



Edinburgh  
2009

PerAda-SICSA Summer School

## Nature Knows Best?

Case study led by

Ruth Falconer - SIMBIOS Centre, University of Abertay &  
Emma Hart - Centre for Emergent Computing, Edinburgh Napier University

Notes written by

Despina Davoudani - School of Computing, Edinburgh Napier University

## Overview

In Wireless Sensor Networks (WSNs), routing models are typically used for data gathering, i.e. a sink  $s$  collects sensor information from data sources. In the extended version of [SpeckSim](#) you have installed, two routing algorithms are implemented which are based on:

- the *UDG-NNT* algorithm [1], (Unit Disk Graph<sup>1</sup>-Nearest Neighbour Tree), an approximation of minimum spanning trees for data aggregation in static, multi-hop WSNs, and
- the *AdHocNet* algorithm [2], an adaptive routing algorithm for mobile ad hoc networks (MANETs) inspired by ideas from Ant Colony Optimisation (ACO).

The simulator offers a unit cube which can be populated with different types of specks. A speck is essentially a network device. The default components offered by [SpeckSim](#) are a radio unit and a battery. A speck is extensible though, so more components may be added by implementing the provided interfaces. More details on how to extend [SpeckSim](#) can be found [here](#).<sup>2</sup>

For the purposes of implementing the aforementioned routing algorithms—which will be referred to simply as *NNT* (based on UDG-NNT) and *ACO* (based on AntHocNet)—four types of specks were created, a pair of *sink*  $s$  and *source*  $i$  for each algorithm:

- NNTSink & NNTSource for NNT, and
- ACOSink & ACOSource for ACO

All specks have distinct IDs from a totally ordered set  $[1..N]$ , where  $N$  is the the number of all types of specks in the network.

To create a network for testing a routing algorithm, populate the unit cube with **exactly one** speck of sink type and **at least one** speck of source type. The current implementation does not support more than one sinks in the network. The reason is mainly due the fact that UDG-NNT assumes that one special node, the sink  $s$ , is designated to be the root of the tree.

The following sections discuss the routing algorithms in more detail.

## NNT Algorithm

The UDG-NNT is a *proactive* routing algorithm, in the sense that it provides routing information which can be used at any time to forward data through the network. However, the data are not communicated between any two nodes in the network, rather they are forwarded towards a specific destination, the sink. The UDG-NNT algorithm is executed in two phases:

- A. The source nodes choose their ranks randomly; this phase is initiated by sink  $s$ .
- B. Each node, except  $s$ , creates an edge in the NNT with the nearest node among its neighbours.

The algorithm itself (as presented in [1]) doesn't provide any maintenance mechanism to accommodate addition or deletion of nodes to/from the tree. (The authors present additional distributed algorithms, but these apply to a wireless network modelled as a

---

<sup>1</sup> In the Unit Disk Graph (UDG) wireless network model, two nodes  $u$  and  $v$  communicate directly if and only if  $d(u, v) \leq R$  for some given  $R$  [1]. The UDG is a popular graph model for multi-hop wireless networks.

<sup>2</sup> The main URL for [SpeckSim](http://www.specknet.org/dev/specksim) is: <http://www.specknet.org/dev/specksim>

*complete graph* which assumes that any two nodes can communicate *directly* with each other, a situation not common in WSNs).

One solution to maintain a dynamic tree is to perform periodic updates in order to refresh the NNT. The cost of keeping the routing information up-to-date (which depends on the amount of mobility within the network, how often nodes fail etc.) is counterbalanced by the fact that data can be forwarded at any time to the sink.

#### Techniques for maintaining the tree include:

- Recreation of the NNT at regular time intervals; e.g. execution of the UDG-NNT algorithm at a certain frequency.
- Circulation of periodic acknowledgement packets between the nodes within the tree; e.g. nodes a & b have set up a link between them & ping each other periodically. If one fails to reply, then the other sets up a connection with a neighbouring node that has the next best rank in its list.
- Beyond the initial set-up of the tree, NNT-related info can be built into data messages, so that the routing information is being updated as data are communicated within the network.

The NNT variation implemented in SpeckSim, performs only the first phase of the UDG-NNT algorithm and incorporates a maintenance mechanism which updates the tree by simply repeating phase A regularly.

#### NNT Algorithm

Assume special node **sink s** is the **root** of NNT and *i* is any other node in the network.

