BeeAdHocServiceDiscovery
A MANET Service Discovery Algorithm based on Bee Colonies

Gianmaria Arenella, Filomena de Santis and Delfina Malandrino
Dipartimento di Informatica, University of Salerno, Via Giovanni Paolo II, Fisciano, Italy

Keywords: Swarm Intelligence, Routing, MANET Auto-configuration, Web Service Discovery.

Abstract: In a mobile ad-hoc network, nodes are self-organized without any infrastructure support: they move arbitrarily causing the network to experience quick and random topology changes, have to act as routers as well as forwarding nodes, some of them do not communicate directly with each other. Routing, IP address auto-configuration and Web service discovery are among the most challenging tasks in the MANET domain. Swarm Intelligence is a property of natural and artificial systems involving minimally skilled individuals that exhibit a collective intelligent behaviour derived from the interaction with each other by means of the environment. Colonies of ants and bees are the most prominent examples of swarm intelligence systems. Flexibility, robustness, and self-organization make swarm intelligence a successful design paradigm for difficult combinatorial optimization problems. This paper proposes *BeeAdHocServiceDiscovery* a new service discovery algorithm based on a bee swarm labour that may be applied to large scale MANET with low complexity, low communication overhead, and low latency. Eventually, future research directions are established.

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 multi-hop. 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 pre-installed resources. Remarkable uses of mobile ad-hoc networks are in calamity and military scenario; with the increasing diffusion of radio technologies, many multimedia applications take also advantages from running over them. MANETs suffer from a variety of questions: the routing and the IP (Internet Protocol) address auto-configuration problems are among the most challenging ones. Many different approaches dealing with them do exist, even though there are not algorithms that fit in all cases (Royer and Toh, 1999; Nesargi and Prakash, 2002). 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) (Choudhury et al., 2011). 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 meta-heuristic (Wedde and Farooq, 2005a; Wedde and Farooq, 2005b; Wedde, Horst F. *at al*., 2005; Dorigo and Blum, 2005). Swarm Intelligence (SI) is a well-known distributed paradigm for the solution of hard problems taking insight from biological scenario such as colonies of ants, bees, and termites, schools of fish, 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

Arenella G., de Santis F. and Malandrino D..
BeeAdHocServiceDiscovery - A MANET Service Discovery Algorithm based on Bee Colonies.
DOI: 10.5220/0005045902440251
In *Proceedings of the 11th International Conference on Informatics in Control, Automation and Robotics* (ICINCO-2014), pages 244-251
ISBN: 978-989-758-039-0
Copyright © 2014 SCITEPRESS (Science and Technology Publications, Lda.)

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 auto-configuration and the service discovery problems (Bonabeau et al., 1999; Wedde and Farooq, 2005a; Pariselvam and Parvathi, 2012). The remainder of this paper is organized as follows. Section 2 reviews the basics of the bee colony optimization meta-heuristic and routing and auto-configuration 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. Finally, 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, 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. 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 indicate 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 is the quality of food, the greater is the number of foragers recruited for harvesting. The Bee Colony Optimization (BCO) meta-heuristic has been derived from this behavior and satisfactorily tested on many combinatorial problems (Teodorovic et al., 2006). *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, (Wedde and Farooq, 2005b). Packers mimic the task of a food-storekeeper bee, reside inside a network node, 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. 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 at the same time energy. 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 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. The entrance floor is an interface to the lower MAC layer, the packing floor is an interface to the upper transport layer while 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 the algorithm 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 consume. In (Saleem et al., 2008) 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 (De Santis, 2012). 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 re-broadcasted. 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 either if 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, 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 that on their route have been seen already are deleted in order to limit the overhead.

3 MANET SERVICE DISCOVERY

3.1 Web Services & 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: "*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 a specific invocation mechanisms. The service-oriented computing offers a new model for distributed application development, obtained through the integration of different applications, 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 of creating a description of the service and of 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. The Service Requestor 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 of 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, e.g., a program could be both a Service Provider, providing a Web service to different applications, both a Service Requestor, by invoking a Web service offered by others. According to the aforementioned roles, SOA supports three types of operations: (1) publish (service description and publication), (2) find (search for services that match the provided criteria), (3) bind (connection 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 locate service providers and retrieve 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 to publish and to query 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 (Varshavsky et al., 2005) it has 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 (Shao et al., 2009) a multi-path cross-layer service discovery (MCSD) for mobile ad hoc networks has been proposed that takes advantage from 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 (Pariselvam and Parvathi, 2012) 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 oper-

ating context. For a set of mobile hosts, which are parts of the context defined, a cost effective routing tree is constructed and maintained dynamically. For a client's service request, the service discovery component (SDC) lookup for the service providers with most relevant QoS entries matching the QoS request of the service requester.

