Hindawi Publishing Corporation
Journal of Computer Networks and Communications
Volume 2014, Article ID 450194, 13 pages
http://dx.doi.org/10.1155/2014/450194

Research Article

QoS-Based Web Service Discovery in Mobile Ad Hoc Networks Using Swarm Strategies

Filomena de Santis and Delfina Malandrino

Department of Computer Science, University of Salerno, Via Giovanni Paolo II 132, 84084 Fisciano, Italy

Correspondence should be addressed to Delfina Malandrino; delmal@dia.unisa.it

Received 29 May 2014; Accepted 12 October 2014; Published 10 November 2014

Academic Editor: Rui Zhang

Mobile ad hoc networks are noncentralised, multihop, wireless networks that lack a common infrastructure and hence require self-organisation. Their infrastructureless and dynamic nature entails the implementation of a new set of networking technologies in order to provide efficient end-to-end communication according to the principles of the standard TCP/IP suite. Routing, IP address autoconfiguration and Web service discovery are among the most challenging tasks in the ad hoc network domain. Swarm intelligence is a relatively new approach to problem solving that takes inspiration from the social behaviours of insects, such as ants and bees. Self-organization, decentralization, adaptivity, robustness, and scalability make swarm intelligence a successful design paradigm for the above-mentioned problems. In this paper we propose *BeeAdHocServiceDiscovery*, a new service discovery algorithm based on the bee metaphor, which also takes into account quality metrics estimates. The protocol has been specifically designed to work in mobile ad hoc network scenarios operating with *Beeadhoc*, a well-known routing algorithm inspired by nature. We present both the protocol strategy and the formal evaluation of the discovery overhead and route optimality metrics showing that *BeeAdHocServiceDiscovery* guarantees valuable performances even in large scale ad hoc wireless networks. Eventually, future research suggestions are sketched.

1. Introduction

A mobile ad hoc network (MANET) is a set of mobile nodes that communicate over radio and operate without the benefit of any infrastructure; nodes continuously enter and leave the network according to their mobility needs. The limited transmission range of wireless interfaces makes the source-destination communication multihop. Nodes accomplish the functionality of hosts, as well as that of routers forwarding packets for other nodes. MANETs are very flexible and suitable for several situations and applications since they allow establishing temporary connections without preinstalled resources. Remarkable uses of mobile ad hoc networks are in calamity and military scenario; with the increasing diffusion of radio technologies, many multimedia applications also take advantages from running over them. MANETs suffer from a variety of questions: the routing and the IP (internet protocol) address autoconfiguration problems are among the most challenging ones. Many different approaches dealing with them do exist, even though there are no algorithms that fit in all cases [1, 2]. Moreover, the number and variety of services provided by MANETs are constantly increasing with the expansion of their applications; thus, services offered by single nodes are accordingly spreading as well as the need of sharing useful facilities among nodes. To get benefit from such a practice a device must be able to locate the service provider in the network and to invoke the service itself. Since different nodes providing different services may enter and leave the network at any time, many research efforts aim at improving MANETs usability by means of an efficient and timely service management and discovery, that is, to say, by means of a suitable service discovery protocol (SDP) [3]. In this paper, we present *BeeAdHocServiceDiscovery*, a novel swarm intelligence SDP based on *BeeAdHoc*, a well-known routing algorithm for MANET derived from the bee colony optimization metaheuristic [4–7]. Swarm intelligence (SI) is a well-known distributed paradigm for the solution of hard problems taking insight from biological scenario such

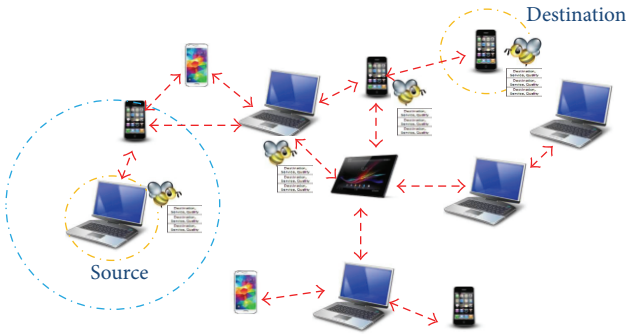

Figure 1: Mobile ad hoc network with routing via bee colonies.

as colonies of ants, bees and termites, schools of fish, and flocks of birds. The most interesting property of SI is the involvement of multiple individuals that interact with each other and the environment, exhibit a collective intelligent behavior, and are able to solve complex problems. Many applications, mainly in the contexts of computer networks, distributed computing and robotics exploit algorithm designs using SI. The basic idea behind this paradigm is that many tasks can be more efficiently completed by using multiple simple autonomous agents instead of a single sophisticated one. Regardless of the improvement in performance, such systems are usually much more adaptive, scalable, and robust than those based on a single, highly capable, agent. An artificial swarm can be generally defined as a decentralized group of autonomous agents having limited capabilities. Due to the adaptive and dynamic nature of MANETs, the swarm intelligence approach is considered a successful design paradigm to solve the routing, the IP address autoconfiguration and the service discovery problems [4, 8, 9]. The remainder of this paper is organized as follows. Section 2 reviews the basics of the bee colony optimization metaheuristic and routing and autoconfiguration algorithms derived from it. Section 3 introduces the fundamentals of service discovery as well as a short review of the literature about it. Section 4 describes the new proposed algorithm and its computational complexity. Eventually, Section 5 sketches conclusions and ideas for future works.