1. Sink *s* broadcasts a **RANK message** with its ID and **rank 0**.
2. Upon **reception** of the sink's RANK<sub>*s*</sub> message, node *i*:
  - stores** locally the information contained in the RANK<sub>*s*</sub> message,
  - calculates** its own rank number,
  - broadcasts** a new RANK message with its own ID and rank,
  - enters **non-reply mode** for time **t<sub>u</sub>**.
3. Upon reception of **subsequent** RANK<sub>*i*</sub> messages:
  - sink s** does nothing,
  - node n**:
    - if* it is in non-reply mode, does nothing,
    - otherwise* it repeats step 2.
4. Sink *s* repeats step 1 periodically every **t<sub>u</sub>** time units.

At some point, every node *i* receives a RANK message from at least one of its neighbour, if the given UDG is connected.

## ACO Algorithm

The AntHocNet [2] is a hybrid algorithm that combines two processes:

- *reactive* route setup (see [2, sec. 4.2.2]); is carried out to gather information about destinations that are involved in communication sessions.
- *proactive* route maintenance (see [2, sec. 4.2.3]); maintains, improves and extends information about existing paths while the communication session is going on.

A node  $i$  organises its routing information in a *pheromone table*  $\mathbb{T}_i$  (see [2, sec. 4.2.1]), which is a two-dimensional matrix. Forwarding of control and data messages is done stochastically, using this table. In addition to a pheromone table, a node stores a *neighbour table*  $\mathbb{N}_i$  used to detect link failures. Link failures ([2, sec. 4.2.5]) are dealt with reactive mechanisms, such as local route repair and the use of warning messages.

### Pheromone Table

An entry  $\tau_{d_{ij}}$  of a pheromone table  $\mathbb{T}_i$  contains information about the route from node  $i$  to destination  $d$  over neighbour  $j$ . This includes:

- a regular pheromone value  $\tau_{d_{ij}}$ , indicating the relative goodness of going over node  $j$  when travelling from node  $i$  to destination  $d$ ,
- a virtual pheromone value  $\omega_{d_{ij}}$ , indicating an alternative estimate of the goodness of the route from  $i$  to  $d$  over  $j$ , and
- an average number of hops  $h_{d_{ij}}$ .

### Neighbour Table

An entry  $N_{ij}$  of a neighbour table corresponds to  $i$ 's neighbour  $j$  and contains a time value  $t_{h_{ij}}$  indicating when  $i$  last heard from  $j$ .

### Routing Metrics

The goodness of a route (i.e. pheromone level) is expressed as the inverse of a cost. Exact values depend on the metric that is used to evaluate the cost of the routes. Routing metric examples include:

- the number of hops
- the end-to-end delay
- a combination of hops and end-to-end delay, and
- a metric based on the signal-to-interference-and-noise ratio of links along the route.

For further details, see [2, sec. 4.2.6].

The ACO routing algorithm implemented in SpeckSim, assumes there is only **one** destination  $d$  (i.e. sink) in the network, and performs the two main processes of AntHocNet:

- the reactive route setup – is triggered whenever a source node generates a new `DATA` message for a destination for which no routing information is available. It involves the sending of a *reactive forward ant* (RFA) from source to destination, and a *reactive backward ant* (RBA) from destination to source.
- the proactive route maintenance – allows to build a mesh of multiple routes around the initial route created during the reactive route process. It consists of two subprocesses:
  - *pheromone diffusion* (see [2, p. 91]), which is aimed at spreading available pheromone information over the network, through the use of periodic update messages and information bootstrapping. This subprocess is executed by all nodes throughout their whole lifetime.
  - *proactive ant sampling* (see [2, p. 94]), which is aimed at controlling and updating pheromone information, through path sampling using *proactive forward ants* (PFA) from source to destination, *proactive backward ants* (PBA) from destination to source. These ants are released by a source node of a communication session, at the start of the session, for as long as the session is going on.

Although the above ACO implementation includes the formation of neighbourhoods, using `HELLO` messages which carry pheromone information in pheromone diffusion process, it does not handle link failures. The `HELLO` messages, however, can easily be used for detecting link failures which, in turn, enable implementation of the link failures mechanisms:

- *link failure notifications* (see [2, p. 98]),
- *local route repair* (see [2, p. 98]), and
- *unicast of WARNING messages* ((see [2, p. 99])).

---

**ACO Algorithm: Reactive Route Setup**

Only one route is set up, so that the source node has exactly one outgoing next hop for the involved destination.