3.4 Hierarchical Service Discovery

In (Tsai et al., 2009) 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 (Chen and Mi, 2007) 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 (Ververidis and Polyzos, 2005) 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*.

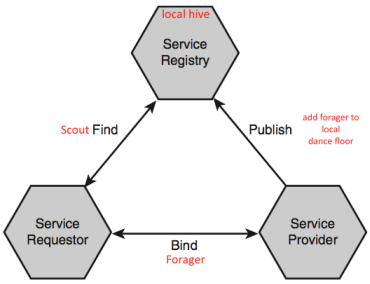

Figure 1: Mapping scout-forager into find-bind-publish operations.

BAHSD combines SOA architecture, namely the Service Discovery phase, with *BeeAdHoc* and *BeeAdHocAutoConf*. Fig. 1 illustrates such a mechanism. Each node has a hive ready to store services offered by it (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 auto-configuration problem into the main components of the service discovery process for MANETs.

Table 1: *BeeAdHocServiceDiscovery* main components mapping from IP address auto-configuration into service discovery.

IP Address Auto-configuration	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 is either Requestor or Provider. The hive is a local Service Registry, namely the UDDI registry that publish 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) and it contains a Routing Frame as well as a Service Frame.

Two main components constitute the architecture of BAHSD: the *Service Description Publication*

(SDP) and the *BeeSwarmServiceDiscovery* (BSSD). In SDP each hive configures its local UDDI, by publishing by means of a "*save_service*" (*API Publish*) operation, the Web Service descriptions offered by itself to all close nodes. 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. Table 2 provides an explanation of the symbols used in the code.

Table 2: Symbols used in the code.

Symbol	Description
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 computes the number of packers waiting for F_{sd} and the values of the path quality metrics; it also checks whether a list L_{SF} of foragers does already exists for the SF corresponding to F_{sd} in order to possibly create it and update the dance number. 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.

Algorithm 1: Services provided by Dance Floor.

```
/* add a forager on dance floor */
void addForager(Fsd)
{
  var waitingPackers = getPackerInQueueForThisForager(Fsd);
  var qualityMetric = getParameterCollectFromForager(Fsd);
  if (LSF not exist for Fsd.SF)
    create LSF;
  add Fsd to LSF;
  updateDanceNumber(Fsd, waitingPackers, qualityMetric);
}

/* lookup a specific forager on dance floor */
matchingForager getForager(SF)
{
  var tmp = NULL;
  if (Fsd exists in LSF) {
    while(tmp == NULL && Fsd exists in LSF) {
      choose randomly a Fsd among multiple foragers in LSF;
      if (Fsd.lifetime > currentTime)
        if (Fsd.danceNumber > 0) {
          tmp = copy(Fsd);
          decrease danceNumber;
        }
        else {
          tmp = Fsd;
          delete Fsd from dance floor;
        }
      else
        kill Fsd;
    }
  }
  return tmp;
}
```

Packing Floor: it implements 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.

For each received packet P_{sd} from the *Entrance Floor*, 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 the *Dance Floor*, the presence of data packet D_{sj} in the send buffer waiting for it will be verified.

Algorithm 2: Actions taken at Packing Floor.

```
/* service requests received from higher layers */
for each(SF received from higher layers) {
  var Fsd = danceFloor.getForager(SF);
  if (Fsd != NULL) {
    encapsulate Dsd into the payload of Fsd;
    send Fsd to entranceFloor;
  }
  else {
    insert Dsd into the packet queue;
    create a new scout Ssd with ID, initial TTL;
    encapsulate SF into the header of Ssd;
    set timer of Ssd;
    send Ssd to entranceFloor;
  }
}

/* packets coming from entrance floor */
  for each(Psd received from entrance) {
    if (Psd is a forager) {
      danceFloor.addForager(Psd);
      extract Dsd from the payload of forager;
      deliver Dsd to higher layers;
    }
    else if (Psd is a scout) {
      for each(SFj gathered by Psd) {
        create a forager Fsj foreach SFj;
        danceFloor.addForager(Fsj);
      }
      kill Psd;
    }
    for each(Fsj add to dance floor) {
      var packers = getNumberPacketInQueueForForager(Fsj);
      while (packers > 0 && Fsj.danceNumber > 0) {
        encapsulate Dsj into payload of Fsj;
        send Fsj to entrance;
```

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS

```
        decrease packers;
      }
    }
  }

/* check scout return */
if (timer expired and scout S_sd not returned) {
  compute newTTL of S_sd and assign it a newID;
  set timer of S_sd;
  send it to entrance;
}
```

Entrance Floor: it manages the foragers and the scouts coming from the MAC layer. 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 flight towards its source node *s* or the destination *d*. If is coming back 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 node *i* it will arrive, it will call 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 not 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 be not discarded, but returned to the source in order to update the Local UDDI registry with all *SF* description collected along the taken path.

