Chapter 4

AntHocNet: an adaptive routing algorithm for ad hoc wireless multi-hop networks

In this chapter, we describe AntHocNet, an adaptive routing algorithm for AHWMNs. AntHocNet is a hybrid routing algorithm, in the sense that it contains elements from both reactive and proactive routing. Specifically, it combines a reactive route setup process with a proactive route maintenance and improvement process. AntHocNet was in the first place inspired by the ACO approach to routing. This is evident in the way that it gathers, stores and uses routing information. Consequently, the terminology used in this chapter is mostly related to the ACO routing literature. Nevertheless, AntHocNet also contains elements from distance vector routing. In particular, the information gathering process used in its proactive route maintenance and improvement process combines the route sampling strategy from ACO routing with an information bootstrapping process that is similar to the one used in distance vector routing algorithms. The way both approaches are combined is novel and allows the algorithm to get the best of both worlds. AntHocNet was in the first place developed for MANETs and WMNs. Descriptions of the AntHocNet algorithm have been published in [73–75, 94, 95, 98].

The rest of this chapter is organized as follows. First, we give a general overview of the AntHocNet routing algorithm. This includes a high level description of the algorithm in words, and a schematic representation. Then, we give a detailed description of each of the algorithm's components. Finally, we provide further discussions on the presented material, such as an investigation of the relations between AntHocNet and other AHWMN routing algorithms, a discussion of how AntHocNet relates to the RL solution methods presented

in chapter 3, and a description of elements that were present in older versions of AntHocNet. Evaluations of AntHocNet and experimental comparisons with other routing algorithms will be provided in the following chapters.

4.1 General overview of the AntHocNet routing algorithm

In this section, we give an overview of the algorithm. We start with a general description in words. Then, we also provide a schematic representation in the form of a finite state machine.

4.1.1 Algorithm description

AntHocNet is a hybrid algorithm, containing both reactive and proactive elements. The algorithm is reactive in the sense that it only gathers routing information about destinations that are involved in communication sessions. It is proactive in the sense that it tries to maintain and improve information about existing paths while the communication session is going on (unlike purely reactive algorithms, which do not search for routing information until the currently known routes are no longer valid). Routing information is stored in pheromone tables that are similar to the ones used in other ACO routing algorithms. Forwarding of control and data packets is done in a stochastic way, using these tables. Link failures are dealt with using specific reactive mechanisms, such as local route repair and the use of warning messages. Below, we describe the general working of the AntHocNet routing algorithm. Details will follow later in section 4.2.

In AntHocNet, routing information is organized in pheromone tables, similar to the ones used in other ACO routing algorithms such as the earlier described AntNet (see subsection 3.2.3). Each node i maintains one pheromone table $\mathcal{T}_i$, which is a two-dimensional matrix. An entry $\mathcal{T}_{ij}^d$ of this pheromone table contains information about the route from node i to destination d over neighbor j. This information includes the pheromone value τ_{ij}^d, which is a value indicating the relative goodness of going over node j when traveling from node i to destination d, as well as statistics information about the path, and possibly virtual pheromone (see later). Apart from a pheromone table, each node also maintains a neighbor table, in which it keeps track of which nodes it has a wireless link to. Details about the data structures maintained under AntHocNet are described in subsection 4.2.1.

At the start of a communication session, the source node of the session controls its pheromone table, to see whether it has any routing information available for the requested destination. If it does not, it starts a reactive route setup process, in which it sends an ant packet out over the network to find a route to the destination. Such an ant packet is called a reactive forward ant. Each intermediate node receiving a copy of the reactive forward ant forwards it. This is done via unicasting in case the node has routing information about the ant's

destination in its pheromone table, and via broadcasting otherwise. Reactive forward ants store the full array of nodes that they have visited on their way to the destination. The first copy of the reactive forward ant to reach the destination is converted into a reactive backward ant, while subsequent copies are destroyed. The reactive backward ant retraces the exact path that was followed by the forward ant back to the source. On its way, it collects quality information about each of the links of the path. At each intermediate node and at the source, it updates the routing tables based on this quality information. This way, a first route between source and destination is established at completion of the reactive route setup process. The full process is repeated later if the source node falls without valid routing information for the destination of the session while data still need to be sent. Details about the reactive route setup process are provided in subsection 4.2.2.

