, China, in 2002, and her M.Sc. degree in Telematics from University of Twente, the Netherlands, in 2004. Her research interests include mobile and wireless networks.



**Geert Heijenk** received his M.Sc. in Computer Science from University of Twente, the Netherlands, in 1988. He has worked as a research staff member at the same university and received his Ph.D. in Telecommunications in 1995. He has also held a part-time position as researcher at KPN research, the Netherlands, from 1989 until 1991. From 1995 until 2003, he was with Ericsson EuroLab Netherlands, first as a senior strategic engineer, and from 1999 as a research department manager. From 1998 until 2003 he was also a part-time senior researcher at the University of Twente. Currently, he is a full-time associate professor at the same university. Geert Heijenk has been a visiting researcher at University of Pennsylvania, Philadelphia and a visiting associate professor at University of California, Irvine. He is a senior member of the IEEE. His research interests include mobile-, wireless-, and ad-hoc networks, resource management, and quality of service.