2. The Swarm Paradigm

2.1. BeeAdHoc Routing Algorithm. Bee colonies (*Apis mellifera*) and the majority of ant colonies (*Argentine ant* and *Linepithema humile*) show similar structural characteristics, such as the presence of a population of minimalist social individuals and must face analogous problems for what is concerned with distributed foraging, nest building, and maintenance. A honeybee colony consists of morphologically uniform individuals with different temporary specializations. The benefit of such an organization is an increased flexibility to adapt to the changing environments. Thousands of worker bees perform all the maintenance and management jobs in the hive. There are two types of worker bees, namely, scouts and foragers. The scouts start from the hive in search of a food source randomly keeping on this exploration process until they are tired. When they return to the hive, they convey to the foragers information about the odor of the food, its direction, and the distance with respect to the hive by performing dances. A round dance indicates that the food source is nearby whereas a waggle dance indicate that the food source is far away. Waggling is a form of dance made in eight-shaped circular direction and has two components: the first component is a straight run and its direction conveys information about the direction of the food; the second component is the speed at which the dance is repeated and indicates how far away the food is. Bees repeat the waggle dance repeatedly giving information about the food source quality. The better the quality of food is, the greater the number of foragers recruited for harvesting is. The bee colony optimization (BCO) metaheuristic has been derived from this behavior and satisfactorily tested on many combinatorial problems [10]. *BeeAdHoc* is a reactive source routing algorithm based on the use of four different bee-inspired types of agents: packers, scouts, foragers, and bee swarms [5]. Packers mimic the task of a food-storekeeper bee, reside inside a network node, and receive and store data packets from the upper transport layer. Their main task is to find a forager for the data packet at hand. Once the forager is found and the packet is handed over, the packer will be killed. Scouts discover new routes from their launching node to their destination node (see Figure 1). A scout is broadcasted to all neighbors in range using an expanding time to live (TTL). At the start of the route search, a scout is generated; if after a certain amount of time the scout is not back with a route, a new scout is generated with a higher TTL in order to incrementally enlarge the search radius and increase the probability of reaching the searched destination. When a scout reaches the destination, it starts a backward journey on the same route that it has followed while moving forward toward the destination. Once the scout is back to its source node, it recruits foragers for its route by dancing. A dance is abstracted into the number of clones that could be made of the same scout. Foragers are bound to the beehive of a node. They receive data packets from packers and deliver them to their destination in a source-routed modality. To attract data packets foragers use the same metaphor of a waggle dance as scouts do. Foragers are of two types: delay and lifetime. From the nodes they visit, delay foragers gather end-to-end delay information, while lifetime foragers gather information about the remaining battery power. Delay foragers try to route packets along a minimum delay path, while lifetime foragers try to route packets in such a way that the lifetime of the network is maximized. A forager is transmitted from node to node using a unicast, point-to-point modality. Once a forager reaches the searched destination and delivers the data packets, it waits there until it can be piggybacked on a packet directed to its original source node. In particular, since TCP (transport control protocol) acknowledges received packets, *BeeAdHoc* piggybacks the returning foragers in the TCP acknowledgments. This reduces the overhead generated by control packets, saving energy at the same time. Bee swarms are the agents that are used to transport foragers back to their source node when the applications are using an unreliable transport protocol like

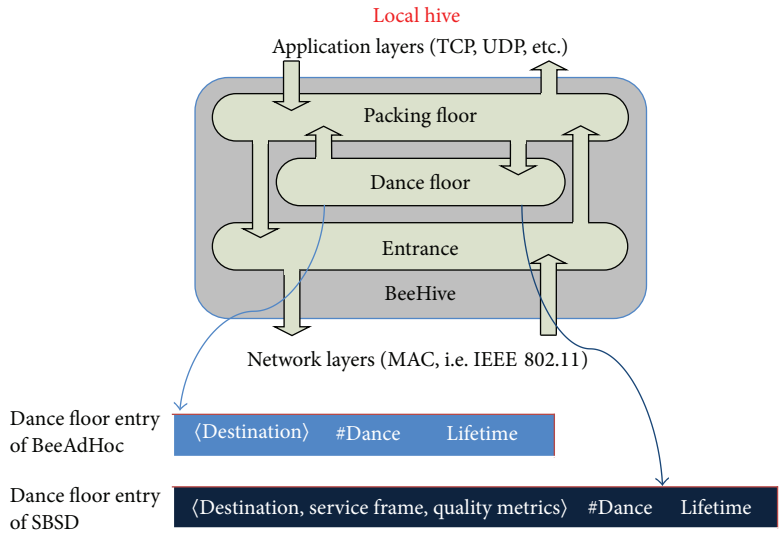

Figure 2: The network layer architecture of *BeeAdHoc*.

UDP (user datagram protocol). The algorithm reacts to link failures by using special hello packets and informing other nodes through route error messages (REM). In *BeeAdHoc*, each MANET node contains at the network layer a software module called hive. It consists of three parts: the packing floor, the entrance floor, and the dance floor (see Figure 2). The entrance floor is an interface to the lower MAC layer; the packing floor is an interface to the upper transport layer; the dance floor contains the foragers and the routing information. *BeeAdHoc* has been implemented and evaluated both in simulation and in real networks. Results demonstrate a very substantial improvement with respect to congestion handling, for example, due to hello messages overhead and flooding, and prove that the algorithm is far superior to common routing protocols, both single and multipath. Moreover, for *BeeAdHoc* mathematical tools have been utilized in order to overcome shortcomings of simulation-based studies such as their scenario specificity, scalability limitations, and time consumption. In [11, 12] mathematical models of two key performance metrics, routing overhead and route optimality have been presented providing valuable insight about the behaviour of the protocol.

2.2. BeeAdHocAutoConf Algorithm. *BeeAdHocAutoConf* is an IP address allocation algorithm based on the bee metaphor [13]. When a node wishes to join a network, it randomly picks up an address, starts setting up a local allocation table, and broadcasts a scout to all neighbours in its range using an expanding TTL. The TTL controls the number of times a scout may be rebroadcasted. Each scout is uniquely identified with a key based on its source node identifier (ID) and a sequence number. The task of the scout is twofold: it checks whether or not other nodes on its route are using the same address of its source node and brings back useful information whether it finds a duplicate address occurrence or not. The source node broadcasts the scout after assigning a small TTL to it and setting up a timer for itself. When the TTL expires, the scout might increment it in order to enlarge the search radius and increase the probability of reaching a node that might use a duplicate address. A maximum TTL is also established with respect to a reasonable size for an ad hoc network. Scouts with exceeded TTL might be killed or not depending on the information which they have gathered until then. This mechanism helps ensuring the address uniqueness when the TTL expires and useful address information has not been collected meaning that the source node is a network initiator. Scouts already seen (i.e., a copy) are deleted in order to limit the overhead.

3. MANET Service Discovery

3.1. Web Services and SOA. Web services is an evolving collection of standards, specifications, and implementation technologies in the areas of application integration and distributed computing. As defined by the W3C (http://www.w3.org/TR/ws-arch/): "a web service is a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other web-related standards." Web services do not necessarily need to exist on the World Wide Web (i.e., they can be located in an intranet) while implementation details about the distribution platform can be ignored by the programs that invoke the service. A web service is accessible through its APIs and specific invocation mechanisms (network protocol scheme, data encoding and so on). The service-oriented computing offers a new model for distributed application development, obtained through the integration of different applications and offered as services. A key element of this approach is SOA (service oriented architecture); an architectural style that is flexible enough