Once the first route is constructed via the reactive route setup process, the algorithm starts the execution of the proactive route maintenance process, in which it tries to update, extend and improve the available routing information. This process runs for as long as the communication session is going on. It consists of two different subprocesses: pheromone diffusion and proactive ant sampling. The aim of the pheromone diffusion subprocess is to spread out pheromone information that was placed by the ants. Nodes periodically broadcast messages containing the best pheromone information they have available. Using information bootstrapping, neighboring nodes can then derive new pheromone for themselves and further forward it in their own periodic broadcasts. Details about this process will be given later, in subsection 4.2.3. Here, it is sufficient to know that the pheromone diffusion process is similar to the dynamic programming approach used in distance vector routing. As was pointed out earlier in subsections 3.1.1 and 3.3.2, such approaches to gathering routing information are very efficient, but can be slow to adapt to dynamic situations, possibly temporarily providing erroneous information. Therefore, the pheromone diffusion process can be considered as a cheap but potentially unreliable way of spreading pheromone information. Because of this potential unreliability, the pheromone that is obtained via pheromone diffusion is kept separate from the normal pheromone placed by the ants, and is called virtual pheromone; the pheromone placed by the ants will in what follows be called regular pheromone. The virtual pheromone is used to support the second subprocess of proactive route maintenance, which is proactive ant sampling. In this subprocess, all nodes that are the source of a communication session periodically send out proactive forward ants towards the destination of the session. These ants construct a path in a stochastic way, choosing a new next hop probabilistically at each intermediate node. Different from reactive forward ants, they are never broadcast. When calculating the probability of taking a next hop, proactive forward ants consider both regular and virtual pheromone. This way, they can leave the routes that were followed by previous ants, and follow the (potentially unreliable) routes that have emerged from pheromone diffusion. Once a proactive forward ant reaches the destination, it is converted into a proactive backward ant that travels back to the source and leaves pheromone along the

way (regular, not virtual pheromone), just like reactive backward ants. This way, proactive ants can follow virtual pheromone and then, once they have experienced that it leads to the destination, convert it into regular pheromone. One could say that pheromone diffusion suggests new paths and that proactive ants check them out. The ant based full path sampling provides the reliability that is lacking in the efficient information bootstrapping process. Details about the proactive route maintenance process are given in subsection 4.2.3.

Data packet forwarding in AntHocNet is done similarly to other ACO routing algorithms: routing decisions are taken hop-by-hop, based on the locally available pheromone. Only regular pheromone is considered, as virtual pheromone is not considered reliable enough. Each forwarding decision is taken using a stochastic formula that gives preference to next hops that are associated with higher pheromone values. The formula is different from that used by the forward ants, so that data packets can follow a less exploratory strategy. Via parameter tuning, it is possible to vary between spreading the data packets over all possible available paths and deterministically sending them over the best path. While the former can in principle provide higher throughput through the use of multiple paths (see subsection 2.4.3), the latter allows greedy exploitation of the learned information. Later, in chapter 5, we compare both strategies empirically. Details about data packet forwarding in AntHocNet are provided in subsection 4.2.4.

Link failures can be detected in AntHocNet via failed transmissions of data or control packets, or through the use of hello messages. Hello messages are short messages that are periodically sent out by all nodes in the network. The reception of a hello message is indicative of the presence of a wireless link, while the failure to receive such messages point to the absence of a link. In practice in AntHocNet, the function of hello messages is fulfilled by the same periodic messages that are used for pheromone diffusion. When a node detects a link failure, it controls its pheromone table, to see which routes become invalid due to the failure, and whether alternative routes are available for the affected destinations. Then, it broadcasts a link failure notification message to warn neighboring nodes about all relevant changes in its pheromone table. In case the link failure was associated with a failed data packet transmission, the node can also start a local route repair to restore the route to the destination of this data packet. To this end, it sends out a repair forward ant. Repair forward ants are similar to reactive forward ants, in the sense that they follow available pheromone information where possible, and are broadcast otherwise, but they have a limited maximum number of broadcasts, so that they cannot travel far from the old failed route. Upon arrival at the destination, the repair forward ant is converted into a repair backward ant that travels back to the node that started the repair process and sets up the pheromone for the repaired route. A last tool in dealing with link failures is the use of unicast warning messages. These are needed when data packets for a lost destination still arrive at the node after a link failure notification has already been broadcast. This can be due to bad reception of the broadcast notification message. In this case, the node unicasts a warning to the node it received the data from, in order to inform