Algorithm 3: Actions taken at Entrance Floor.

```
/* forager received from MAC layer */
for each(F_sd received from MAC layer) {
  if (F_sd arrived at d)
    send F_sd to packingFloor;
  else {
    collect optimization parameter from the node;
    sendPacketToNextHop(F_sd);
  }
}

/* scout received from MAC layer */
for each(S_sd received from MAC layer) {
  if (S_sd is on return path toward s) {
    if (S_sd is at s)
      send S_sd to packingFloor;
    else {
      sendPacketToNextHop(S_sd);
    }
  }
  else if (S_sd is on forward path toward d) {
    var   F_id = danceFloor.getForager(S_sd.SF);
    if ( F_id != NULL) {
      insert in payload of S_sd F_id.SF;
      insert in payload of S_sd other foragers in danceFloor;
      complete the route by concatenating S_si + F_id;
      change S_sd to a backward scout;
      reverse the source route in the header;
      sendPacketToNextHop(S_sd);
    }
    else {
      if (TTL expired || S_sd exists in seenScoutList)
        if (SF list collected from S_sd is empty)
          kill S_sd;
        else {
          insert in payload of S_sd foragers from danceFloor;
          change S_sd to a backward scout;
          reverse the source route in the header;
          sendPacketToNextHop(S_sd);
        }
      else {
        insert in payload of S_sd foragers from danceFloor;
        insert address i in the source route header;
        insert ID and source of S_sd into seenScoutList;
        decrease TTL;
        broadcast S_sd to all neighbors of i;
      }
    }
  }
}

/* forager received from packing floor */
for each(F_sd received from Packing floor) {
  collect optimization parameter from the node;
  sendPacketToNextHop(F_sd);
}

/* scout received from packing floor */
for each(S_sd received from Packing floor) {
  insert address i in the source route header;
  insert ID and source of S_sd into seenScoutList;
  decrease TTL;
  broadcast S_sd to all neighbors of i;
}

void sendPacketToNextHop(P_sd)
{
  find h_next in the source route header;
  send P_sd to MAC Interface of h_next;
}
```

In Fig. 2 we show a concise representation of the *BeeAdHocServiceDiscovery* working principles, with all steps required to make the final SOAP service invocation.

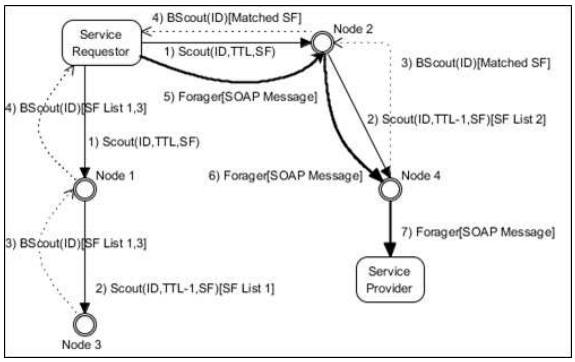

Figure 2: *BeeAdHocService Discovery* working principles.

5 CONCLUSIONS

BeeAdHocService Discovery is a new protocol of service discovery and selection for MANET based on the foraging behaviour of honeybees that totally benefits of results discussed in (Saleem et al., 2008). It uses a cross layer mechanisms that allows gathering routing information, such as path breaks and updates, in order to minimize the number of control messages and the node energy consumption with interesting advantages for the Web service accuracy and the network load balancing. *BeeAdHocService Discovery* maps the key concept of the MANET auto-configuration 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 auto-organization of it. The next step in the development of *BeeAdHocService* Discovery will be the extension of the Web Service selection criterions that should include performance parameters, such as CPU load, available RAM memory, server workload and so on (Grieco et al., 2005; Grieco et al., 2006a; Grieco et al., 2006b). Both the energy and privacy constraints (Malandrino et al., 2013; Malandrino and Scarano, 2013; D'Ambrosio et al., 2014) will be also taken into account. Performance and simulation experiments will be performed accordingly. Eventually, different forms of swarms might be exploited.

REFERENCES

Bonabeau, E., Dorigo, M., and Theraulaz, G. (1999). *Swarm intelligence: from natural to artificial systems*. Oxford University Press, Inc.