to allow the design of distributed applications from a set of functional units (services) available on the net and accessible through well-defined interfaces. The main goal of SOA is to ensure interoperability between different applications in order to build software systems based on loosely coupled components, which are combined dynamically. Applications are available on the network as services or integrated with other services. Finally, web services are the most suitable technology to implement SOA. A SOA architecture is based on three fundamental elements: the "Service Requestor," the "Service Provider," and the "Service Registry." The Service Provider provides a service via a standard middleware, makes it available to others over a network and, finally, manages its implementation. The Service Provider is responsible for creating a description of the service and for publishing it in one or more registries. It also receives all invocations for a specific service, providing the corresponding responses. The service description (a WSDL document) must contain all the information needed to use the service (server IP address, transport protocol, port number, and also interfaces of functions to invoke). The service description must be provided in a language neutral to the platform, to the programming language, and to the way in which the service has been implemented. The Service Requestor, or web service Client, invokes the service to ask for a specific functionality. It must firstly retrieve the description of the needed service and then use it to implement the binding process. The search operation in the registry is a name-based search: each service is uniquely identified by a specific name. The Service Requestor is responsible for the translation of the description of the service into the data structures needed to carry out the binding. The Service Registry is the component that advertises the service descriptions published by the Service Providers and allows Service Requestors to look for the requested functionality among the published descriptions. Each of these three roles can be played by any program or node in the network. In some circumstances, a single program may play more than one role, for example, a program could be both a Service Provider, providing Web services to invoke, and a Service Requestor that asks for a functionality offered by others nodes in the network. According to the aforementioned roles, SOA supports three types of operations: (1) publish (service description and publication), (2) find (the Service Requestor specifies the search criteria and the registry looks for descriptions corresponding to these criteria), and (3) bind (to connect a Service Requestor with a Service Provider). The "Service Discovery" process establishes the relationship between the Service Requestor and the Service Providers: it defines, in fact, the mechanism for locating Service Providers and retrieving the published service descriptions.

3.2. Universal Description Discovery and Integration. UDDI (universal description, discovery, and integration) is an XML-based centralized registry, independent from the platform, which allows publishing and querying service descriptions. The goal of UDDI is to facilitate the discovery of services both in the process of designing a service, and dynamically, at runtime. In the web services scenario, Service Providers publish in the Service Registry the information about where to retrieve the WSDL documents of the services. Service Requestors query the Service Registry to find out where to retrieve the WSDL documents, in order to invoke the services providing the needed functionalities. Due to a variety of reasons, service discovery in MANETs is a more challenging task. First, it has to allow wireless resource-constrained devices to discover services dynamically, while minimizing the traffic and tolerating the irregular connectivity of the network. Secondly, it has to provide service delivery to any other heterogenous device, regardless of its hardware and software platforms. Eventually, it has to enable service requesters to differentiate service instances according to provided nonfunctional properties, so that services match against the application quality of service requirements. In the sequel, we briefly review the literature main results.

3.3. Cross Layer Based Service Discovery. The service selection in MANETs requires the cross-layer integration of service discovery and selection with MANET routing mechanisms. The advantages of such a cross-layer approach over the traditional application layer implementation that preserves the modularity of the protocol stack are twofold. First, clients learn about available services and routes to servers offering them at the same time with obvious cost reduction and accuracy increase of service selection. Secondly, the existence of explicit routing information about path breaks or updates allows clients to efficiently detect changes in network topology and switch to nearby servers without additional cost. In [14] it has been proved that the network performance maximization requires that service selection decisions must be continuously reassessed to offset the effects of topology changes. It is also argued that, when multiple entries in the service table match a client's service description, a cross-layer approach allows the client to make a choice based on the lowest hop count and some service specific metrics like load and CPU usage. In [15] a multipath cross-layer service discovery (MCSD) for mobile ad hoc networks has been proposed that takes advantage of the network-layer topology information and the routing message exchange. The algorithm focuses on double-path cross-layer service discovery (DCSD), a special and most important case of MCSD. The iDCSD heuristic is also presented: from a number of candidate paths it finds the optimal routes from a client to a server and from a client to two servers by minimizing the hop count in the network layer. The MCSD protocol, however, selects multipath by considering only the lowest total hop count from a client to one or more servers without taking into account QoS metrics like available bandwidth and residual energy. The service update in multiple servers becomes difficult too. In [9] SISDA (swarm intelligence based service discovery architecture) has been developed, a swarm intelligence based service discovery architecture for MANETs. It is based on AntNet, an adaptive agent-based routing algorithm that has outperformed the best-known routing algorithms. It provides the service requestor (SR) to specify the operating context. For a set of mobile hosts, which are parts of the context defined; a cost effective routing

tree is constructed and maintained dynamically. The service discovery component (SDC) asks the service providers for the most suitable entry, for example, the entry that matches the QoS criterions specified in the client request.

3.4. Hierarchical Service Discovery. In [16] SGrid, a service discovery protocol based on a hierarchical grid has been presented. The network geographical area is divided into a two-dimensional hierarchical grid. The information about the available services is stored in directory nodes, one for each cell, along a trajectory properly defined with the aim of improving the efficiency of registration and discovery. Service providers register their services along the trajectory; requestors discover services along it and acquire the available information. The sparse node network topology is also avoided by means of a suitable process. In [17] the service discovery area (SDA) is spontaneously set up and managed by a service discovery area manager (SDAM) responsible for centralized service repository and service request processing. The protocol provides scalability to large MANET and can work efficiently without manual monitoring and management. Unfortunately, the SDAM and the centralized nature of it produce a considerable amount of overhead.

3.5. Routing Layer Based Service Discovery. In [18] the concept of service discovery provided with routing layer support was first introduced. For a proactively routed MANET a service reply extension added to topology updating messages provides both service and route discovery. For a reactively routed MANET the service discovery process follows the traditional route discovery process by means of the route request packets (RREQ) and the route reply packets (RREP). It further extends the idea by carrying a service request or reply in their respective areas by invoking the hybrid zone routing protocol (ZRP).

4. *BeeAdHocServiceDiscovery*

BeeAdHocServiceDiscovery (BAHSD) is a novel service discovery and selection algorithm based on honeybee foraging behaviour. It uses a decentralized cross-layer approach starting from the reactive routing algorithm *BeeAdHoc*.

BAHSD combines SOA architecture, namely, the Service Discovery phase, with *BeeAdHoc* and *BeeAdHocAutoConf*. Figure 3 illustrates such a mechanism. Each node has a hive organized as in Figure 2 and ready to store the services to offer (UDDI Service Registry). The hive architecture is the base routing mechanisms that BAHSD uses in its cross-layer approach each time a Service Requestor needs to look for a service and to invoke it soon after. Scouts realize the find operation when looking for food (source-destination path search); foragers realize the bind operation when collecting nectar (packet transmission); new foragers added to the dance floor of the hive realize the publish operation (forager recruited in order to specify the web service description). Table 1 maps the key concepts of the IP address autoconfiguration problem into the main components of the service discovery process for MANETs.