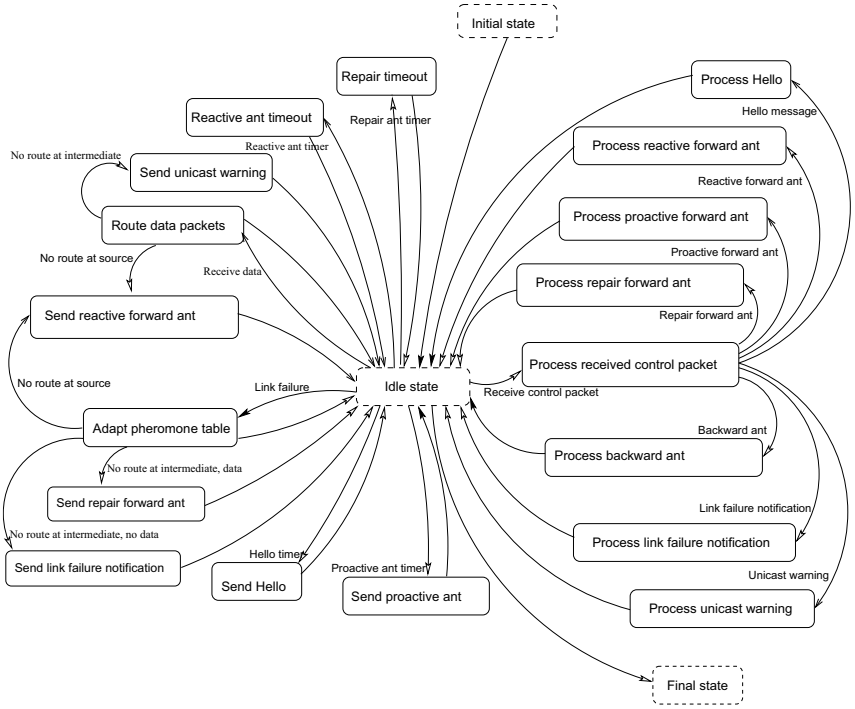

Figure 4.1: A finite state machine representation of the AntHocNet routing algorithm.

it that it can no longer forward data for this destination. Details about how AntHocNet deals with link failures are described in subsection 4.2.5.

4.1.2 Schematic representation

Figure 4.1 gives a schematic representation of the AntHocNet routing algorithm. It contains a finite state machine showing the most important components of the algorithm. Below we explain the structure of the finite state machine. For details about each of the components, we refer to section 4.2.

The algorithm is started up in its initial state, in which internal variables are initialized. Then, it moves to its idle state, where it waits for events to happen. The events that can take place are the arrival of a data packet, the arrival of a control packet, the detection of a link failure, and a number of timer events.

In case of a data reception event, the algorithm tries to route the data. Details about data forwarding are given in subsection 4.2.4. This might be impossible, due to the lack of routing information about the destination. If the current node is the source node of the data packet, the unavailability of routing

information can be because the data packet is the first of a new communication session, or because it belongs to a session for which all routing information has become invalid. In both cases, the node starts a reactive route setup process. Details about this process are given in subsection 4.2.2. If the current node is not the source of the data packet, it concludes that the upstream node of the data has wrong routing information, and sends it a unicast warning message. This is further explained in subsection 4.2.5, which talks about link failures.

In case of a control packet reception event, the algorithm checks which type of control packet it is dealing with. In case it is a hello message, the node needs to take note in its neighbor table that it has a wireless link with the packet's sender, and it needs to extract the routing information inside to update its own virtual pheromone information. This is part of the proactive route maintenance process described in subsection 4.2.3. In case it is a reactive, proactive or repair forward ant, the node needs to execute the correct forwarding action, or, if it is the final destination of the ant, create a backward ant and send it back towards the source. In case the control packet is a backward ant (reactive, proactive and repair backward ants have essentially the same behavior, and are therefore collapsed here), the node needs to adapt its pheromone table, and forward the ant if it is not its final destination. Details about the treatment of reactive, proactive and repair forward and backward ants are given respectively in subsections 4.2.2, 4.2.3 and 4.2.5. In case the control packet is a link failure notification, the node needs to update its pheromone table, and possibly forward the notification. Finally, if it is a unicast warning, it needs to update its pheromone table, removing the erroneous route. Details about link failure notifications and unicast warnings are given in subsection 4.2.5.

In case of a link failure event, the node first of all adapts the information in its pheromone table to reflect the changed situation. Then, if destinations have become unreachable due to the link failure, it needs to take action. If the current node is the source of a session to one of the lost destinations, it starts a reactive route setup process. If, on the other hand, it is an intermediate node on the lost route, it controls whether the link failure event involved the unsuccessful transmission of a data packet. If this is the case, it starts a local route repair process. Otherwise, it broadcasts a link failure notification message. Details on how link failures are dealt with are given in subsection 4.2.5.