1. **Source i:**  
 starts a communication session with destination *d*, i.e.  
**generates** a new **DATA** message for *d*:  
     *if* it has routing info for *d*, **unicasts** the DATA message  
         stochastically using formula 4.6 [2],  
     *otherwise* it **broadcasts** a new **RFA** for *d*.
  2. Upon **reception** of a **DATA** message, each **intermediate node:**  
     *if* it has routing info for *d*, **forwards** the DATA message  
         stochastically using formula 4.6 [2],  
     *otherwise* it **discards** the DATA message.
  3. Upon **reception** of a **RFA**, each **intermediate node:**  
     **updates** the list of visited hops in the RFA,  
     *if* it has routing info for *d*, **unicasts** the RFA stochastically  
         using formula 4.1 [2],  
     *otherwise* it **broadcasts** only the **first copy** of the RFA it sees.
  4. Upon **reception** of a **RFA**, **destination d:**  
     **converts** the RFA to RBA,  
     **unicasts** the **RBA** back to source.
  5. Upon **reception** of a **RBA**, each **intermediate node:**  
     **updates** the **RBA** with the cost so far,  
     **updates** its **routing table** (the average number of hops & regular  
         pheromone, using formulas 4.2 & 4.3 [2]),  
     **unicasts** the RBA to the next hop in the included path.
  6. Upon **reception** of a **RBA**, **source i:**  
     **updates** its routing table (the average number of hops & regular  
         pheromone, using formulas 4.2 & 4.3 [2]).
- 

**ACO Algorithm: Pheromone Diffusion**

Update virtual pheromone and, when it is reliable enough (for conditions see [2, p. 94]), update regular pheromone as well.

1. **Node i:**  
     **broadcasts** a **HELLO** message every **t\_hello** time units.
  2. Upon **reception** of a **HELLO** message, **node j:**  
     **updates** its neighbour table,  
     **updates** its routing table (the virtual pheromone, and  
         *conditionally* the regular pheromone, using formula 4.4 [2]).
-

---

**ACO Algorithm: Proactive Ant Sampling**

Update goodness estimates of existing routes, and find new routes based on the hints provided by the pheromone diffusion process.

**1. Source i:**

after sending a new DATA message for d:  
if its best virtual pheromone is significantly better  
than its best regular pheromone:  
unicasts a new PFA message for d stochastically  
using formula 4.5 [2].

**3. Upon reception of a PFA, each intermediate node:**

updates the list of visited hops in the PFA,  
if it has routing info for d, unicasts the PFA stochastically  
using formula 4.5 [2],  
otherwise it discards the PFA.

**4. Upon reception of a PFA, destination d:**

converts the PFA to PBA,  
unicasts the PBA back to source.

**5. Upon reception of a PBA, each intermediate node:**

updates the PBA with the cost so far,  
updates its routing table (the average number of hops & regular  
pheromone, using formulas 4.2 & 4.3 [2]),  
unicasts the PBA to the next hop in the included path.

**6. Upon reception of a PBA, source i:**

updates its routing table (the average number of hops & regular  
pheromone, using formulas 4.2 & 4.3 [2]).

---

## References

- [1] Khan M., Pandurangan G., and Kumar V.S.A., "Distributed Algorithms for Constructing Approximate Minimum Spanning Trees in Wireless Sensor Networks", IEEE Transactions on Parallel and Distributed Systems, vol. 20, no. 1, pp. 124-139, 2009 ([web](#); [pdf](#); [bibtex](#))
- [2] Ducatelle, F., "Adaptive Routing in Ad Hoc Wireless Multi-hop Networks", PhD thesis, Università della Svizzera Italiana, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, 2007 ([web](#); [pdf](#); [bibtex](#))

## BibTeX Citations

```
@article{KhanMetal2009,  
  author = {Maleq Khan and Gopal Pandurangan and V.S. Anil Kumar},  
  title = {Distributed Algorithms for Constructing Approximate Minimum Spanning Trees in  
Wireless Sensor Networks},  
  journal = {IEEE Transactions on Parallel and Distributed Systems},  
  volume = {20},  
  number = {1},  
  issn = {1045-9219},  
  year = {2009},  
  pages = {124--139},  
  doi = {http://doi.ieeecomputersociety.org/10.1109/TPDS.2008.57},  
  publisher = {IEEE Computer Society},  
  address = {Los Alamitos, CA, USA},  
}
```

```
@phdthesis{DucatelleF2007,  
  author = {Frederick Ducatelle},  
  title = {Adaptive Routing in Ad Hoc Wireless Multi-hop Networks},  
  school = {Università della Svizzera Italiana, Istituto Dalle Molle di Studi sull'Intelligenza  
Artificiale},  
  year = {2007},  
}
```