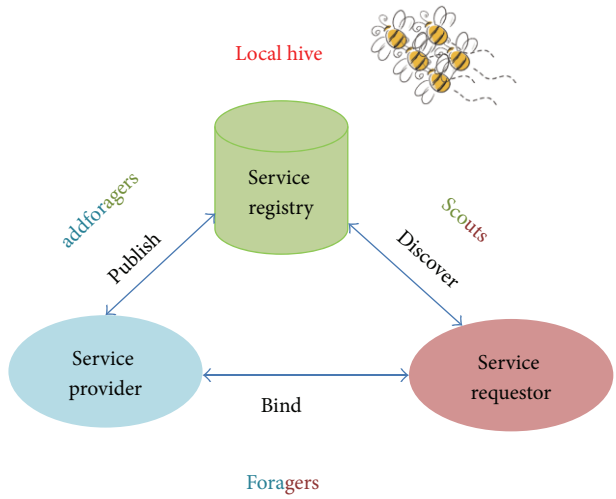

Figure 3: Schematic representation of services and mapping scout-forager into find-bind-publish operations.

Table 1: *BeeAdHocServiceDiscovery* main components mapping from IP address autoconfiguration into service discovery.

IP Address autoconfiguration	Service discovery
Allocation table	Service registry UDDI
IP address	Business service
Duplicate address search	API inquiry UDDI specification
IP address assignment	API publish UDDI specification
Node leaving the MANET	API delete UDDI specification

More precisely, *each* MANET node accomplishes the Service Registry functionality, when it is either Requestor or Provider. The hive is a local Service Registry, namely, the UDDI registry that publishs descriptions of services provided by neighbour nodes in the form of businessService entities. An extension of the dance floor definition in *BeeAdHoc* allows implementing the local Service Registry. Each entry of the dance floor is indeed a different forager for each different pair (destination, businessService). Table 2, and specifically the second row, shows the entry format in the dance floor for BAHSD as compared with its correspondent in *BeeAdHoc*: it contains a Routing Frame, a Service Frame and, finally, Quality metrics information. Specifically, information about CPU load, free memory percentage, bandwidth, number of active network connections, number of wait connections, and context information [19–21] represent some of the information that can be gathered on the nodes and piggybacked in the ACK segment towards the source node. In this way, the source node takes the information from the received packet as they are sent back to it, allowing scheduling decision based on the most recent results. As a consequence, no overhead is introduced in the network, in fact, load information is not periodically exchanged.

Two main components, as shown in Figure 4, constitute the architecture of BAHSD: the *service description publication* (SDP) and the *BeeSwarmServiceDiscovery* (BSSD). In the SDP component, each hive configures its local UDDI, by

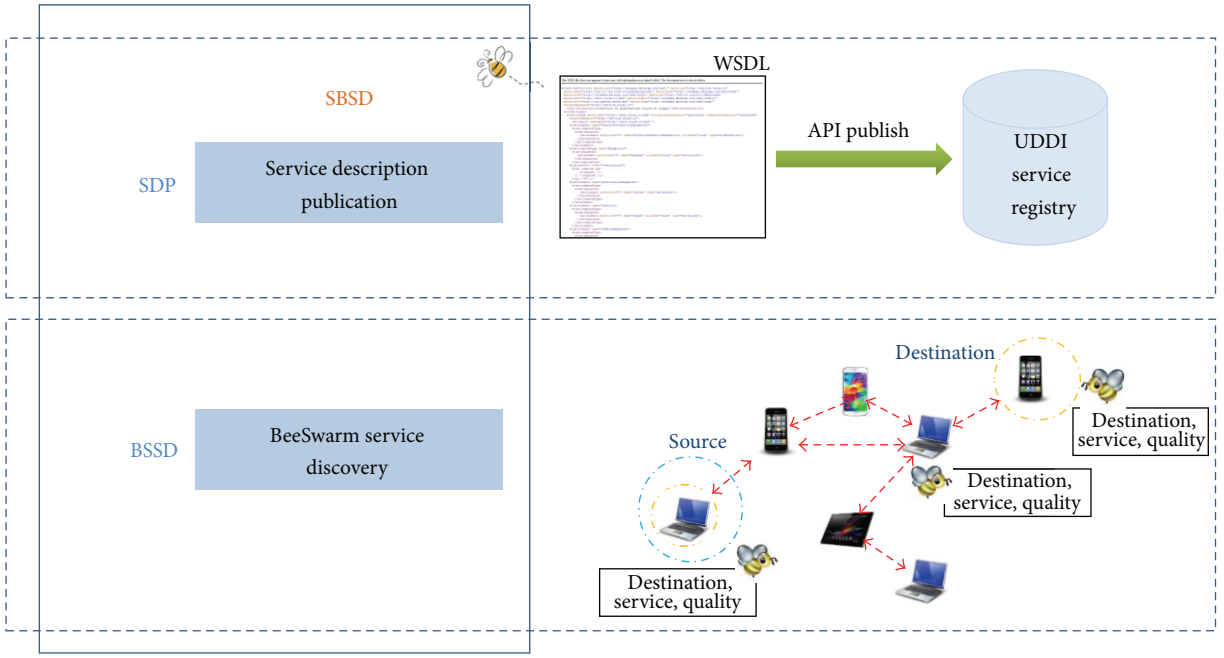

Figure 4: Main components of the SBSD Protocol: the SDP for service publication and the BSSD for service discovery.

Table 2: BAHSD and *BeeAdHoc* entries in the dance floor.

Algorithm	Routing information		
BeeAdHoc	Destination	# dance	lifetime
BAHSD	Destination, service frame, and quality metrics	# dance	lifetime

publishing the Web Service descriptions that can be invoked. This operation is performed by using the "*save service*" method provided by the *API Publish* UDDI specification. BSSD exploits the *BeeAdHoc* routing operations to gather descriptions and locations about the requested service.

4.1. The Algorithm. The algorithm description will be done by means of the three logical blocks that correspond to the operations in the *Dance Floor*, *Packing Floor*, and the *Entrance Floor*, respectively. For each of them new functions have been implemented with respect to those of *BeeAdHoc* in order to support the service discovery mechanisms. The following list provides an explanation of the symbols used in the code:

s: Packet source node,

d: Packet destination node,

i: Current node,

j: Any MANET node,

S_{sd}: Scout bee sent from s to d,

F_{sd}: Forager bee sent from s to d,

D_{sd}: Data packet sent from s to d,

P_{sd}: Any packet received at i with source s and dest. d,

h_{next}: Next hop address,

SF: Service Frame, find service/businessService datatype,

L_{SF}: Forager list for a given SF.

Dance Floor. It implements the *addForager* and *getForager* functions. The first of them is equivalent to a *save_service* operation into the local UDDI Registry; the second of them is equivalent to a *find_service* operation into the local UDDI Registry.