The different timers are events that are scheduled by the node itself, in order to plan delayed actions. Hello timer events are scheduled at regular intervals, from the moment the node is switched on and for as long as it is up and running. Reception of a hello timer event provokes the node to send a hello message, in which it includes its best pheromone information. This is part of the proactive route maintenance process described in subsection 4.2.3. Proactive ant timer events are also scheduled at regular intervals, but only from the moment a session is started, and until the end of it. Reception of a proactive ant timer event leads the node to send out a proactive forward ant. Details about this are given in subsection 4.2.3. Repair timer events are scheduled after a repair forward ant has been sent out. At reception of a repair timer event, the node checks whether a repair backward ant was received, and in case not, it broadcasts a

link failure notification. This is part of the process of dealing with link failures, described in 4.2.5. Finally, reactive ant timer events are similarly sent out after the transmission of a reactive forward ant. At reception of a reactive ant timer event, the node controls whether it has already received a reactive backward ant. In case not, it can send out a new reactive forward ant (in case the maximum number of retransmissions has not yet been reached), or conclude that the destination is currently unreachable, and drop queued data packets for it. This is described in detail in subsection 4.2.2, which explains the reactive route setup phase.

4.2 Detailed descriptions

In this section, we give a detailed description of the different components of the AntHocNet routing algorithm. We follow the same structure as in subsection 4.1.1. First we describe the data structures that are maintained in each node. Then, we give details about the reactive route setup process. Subsequently, we discuss the proactive route maintenance process. Next, we talk about data packet forwarding. After that, we discuss the algorithm's behavior with respect to link failures. Finally, we talk about the routing metrics used in AntHocNet.

4.2.1 Data structures in AntHocNet

Here, we describe the different data structures that are maintained by each of the network nodes under AntHocNet. In particular, we talk about pheromone tables and neighbor tables.

Pheromone tables

Under AntHocNet, each node i maintains a pheromone table $\mathcal{T}_i$, which is a two-dimensional matrix. An entry $\mathcal{T}_{ij}^d$ of this matrix contains information about the route from node i to destination d over neighbor j. This includes a regular pheromone value τ_{ij}^d, a virtual pheromone value ω_{ij}^d, and an average number of hops h_{ij}^d. The regular pheromone value τ_{ij}^d is an estimate of the goodness of the route from i to d over j. Goodness is expressed as the inverse of a cost. Exact values depend on the metric that is used to evaluate the cost of routes. More about the use of different metrics will follow in subsection 4.2.6. Regular pheromone is updated by backward ants. These can be reactive, proactive or repair backward ants. Details about the updating process for regular pheromone are given in subsection 4.2.2. The virtual pheromone value ω_{ij}^d forms an alternative estimate of the goodness of the route from i to d over j. Differently from τ_{ij}^d, it is obtained through information bootstrapping using goodness values reported by neighbor nodes during the proactive route maintenance process. The updating of virtual pheromone is discussed in subsection 4.2.3. The average number of hops h_{ij}^d is, like the regular pheromone, updated by backward ants.

This updating process is described in subsection 4.2.2. h_{ij}^d is used when deciding how long to wait for repair backward ants (see subsection 4.2.5).

Nodes do not necessarily always have values available for τ_{ij}^d, ω_{ij}^d, and h_{ij}^d for each possible combination of destination and next hop. This is in the first place because nodes do not maintain routing information about all possible destinations in the network (they only gather routing information for destinations which communication sessions are going on with), and because for a specific destination, nodes do not necessarily have a route available over each of their possible neighbors (for instance, during a reactive route setup phase only one route is set up, so that the source node has exactly one outgoing next hop for the involved destination). Also, since regular and virtual pheromone are obtained through different processes, it is possible that a node has a value for τ_{ij}^d, but not for ω_{ij}^d, or vice versa. On the other hand, since τ_{ij}^d and h_{ij}^d are both obtained from backward ants, nodes that have a value for one of the two will also have a value for the other.

Neighbor tables

Apart from the pheromone table, each node also maintains a neighbor table. The neighbor table $\mathcal{N}_i$ kept by node i is a one-dimensional vector with one entry for each of i's neighbors. The entry $\mathcal{N}_{ij}$ corresponding to i's neighbor j contains a time value th_{ij} indicating when i last heard from j. Node i uses this time value to derive whether there is a wireless link with node j, and to detect link failures. More on the detection and handling of link failures will follow later in subsection 4.2.5.

4.2.2 Reactive route setup

The reactive route setup process is triggered whenever a node receives a data packet that was locally generated (i.e., the current node is the packet's source) for a destination for which no routing information is available. This lack of routing information can happen either because the data packet in question is the first of a new communication session, and no routing information for its destination is available from a different or previous session, or because the data packet belongs to an ongoing session for which all routes have become invalid (e.g., due to node movements). The reactive route setup process involves the sending of a reactive forward ant from source to destination, and a reactive backward ant from destination to source. Below, we discuss each of these separately.