Chen, Y. and Mi, Z. (2007). A Novel Service Discovery Mechanism in MANET Using Auto-Configured SDA. In *WiCom 2007.*, pages 1660–1663.

Choudhury, P., Sarkar, A., and Debnath, N. (2011). Deployment of Service Oriented architecture in MANET: A research roadmap. In *INDIN*, pages 666–670.

D'Ambrosio, S., De Pasquale, S., Iannone, G., Malandrino, D., Negro, A., Patimo, G., Petta, A., Scarano, V., Serra, L., and Spinelli, R. (2014). Phone batteries draining: is GWeB (Green Web Browsing) the solution? In *2014 International Green Computing Conference*, IGCC'14. Dallas, Texas, USA.

De Santis, F. (2012). An Efficient Bee-inspired Autoconfiguration Algorithm for Mobile Ad Hoc Networks. *International Journal of Computer Applications*, 57(17):9–14.

Dorigo, M. and Blum, C. (2005). Ant Colony Optimization Theory: A Survey. *Theor. Comput. Sci.*, 344(2-3):243–278.

Grieco, R., Malandrino, D., Mazzoni, F., and Riboni, D. (2006a). Context-aware provision of advanced Internet services. In *Fourth Annual IEEE International Conference on Pervasive Computing and Communications Workshops, 2006.*, pages 4 pp.–603.

Grieco, R., Malandrino, D., and Scarano, V. (2005). SEcS: Scalable Edge-computing Services. SAC '05, pages 1709–1713.

Grieco, R., Malandrino, D., and Scarano, V. (2006b). A scalable cluster-based infrastructure for edge-computing services. *World Wide Web*, 9(3):317–341.

Malandrino, D., Petta, A., Scarano, V., Serra, L., Spinelli, R., and Krishnamurthy, B. (2013). Privacy Awareness About Information Leakage: Who Knows What About Me? WPES '13, pages 279–284.

Malandrino, D. and Scarano, V. (2013). Privacy leakage on the web: Diffusion and countermeasures. *Computer Networks*, 57(14):2833 – 2855.

Nesargi, S. and Prakash, R. (2002). MANETconf: Configuration of Hosts in a Mobile. In *INFOCOM*, pages 1059–1068.

Pariselvam, S. and Parvathi, R. (2012). Swarm Intelligence Based Service Discovery Architecture for Mobile Ad Hoc Networks. In *Europ. Jour. Scient. Res.*, volume 74, pages 205–216.

Royer, E. and Toh, C.-K. (1999). A review of current routing protocols for ad hoc mobile wireless networks. *Personal Communications, IEEE*, 6(2):46–55.

Saleem, M., Khayam, S., and Farooq, M. (2008). Formal Modeling of BeeAdHoc: A Bio-inspired Mobile Ad Hoc Network Routing Protocol. In *Ant Colony Optimization and Swarm Intelligence*, volume 5217, pages 315–322.

Shao, X., Ngoh, L. H., Lee, T. K., Chai, T., Zhou, L., and Teo, J. (2009). Multipath cross-layer service discovery (MCSD) for mobile ad hoc networks. In *APSCC 2009*, pages 408–413.

Teodorovic, D., Lucic, P., Markovic, G., and Dell' Orco, M. (2006). Bee Colony Optimization: Principles and Applications. In *NEUREL 2006*, pages 151–156.

Tsai, H.-W., Chen, T.-S., and Chu, C.-P. (2009). Service Discovery in Mobile Ad Hoc Networks Based on Grid. *Vehicular Technology, IEEE Transactions on*, 58(3):1528–1545.

Varshavsky, A., Reid, B., and de Lara, E. (2005). A cross-layer approach to service discovery and selection in MANETs. In *Mobile Adhoc and Sensor Systems Conference, 2005.* , pages 8 pp.–466.

Ververidis, C. and Polyzos, G. (2005). Extended ZRP: a routing layer based service discovery protocol for mobile ad hoc networks. In *Mobile and Ubiquitous Systems: Networking and Services, 2005*, pages 65–72.

Wedde, H. and Farooq, M. (2005a). BeeHive: Routing Algorithms Inspired by Honey Bee Behavior. *KI*, 19(4):18–24.

Wedde, H. and Farooq, M. (2005b). The wisdom of the hive applied to mobile ad-hoc networks. In *SIS 2005*, pages 341–348.

Wedde, Horst F. *at al*. (2005). BeeAdHoc: An Energy Efficient Routing Algorithm for Mobile Ad Hoc Networks Inspired by Bee Behavior. GECCO '05, pages 153–160.

SCITEPRESS SCIENCE AND TECHNOLOGY PUBLICATIONS