The *addForager* function, in the Algorithm 1, computes the number of packers waiting for F_{sd} and the values of the path quality metrics (lines 4-5); it also checks whether a list L_{SF} of foragers already exists for the SF corresponding to F_{sd} in order to possibly create it and update the dance number (lines 6–9). The *getForager* function makes a lookup into the *Dance Floor* with the aim to search for at least one forager matching the SF input service description; it might return a random chosen forager or a null value.

Packing Floor. It implements, as shown in the Algorithm 2, the service requests entailed from the upper layer and takes care of packets attained from the *Entrance Floor* with different operations whether the incoming packet is either a forager or a scout. For each received SF from the upper layer, the local registry UDDI might already have the requested information (*getForager* returns F_{sd}) or might not have it (*getForager* returns a null value) requiring a new scout creation (lines 1–15).

For each received packet P_{sd} from the *Entrance Floor* (lines 18–39), either a forager is added into the *Dance Floor* (*addForager*) or a different forager F_{sj} is created for each Service Frame that the scout collected on the path s-j. However, in both cases, for each forager F_{sj} added into

```
(1)  /* add a forager on the dance floor */
(2)  void addForager(F_sd)
(3) {
(4)     var waitingPackers = getPackerInQueueForThisForager(F_sd);
(5)     var qualityMetric = getParameterCollectFromForager(F_sd);
(6)     if (L_SF not exist for F_sd · SF)
(7)        create L_SF;
(8)     add F_sd to L_SF;
(9)     updateDanceNumber (F_sd, waitingPackers, qualityMetric);
(10) }
(11)
(12)
(13) /* lookup a specific forager on dance floor */
(14) matchingForager getForager(SF)
(15) {
(16)    var tmp = NULL;
(17)    if (F_sd exists in L_SF) {
(18)       while (tmp == NULL && F_sd exists in L_SF) {
(19)          choose according to quality metrics or randomly
(20)                a F_sd among multiple foragers in L_SF;
(21)          if (F_sd · lifetime > currentTime)
(22)             if (F_sd · danceNumber > 0) {
(23)                tmp = copy(F_sd);
(24)                decrease danceNumber;
(25)             }
(26)             else {
(27)                tmp = F_sd;
(28)                delete F_sd from dance floor;
(29)             }
(30)          else
(31)             kill F_sd;
(32)       }
(33)    }
(34)    return tmp;
(35) }
```

Algorithm 1: Services provided by the Dance Floor.

the *Dance Floor*, the presence of data packet D_{sj} in the send buffer waiting for it will be verified.

Entrance Floor. It manages, as shown in the Algorithm 3, the foragers and the scouts coming from the MAC layer (lines 1-9). A forager might be sent to the *Packing Floor* or to the next hop (after having measured quality metrics), whether it has reached its destination or not. A scout might flying towards its source node s or the destination d (lines 10–49). If is flying to the source s (Service Requestor) in each intermediate node will be forwarded to the next hop by means of the function *sendPacketToNextHop*, whereas in s will be forwarded to the *Packing Floor*. If the scout is flying towards the destination d (Service Provider), in each intermediate node i, will invoke the function *getForager* from the *Dance Floor*; if the function returns a forager F_{sd} the search ends successfully (i.e., the UDDI registry of the node i knows the path toward the desired Service Provider d). At this point the scout will become a backward scout, coming back to the source by bringing the following information: (1) the description of the requested service; (2) the description of the other services gathered during the path; (3) the complete route toward the Service Provider d, built by concatenating the path until i with the route from i to d. Conversely, if the *getForager* returns a null value, it means that there are no foragers for the required service. Now, if the TTL is not expired and the scout is not present in the list of the scouts seen by the node i, it will be retransmitted in broadcast to all nodes neighbours of i. Finally, if (1) the TTL is expired, (2) the scout is not available in the list of scouts seen by the node i, and (3) the list of SF gathered along the path is not empty, then the scout will not be discarded but returned to the source in order to update the Local UDDI registry with all SF descriptions collected along the taken path.

Figure 6 is a concise representation of the *BeeAdHocServiceDiscovery* working principles.

Network merging does not affect the protocol operation. On the contrary, when two or more networks merge, it may happen that a service is offered by two different service providers allowing straightforward load balancing, energy saving, and reduced band consumption. However, network partition may create difficulties. In this case, a node invoking

```
(1)  /* service requests received from higher layers */
(2)  for each (SF received from higher layers) {
(3)     var F_sd = danceFloor · getForager(SF);
(4)     if (F_sd != NULL) {
(5)        encapsulate D_sd into the payload of F_sd;
(6)        send F_sd to entranceFloor;
(7)     }
(8)     else {
(9)        insert D_sd into the packet queue;
(10)       create a new scout S_sd with ID, initial TTL;
(11)       encapsulate SF into the header of S_sd;
(12)       set timer of S_sd;
(13)       send S_sd to entranceFloor;
(14)    }
(15) }
(16)
(17) /* packets coming from entrance floor */
(18)    for each (P_sd received from entrance) {
(19)       if (P_sd is a forager) {
(20)          danceFloor · addForager(P_sd);
(21)          extract D_sd from the payload of forager;
(22)          deliver D_sd to higher layers;
(23)       }
(24)       else if (P_sd is a scout) {
(25)          for each (SF j gathered by P_sd) {
(26)             create a forager F_sj foreach SF_j;
(27)             danceFloor · addForager(F_sj);
(28)          }
(29)          kill P_sd;
(30)       }
(31)       for each (F_sj add to dance floor) {
(32)          var packers = getNumberPacketInQueueForForager(F_sj);
(33)          while (packers > 0 && F_sj · danceNumber > 0) {
(34)             encapsulate D_sj into payload of F_sj;
(35)             send F_sj to entrance;
(36)             decrease packers;
(37)          }
(38)       }
(39)    }
(40)
(41) /* check scout return */
(42) if (timer expired and scout S_sd not returned) {
(43)    compute newTTL of S_sd and assign it a newID;
(44)    set timer of S_sd;
(45)    send it to entrance;
(46) }
```

Algorithm 2: Actions taken at the Packing Floor.

a web service offered by a provider, which is not anymore in its partition, needs to restart the service discovery process in order to locate an equivalent service among the ones offered by the server providers belonging to its own partition. Examples of network merging and network partition are shown in Figure 5.