Reactive forward ants

At the start of the reactive route setup process, the source node s creates a reactive forward ant. This is a control packet that has as a goal to find a path from s to an assigned destination d. At the start, the ant contains just the addresses of s and d. Later, as it proceeds through the network, it collects a list $\mathcal{P} = [1, 2, \ldots, d-1]$ of all the nodes that it has visited on its way from s to d.

After its creation at s, the reactive forward ant is broadcast, so that all of s's neighbors receive a copy of it. At each subsequent node, the ant is either unicast or broadcast, depending on whether the current node has routing information for d. If routing information is available, the node chooses a next hop for the ant probabilistically, based on the different pheromone values associated with next hops for d. Concretely, a node i chooses node n as next hop for the ant with probability P_{in}^d, as calculated by equation 4.1. In this equation, N_i^d is the set of neighbors of i over which a path to d is known, and β_1 is a parameter value which can control the exploratory behavior of the ants. In our experiments, we keep β_1 relatively high, on 20. This is because we want to obtain the initial route as fast as possible, and limit the time we spend on exploration at this stage.

$$P_{in}^d = \frac{(\tau_{in}^d)^{\beta_1}}{\sum_{j \in N_i^d} (\tau_{ij}^d)^{\beta_1}}, \quad \beta_1 \geq 1, \tag{4.1}$$

In case the intermediate node i does not have routing information for d, it broadcasts the reactive forward ant. Due to this broadcasting (and also the initial broadcasting at s), a reactive forward ant can proliferate quickly over the network, with different copies of the ant following different paths to the destination. In order to limit the amount of overhead that is created this way, nodes only forward the first copy of the ant that they receive. Subsequent copies are simply discarded. In previous versions of AntHocNet, nodes were to some extent allowed to forward multiple copies of the same ant, in order to improve the creation of multiple paths (see also subsection 4.3.4). However, this lead to a lot of overhead.

At the destination, the reactive forward ant is converted into a reactive backward ant, which follows the list of nodes $\mathcal{P}$ visited by the forward ant back to s. If more than one copy of the forward ant is received, only the first is accepted and converted into a backward ant, while subsequent copies are discarded. This way, only one route is set up during the reactive route setup process. The reason is again to reduce overhead created during this procedure. For the creation of multiple routes, AntHocNet relies on the proactive route maintenance process, which extends the initially created route into a full mesh of routes during the course of the communication session (see subsection 4.2.3).

Reactive backward ants

The reactive backward ant created by the destination node in response to a reactive forward ant contains the addresses of the forward ant's source node s and destination node d, as well as the full list of nodes $\mathcal{P}$ that the forward ant has visited. The reactive backward ant is unicast from d and between the nodes of $\mathcal{P}$ back to s.

The aim of the reactive backward ant is to update routing information in each of the nodes of $\mathcal{P}$ and in s. At each node i that it visits, it updates the number of hops h_{in}^d and the regular pheromone value τ_{in}^d in the pheromone table

entry $\mathcal{T}_{in}^{d}$, where n is the node that it visited before i on its way from d. The updating of h_{in}^{d} is done using a moving average, as shown in equation 4.2. In this equation, h is the number of hops that the backward ant has traveled between d and i, and α is a parameter regulating how quickly the formula adapts to new information. In our experiments, α is always kept on 0.7.

$$h_{in}^{d} \leftarrow \alpha h_{in}^{d} + (1-\alpha)h, \quad \alpha \in [0,1] \tag{4.2}$$

Updating of the regular pheromone τ_{in}^{d} is done based on the cost of the route from i to d. This cost can be calculated using different metrics, such as the number of hops, the end-to-end delay, etc.. Later on, in subsection 4.2.6, we comment on different metrics used in AntHocNet. Here, we talk in terms of a generic cost c, where c_i^d is the cost of the route from i to d, and c_i^j is the cost of the link from i to its neighbor j (it is the cost of a one-hop route). Under AntHocNet, each node maintains a local estimate of the cost c_i^j to go to each of its neighbors j. Details about how these local estimates are calculated depend on the metric and are discussed in subsection 4.2.6. The reactive backward ant reads at each node i the local estimate c_i^n of the cost to go from i to the next hop n that the ant is coming from. It adds this cost to the total cost c_n^d of the route from n to d (which it has been calculating on its way back from d), which is stored inside the ant. The new cost c_i^d is used to update the pheromone value τ_{in}^{d} in node i, using the moving average formula of equation 4.3. In this equation, γ is a parameter regulating the speed of adaptation of the pheromone to new cost values. In our experiments, γ was kept on 0.7. The cost value c_i^d is inverted to calculate the pheromone value τ_{ij}^{d}, as pheromone indicates the goodness of a route, rather than its cost.