5. Formal Performance Evaluation

In [11] an evaluation framework to model traditional performance metrics of MANET routing algorithms, such as routing overhead and route optimality, is recommended in order to bypass the drawbacks arising from the use of network simulators. On one hand, indeed, Kurkowski et al. [22] have shown that many simulations in the ad hoc network scenarios lack repeatability, fairness, rigour, and statistical soundness. On the other hand, scalability experiments become often unworkable because of the time demand fast growth. Therefore, it is preferred that simulation-based estimates for ad hoc network protocols, would rather be carried out with the use of a mathematical model evaluating their fundamental performance parameters in order to guarantee fair and

```
(1)  /* forager received from MAC layer */
(2)  for each (F_sd received from MAC layer) {
(3)    if (F_sd arrived at d)
(4)      send F_sd to packingFloor;
(5)    else {
(6)      collect optimization parameter from the node;
(7)            //delay, lifetime, or energy consumption
(8)      sendPacketToNextHop(F_sd);
(9)    }
(10) }
(11)
(12) /* scout received from MAC layer */
(13) for each (S_sd received from MAC layer) {
(14)   if (S_sd is on return path toward s) {
(15)     if (S_sd is at s)
(16)       send S_sd to packingFloor;
(17)     else {
(18)       sendPacketToNextHop(S_sd);
(19)     }
(20)   }
(21)   else if (S_sd is on forward path toward d) {
(22)     var F_id = danceFloor · getForager(S_sd · SF);
(23)     if (F_id != NULL) {
(24)       insert in payload of S_sd  F_id · SF;
(25)       insert in payload of S_sd other foragers in danceFloor;
(26)       complete the route by concatenating S_si + F_id;
(27)       change S_sd to a backward scout;
(28)       reverse the source route in the header;
(29)       sendPacketToNextHop (S_sd);
(30)     }
(31)     else {
(32)       if (TTL expired || S_sd exists in seenScoutList)
(33)         if (SF list collected from S_sd is empty)
(34)           kill S_sd;
(35)         else {
(36)           insert in payload of S_sd foragers from danceFloor;
(37)           change S_sd to a backward scout;
(38)           reverse the source route in the header;
(39)           sendPacketToNextHop(S_sd);
(40)         }
(41)       else {
(42)         insert in payload of S_sd foragers from danceFloor;
(43)         insert address i in the source route header;
(44)         insert ID and source of S_sd into seenScoutList;
(45)         decrease TTL;
(46)         broadcast S_sd to all neighbors of i;
(47)       }
(48)     }
(49)   }
(50) }
(51)
(52) /* forager received from packing floor */
(53) for each (F_sd received from Packing floor) {
(54)   collect optimization parameter from the node;
(55)   sendPacketToNextHop(F_sd);
(56) }
(57)
(58) /* scout received from packing floor */
(59) for each (S_sd received from Packing floor) {
(60)   insert address i in the source route header;
(61)   insert ID and source of S_sd into seenScoutList;
```

Algorithm 3: Continued.

```
(62)    decrease TTL;
(63)    broadcast S_sd to all neighbors of i;
(64) }
(65)
(66) void sendPacketToNextHop (P_sd)
(67) {
(68)    find h_next in the source route header;
(69)    send P_sd to MAC Interface of h_next;
(70) }
```

Algorithm 3: Actions taken at the Entrance Floor.

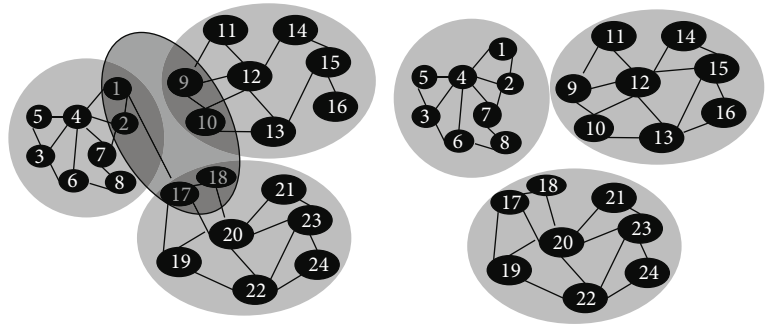

Figure 5: Examples of topology changes.

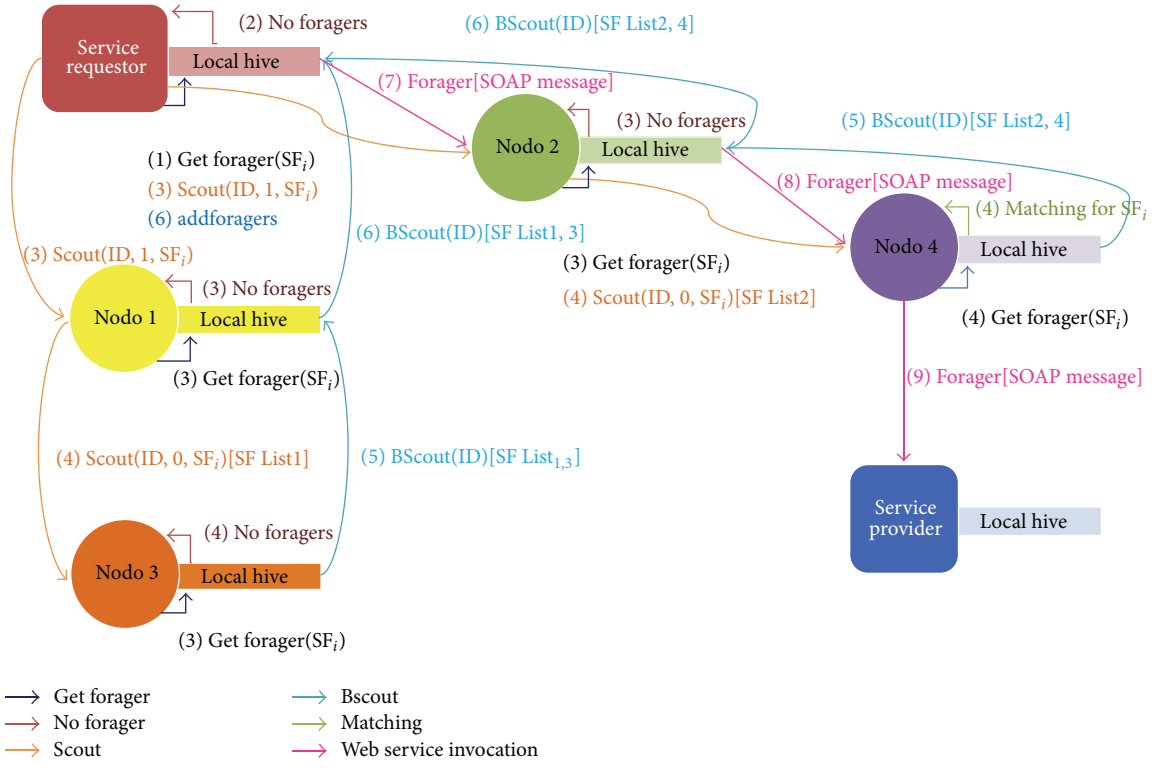

Figure 6: *BeeAdHocService Discovery* working principles.

provable comparisons as well as early investigations about the algorithm validity. In this context, *BeeAdHocServiceDiscovery* routing overhead and route optimality have been evaluated following the model presented in [12]. Thus, a dense MANET of N nodes, homogeneously distributed according to Poisson distribution with node density ρ and connected by symmetric links, is considered. The network runs a CSMA/CA-based MAC layer protocol for contention resolution, may lose packet because of channel errors, does not change its topology during a route discovery process, counts on stable channel conditions during the scout transmission between two nodes, and transmits at a single uniform rate.

5.1. Service Discovery Overhead. The service discovery overhead can be defined as the number of scouts generated in the network up to a particular number of hops (h) during a route discovery phase since in *BeeAdHocServiceDiscovery* the routing and discovery processes coincide. Thus, the definition of discovery overhead mirrors the time to live (TTL) mechanism used in *BeeAdHoc* during a route discovery. When TTL expires, nodes stop rebroadcasting the scouts. The expected number of scouts up to h hops from the source node can be calculated as follows:

$$C_p^{(BeeAdHoc)} = 1 + \overline{p_c} \cdot d_{\text{avg}} + \left(\overline{p_c}\right)^2 \cdot d_{\text{avg}} \cdot d_{f[1]} + \cdots + \left(\overline{p_c}\right)^h \cdot d_{\text{avg}} \cdot d_{f[1]} \cdot d_{f[2]} \cdot \ldots \cdot d_{f[h-1]}, \tag{1}$$

where d_{avg} is the average degree of the node, $d_{f[1]}, d_{f[2]}, \ldots, d_{f[h-1]}$, the expected forward degree of nodes at $1, 2, \ldots, h-1$ hops from the source node, respectively, and pc the probability of no collision that for an IEEE 802.11b MANET with its distributed coordination function (DCF) at MAC layer is given by

$$p_c = 1 - \left(1 - \frac{1}{\text{CW}_{\text{min}}}\right)^{M-1}, \tag{2}$$

with CW_{min} (= 31 for 802.11b) the minimum contention window and M the number of contending nodes. It is worth noting that the expected forward degree might be assumed a constant either when the network is sparse with nodes having very small overlapping transmission regions or when the network size is very small.

Assuming that $d_{f[1]} = d_{f[2]} = d_{f[3]} = \cdots, d_{f[h-1]} = d_f$, (1) simplifies to

$$C_p = \begin{cases} 1 + h\overline{p_c} d_f d_{\text{avg}} & \text{if } \overline{p_c} d_f = 1 \\ 1 + \overline{p_c} d_{\text{avg}} \left(\dfrac{1 - \left(\overline{p_c} d_f\right)^h}{1 - \overline{p_c} d_f} \right) & \text{otherwise,} \end{cases} \tag{3}$$

where according to [12] d_f becomes

$$d_f \simeq \frac{d_{\text{avg}} - \rho r_0^2 \left((2\pi/3) - \left(\sqrt{3}/2\right)\right)}{2}. \tag{4}$$

5.2. Service Discovery Route Optimality. *BeeAdHocServiceDiscovery* lays on the *Beeadhoc* routing algorithm; thus, it keeps multiple paths to any pair source-destination (s-d) that generated a scout because of a routing request. Each scout reaching the destination is expected to find a new path. Now, if k edge disjoint paths are assumed to be between (s-d) and a t hops path is assumed to be an optimal path, a function $f[i-t]$ giving the total number of edge-disjoint paths of length i between (s-d) can be introduced. *BeeAdHocServiceDiscovery* finds links with probability p_c and therefore an optimal path with probability $\varepsilon = (1 - p_c)^{t-1}$, as *Beeadhoc* does. Thus, the probability of discovering j optimal paths out of a total of $f[0]$ optimal paths is the binomial distribution:

$$b\left(j; f[0], \varepsilon\right) = P\left(X[t] = j\right) = \binom{f[0]}{j} \varepsilon^j (1 - \varepsilon)^{f[0]-j}, \tag{5}$$

where $X[t]$ is the number of t hop paths effectively found. Consequently, the probability of finding at least a single optimal path is given by

$$P\left(X[t] \geq 1\right) = 1 - (1 - \varepsilon)^{f[0]}, \tag{6}$$

where $(1 - \varepsilon)^{f[0]}$ is the failure probability in discovering an optimal path.

6. Conclusions

BeeAdHocServiceDiscovery is a new protocol of service discovery and selection for MANET based on the foraging behaviour of honeybees. It uses a cross layer mechanism that allows gathering routing information, such as path breaks and updates, in order to minimize the number of control messages and to optimize the node selection with respect to the used metrics. As a consequence, interesting advantages for the web service accuracy and the network load balancing can be achieved. *BeeAdHocServiceDiscovery* maps the key concept of the MANET autoconfiguration algorithm *BeeAdHocAutoConf* into the main components of a MANET service discovery process. Moreover, by using the overall functionality of a reactive multipath routing algorithm such as *BeeAdHoc*, it saves all features of efficiency, scalability, robustness, decentralization, adaptivity, and autoorganization of it. Moreover, the formal evaluation of two traditional metrics, such as discovery overhead and route optimality, shows that the *BeeAdHocServiceDiscovery* performance does not vary at all with respect to *Beeadhoc*.

Simulations are currently under investigation by means of the Network Simulator Version-3 (ns-3) [23]. In our simulation, the channel capacity of all mobile hosts is set to 2 Mbps and the MAC layer protocol works with the distributed coordination function (DCF) of IEEE 802.11 for wireless LAN for a scenario described in Table 3.

Among the total nodes, half of them are treated as clients and the remaining ones as service providers. The comparison algorithms are MCSD [15] and SISDA [8] with average delay, drop, and packet delivery ratio taken into account as performance evaluation metrics both in the case of real-time and non-real-time services. Even though partial results

Table 3: Simulation settings.

Parameters	Values
Simulation area	35 m × 35 m–200 m × 200 m
Network size	50–1600
Mobility pattern	Random walk 2 d mobility
Coverage range	30 m
Simulation number	300

show that *BeeAdHocServiceDiscovery* performs better with respect to MCSD and SISDA, some more experimentations are required in order to assert its definitive power.

Moreover, another ongoing work is about the analysis of the accuracy of our algorithm in terms of the number of correct nodes discovered when looking for a specific web service. Analysis will also include the correctness of the algorithm when different criterions are taken into account.

The next step in the development of *BeeAdHocServiceDiscovery* will be the extension of the web service selection criteria that should include more performance parameters. Both the energy and privacy constraints [24–27] will also be taken into account. Eventually performance and simulation experiments will be performed paying attention to the use of different forms of swarms too.