$$\tau_{ij}^{d} \leftarrow \gamma \tau_{ij}^{d} + (1-\gamma)(c_i^d)^{-1}, \quad \gamma \in [0,1] \tag{4.3}$$

It is interesting to note that in terms of gathering route cost information, there is an important difference here with the AntNet algorithm described in subsection 3.2.3 and other ACO routing algorithms. Rather than relying on the cost values experienced by the forward ants, AntHocNet uses the estimates c_i^j calculated locally by the nodes. This is in order to improve reliability of the measured values. Depending on the cost metric used, the high variability of the wireless medium can cause large differences between values measured by subsequent samples. For example, the delay incurred on a link can vary strongly in case of congestion if IEEE 802.11 is used as a MAC layer protocol (see subsection 2.3.3). Using local estimates that are based on more than one sample can take away some of this variability.

4.2.3 Proactive route maintenance

The proactive route maintenance process serves to update and extend available routing information. In particular, it allows to build a mesh of multiple routes around the initial route created during the reactive route setup process. The proactive route maintenance process consists of two subprocesses: pheromone

diffusion and proactive ant sampling. Pheromone diffusion is aimed at spreading available pheromone information over the network through the use of periodic update messages and information bootstrapping. Proactive ant sampling is aimed at controlling and updating pheromone information through path sampling using proactive forward ants. While proactive ant sampling is started by the source node of a communication session at the start of the session and continues for as long as the session is going on, pheromone diffusion is executed by all nodes throughout their whole lifetime, and is not particulary bound to the occurrence of a single session. Below, we describe each of the two subprocesses of proactive route maintenance separately.

Pheromone diffusion

The reactive route setup process described in subsection 4.2.2 leads to the availability of a single route from the source of a communication session to its destination, indicated by regular pheromone values in the pheromone tables of the nodes. Moreover, each neighbor node of the destination also has a one-hop route to the destination. This is independent of the running session, and is simply due to the fact that neighboring nodes are aware of each other's presence, as is explained further in subsection 4.2.5. The aim of the pheromone diffusion process is to spread all this pheromone information out, so that a field of pheromone pointing towards the destination becomes available in the network. This field of pheromone is indicated in the virtual pheromone values in the pheromone tables of the nodes. The fact that pheromone is spread out is similar to the normal diffusion of real pheromone in nature [185], which allows ants further away to sense it. In the example of figure 4.2, a communication session is going on between node 1 and node 8. Regular pheromone is indicated by solid arrows. It consists of a single route from 1 to 8 over the nodes 3, 6 and 7 that is the result of reactive route setup, and a one-hop route from 5 to 8 that is there independently of the running session, because 5 is aware of the presence of its neighbor 8. The field of virtual pheromone that is the result of pheromone diffusion is indicated with dashed arrows.

A crucial role in the pheromone diffusion process is played by hello messages. These are short messages broadcast every t_{hello} seconds asynchronously by all the nodes of the network throughout their whole lifetime. In AntHocNet, t_{hello} is set to 1 second. Hello messages are used in many existing protocols, such as ABR [257] and OLSR [61] (see subsection 2.4.2), to allow nodes to find out which are their immediate neighbors, and to detect link failures. While also in AntHocNet hello messages are used for this purpose (as is explained in subsection 4.2.5), they also serve a different goal, namely to carry information in the pheromone diffusion process. They serve as the periodic update messages that are needed in the information bootstrapping process of pheromone diffusion. The idea to convey extra routing information inside hello messages has been used in some other routing algorithms, such as the earlier mentioned OLSR.

Nodes include in each hello message that they send out routing information they have available. In particular, a node i constructing a hello message consults