Conflict of Interests

The authors declare that there is no conflict of interests regarding the publication of this paper.

References

[1] E. M. Royer and C.-K. Toh, "A review of current routing protocols for ad hoc mobile wireless networks," *IEEE Personal Communications*, vol. 6, no. 2, pp. 46–55, 1999.

[2] S. Nesargi and R. Prakash, "MANETconf: configuration of hosts in a mobile ad hoc network," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies (IEEE INFOCOM '02)*, vol. 2, pp. 1059–1068, New York, NY, USA, 2002.

[3] P. Choudhury, A. Sarkar, and N. C. Debnath, "Deployment of service oriented architecture in MANET: a research roadmap," in *Proceedings of the 9th IEEE International Conference on Industrial Informatics (INDIN '11)*, pp. 666–670, July 2011.

[4] H. Wedde and M. Farooq, "BeeHive: routing algorithms inspired by honey bee behavior," *KI*, vol. 19, no. 4, pp. 18–24, 2005.

[5] H. F. Wedde and M. Farooq, "The wisdom of the hive applied to mobile ad-hoc networks," in *Proceedings of the IEEE Swarm Intelligence Symposium (SIS '05)*, pp. 341–348, Pasadena, Calif, USA, June 2005.

[6] H. F. Wedde, M. Farooq, T. Pannenbaecker et al., "Beeadhoc: an energy efficient routing algorithm for mobile ad hoc networks inspired by bee behavior," in *Proceedings of the 7th Annual Conference on Gene tic and Evolutionary Computation (GECCO '05)*, pp. 153–160, ACM, New York, NY, USA, 2005.

[7] M. Dorigo and C. Blum, "Ant colony optimization theory: a survey," *Theoretical Computer Science*, vol. 344, no. 2-3, pp. 243–278, 2005.

[8] E. Bonabeau, M. Dorigo, and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems*, Oxford University Press, New York, NY, USA, 1999.

[9] S. Pariselvam and R. M. S. Parvathi, "Swarm intelligence based service discovery architecture for mobile ad hoc networks," *European Journal of Scientific Research*, vol. 74, no. 2, pp. 205–216, 2012.

[10] D. Teodorović, P. Lučić, G. Marković, and M. Dell'Orco, "Bee colony optimization: principles and applications," in *Proceedings of the 8th Seminar on Neural Network Applications in Electrical Engineering (NEUREL '06)*, pp. 151–156, September 2006.

[11] M. Saleem, S. Khayam, and M. Farooq, "Formal modeling of *BeeAdHoc*: a bio-inspired mobile ad hoc network routing protocol," in *Ant Colony Optimization and Swarm Intelligence*, vol. 5217 of *Lecture Notes in Computer Science*, pp. 315–322, Springer, Berlin, Germany, 2008.

[12] M. Saleem, S. A. Khayam, and M. Farooq, "A formal performance modeling framework for bio-inspired Ad Hoc routing protocols," in *Proceedings of the 10th Annual Genetic and Evolutionary Computation Conference (GECCO '08)*, pp. 103–110, July 2008.

[13] F. D. Santis, "An efficient bee-inspired auto-configuration algorithm for mobile ad hoc networks," *International Journal of Computer Applications*, vol. 57, no. 17, pp. 9–14, 2012.

[14] A. Varshavsky, B. Reid, and E. de Lara, "A cross-layer approach to service discovery and selection in MANETs," in *Proceedings of the 2nd IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS '05)*, pp. 459–466, Washington, DC, USA, November 2005.

[15] X. Shao, L. H. Ngoh, T. K. Lee, T. Chai, L. Zhou, and J. Teo, "Multipath cross-layer service dis covery (MCSD) for mobile ad hoc networks," in *Proceedings of the IEEE Asia-Pacific Services Computing Conference (APSCC '09)*, pp. 408–413, December 2009.

[16] H.-W. Tsai, T.-S. Chen, and C.-P. Chu, "Service discovery in mobile ad hoc networks based on grid," *IEEE Transactions on Vehicular Technology*, vol. 58, no. 3, pp. 1528–1545, 2009.

[17] Y. Chen and Z. Mi, "A novel service discovery mechanism in MANET using auto-configured SDA," in *Proceedings of the International Conference on Wireless Communications, Networking and Mobile Computing (WiCOM '07)*, pp. 1660–1663, September 2007.

[18] C. Ververidis and G. Polyzos, "Extended ZRP: a routing layer based service discovery protocol for mobile ad hoc networks," in *the 2nd Annual International Conference on Mobile and Ubiquitous Systems: Networking and Services (MobiQuitous '05)*, pp. 65–72, July 2005.

[19] R. Grieco, D. Malandrino, and V. Scarano, "SEcS: scalable edge-computing services," in *Proceedings of the 20th Annual ACM Symposium on Applied Computing (SAC '05)*, pp. 1709–1713, March 2005.

[20] R. Grieco, D. Malandrino, and V. Scarano, "A scalable cluster-based infrastructure for edge-computing services," *World Wide Web*, vol. 9, no. 3, pp. 317–341, 2006.

[21] R. Grieco, D. Malandrino, F. Mazzoni, and D. Riboni, "Context-aware provision of advanced Internet services," in *Proceedings of the 4th Annual IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops '06)*, pp. 600–603, March 2006.

[22] S. Kurkowski, T. Camp, and M. Colagrosso, "MANET simulation studies: the incredibles," *ACM SIGMOBILE Mobile*

Computing and Communications Review, vol. 9, no. 4, pp. 50–61, 2005.

[23] "ns-3," http://www.nsnam.org/.

[24] D. Malandrino, A. Petta, V. Scarano, L. Serra, R. Spinelli, and B. Krishnamurthy, "Privacy awareness about information leakage: who knows what about me?" in *Proceedings of the 12th ACM Workshop on Workshop on Privacy in the Electronic Society (WPES '13)*, pp. 279–284, 2013.

[25] D. Malandrino and V. Scarano, "Privacy leakage on the Web: Diffusion and countermeasures," *Computer Networks*, vol. 57, no. 14, pp. 2833–2855, 2013.

[26] S. D'Ambrosio, S. De Pasquale, G. Iannone et al., "Phone batteries draining: is GWeB (Green Web Browsing) the solution?" in *Proceedings of the International Green Computing Conference (IGCC '14)*, Dallas, Tex, USA, November 2014.

[27] D. Malandrino, V. Scarano, and R. Spinelli, "How increased awareness can impact attitudes and behaviors toward online privacy protection," in *Proceedings of the International Conference on Social Computing (SocialCom '13)*, pp. 57–62, September 2013.