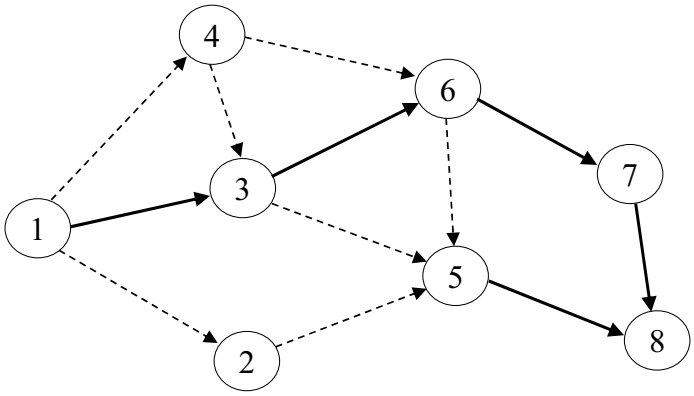

Figure 4.2: An example of available pheromone in an AHWMN. Node 1 is running a communication session with node 8 as destination. Regular pheromone is indicated by solid arrows. The route over the nodes 3, 6 and 7 is the result of a reactive route setup. The one-hop route from node 5 to node 8 is there independent of the running session: node 5 is aware that node 8 is its neighbor and therefore knows it has a one hop path to node 8. Virtual pheromone is indicated by dashed arrows. It forms a field pointing towards the destination node 8. Virtual pheromone is the result of the pheromone diffusion process.

its pheromone table, and picks a maximum number k of destinations it has routing information for. k is normally kept on 10, but in chapter 5 we also present results varying this parameter. If more than k destinations are available, k of them are picked out randomly. For each one of these destinations d, the hello message contains the address of d, the best pheromone value that i has available for d, υ_i^d, and a bit flag. This best pheromone value υ_i^d is taken over all possible values for regular pheromone τ_{ij}^d and virtual pheromone ω_{ij}^d associated with d in i's pheromone table $\mathcal{T}_i$. The bit flag is used to indicate whether the reported value was originally regular or virtual pheromone. In the example of figure 4.2, node 3 has the choice of reporting the regular pheromone value about the route over node 6 to node 8, or reporting the virtual pheromone value about the route over node 5 to node 8. In case the route cost metric in use is hop count, it will prefer to send out the virtual pheromone, as it points to a better route.

A neighboring node j receiving the hello message from i goes through the list of reported destinations. For each listed destination d, it derives from the hello message an estimate of the goodness of going from j to d over i, by applying information bootstrapping: it combines the reported pheromone value υ_i^d, which indicates the goodness of the best route from i to d, with the locally maintained estimate of the cost c_j^i of hopping from j to i. The exact formula is given in equation 4.4. The inversions are needed first to convert the goodness value υ_i^d

into a cost value so that it can be added to the cost value c_j^i, and then to convert the total sum again into a goodness value. The result of the calculation is what we call the bootstrapped pheromone value κ_{ji}^d. In the example of figure 4.2, node 3 receives a hello message from node 5 reporting the one hop route from 5 to 8. Node 3 extracts this reported pheromone and uses it to derive bootstrapped pheromone for the route over node 5 to node 8.

$$\kappa_{ji}^d = \left((\upsilon_i^d)^{-1} + c_j^i\right)^{-1} \tag{4.4}$$

With κ_{ji}^d, node j has obtained a new estimate for the goodness of the path to d over i in a relatively cheap way. Thanks to the use of information bootstrapping, all that was needed in terms of communication overhead was the sending of the value υ_i^d from i to j. Moreover, since υ_i^d was piggybacked on top of a hello message, which i needed to send out anyway in order to support link failure detection, the overhead is limited to a few extra bytes in transmission. In AHWMNs, this is an important detail, since a major part of the cost of data transmission is formed by channel access control activities (see subsection 2.3.3), which only need to be executed once for each packet, making the transmission of one large packet favorable compared to the transmission of several small packets. On the downside, the cheap procedure to obtain κ_{ji}^d comes at a price, in the form of reliability: since κ_{ji}^d is derived from the estimate υ_i^d reported by i, it is only correct as long as υ_i^d is correct. This can be problematic in a highly dynamic environment like AHWMNs, where routing information can get out of date quickly, and especially if the value υ_i^d reported by i was in itself the product of pheromone diffusion (i.e., if the value reported by i was originally virtual pheromone). We have provided discussions on the reliability of information bootstrapping and dynamic programming approaches in subsections 3.1.1 and 3.3.2. Since we do not adopt any additional mechanisms to ensure the reliability of the bootstrapped pheromone (in order to keep the system simple), and since the bootstrapping process is relatively slow using the periodic hello messages (in order to keep it efficient), we have to be aware that the bootstrapped pheromone value κ_{ji}^d is potentially unreliable. This influences the way node j can use κ_{ji}^d to update its pheromone table.

As described earlier in subsection 4.2.1, node j maintains in its pheromone table entry $\mathcal{T}_{ji}^d$ two distinct pheromone values for the route over its neighbor i to destination d: the regular pheromone value τ_{ji}^d and the virtual pheromone ω_{ji}^d. Of these, only the virtual pheromone value ω_{ji}^d is normally updated with the new bootstrapped pheromone value κ_{ji}^d. This way, the pheromone obtained via the pheromone diffusion process is kept separate from the regular pheromone, which is the product of ant based route sampling and is therefore considered more reliable. The updating is done by replacing ω_{ji}^d by κ_{ji}^d. In the example of figure 4.2, node 3 would use the earlier derived bootstrapped pheromone about the route over node 5 to node 8 to update its virtual pheromone. The approach of keeping virtual and regular pheromone separate means that bootstrapped pheromone is not used directly for the forwarding of data packets,

since data packets only consider regular pheromone when choosing a next hop (see subsection 4.2.4). Virtual pheromone is used when forwarding proactive forward ants towards their destination (more on this follows below, when we talk about proactive ant sampling). When reaching the destination, proactive forward ants are converted into proactive backward ants, which do deposit regular pheromone, which in turn is used for routing data packets. So, in this way, bootstrapped pheromone influences data forwarding indirectly. One could say that the potentially unreliable bootstrapped pheromone provides hints about possible routes, which are then explored and verified by the proactive forward ants.

There is one situation that forms an exception to this normal mode of operation, in which we do allow the bootstrapped pheromone value κ_{ji}^d to be used for updating the regular pheromone value τ_{ji}^d and influencing data forwarding directly. This is the case when the following two conditions are fulfilled: a) j already has a non-zero value for the regular pheromone τ_{ji}^d, and b) the bootstrapped pheromone κ_{ji}^d was derived from a reported pheromone value υ_i^d that was based on regular pheromone in i, rather than virtual pheromone (remember that i indicates this in a bit flag in its hello message). In the example of the network of figure 4.2, the described situation arises for instance when node 3 receives a hello message from node 6 reporting the regular pheromone value of the path from node 6 to node 8 going over node 7: node 3 already has non-zero regular pheromone for the route over node 6 to node 8, and the hello message received from node 6 reports regular pheromone. Under these conditions, we know that there is a reliable route from j to d over i, since the presence of regular pheromone indicates that this route has been sampled by ants in the past. Also, we know that κ_{ji}^d reflects information about this reliable route that is available in the next hop i, since it was based on a value υ_i^d that reflects regular pheromone about d available in i. This means that the bootstrapped pheromone is in fact a one step update of the routing information about this specific route. So, under these strict conditions, we consider it reliable enough to update regular pheromone: we replace τ_{ji}^d directly by κ_{ji}^d. This way, pheromone on current paths is kept up-to-date.

Proactive ant sampling

The proactive ant sampling process is started by the source node of a session at the moment the first data packet of a new session is received, and continues for as long as the session is going on. The aim of the process is to use ant based sampling to gather routing information for ongoing sessions. To this end, proactive forward ants are generated. These ants can follow regular pheromone, which is routing information placed by previous ants, or virtual pheromone, which is the result of the pheromone diffusion process described above. While the former leads the ants to update goodness estimates of existing routes, the latter allows them to find new routes based on the hints provided by the pheromone diffusion process. This way, the single route that was initially constructed in the reactive route setup process is extended to a full mesh of multiple paths. In the

example of figure 4.2, a proactive forward ant arriving in node 3 can follow the regular pheromone over node 6 to node 8, or the virtual pheromone indicating the shorter route over node 5 to node 8.

Each node which is the source of a communication session periodically (normally, we use a period of t_{hello} seconds, but we have also done tests with other values; see subsection 5.3.3) schedules the transmission of a proactive forward ant towards the session's destination. In order to improve efficiency, the actual sending of a proactive forward ant is conditional to the availability of good new virtual pheromone: only if the best virtual pheromone is significantly better (in our experiments: at least 10% better) than the best regular pheromone, a proactive forward ant is sent out. The aim of the proactive forward ant is to find a route towards the destination, and to store the list of nodes $\mathcal{P}$ that it visits on the way. The proactive forward ant takes a new routing decision at each intermediate node i, using the formula of equation 4.5 to calculate the probability of choosing each possible next hop n. In this formula, the function $max(a, b)$ takes the maximum of the two values a and b, and β_2 is a parameter that defines the exploratory character of the ants. Like for reactive forward ants, β_2 is normally kept on 20, but in chapter 5 we also compare results using different values for β_2. As can be seen from the equation, unlike reactive forward ants, proactive forward ants rely both on regular and virtual pheromone for their routing decisions: they use the maximum between regular and virtual pheromone to calculate the probability of each next hop. Also different from reactive forward ants is that proactive forward ants are never broadcast: when they arrive at a node that does not have any routing information for their destination, they are discarded.

$$P_{in}^d = \frac{[max(\tau_{in}^d, \omega_{in}^d)]^{\beta_2}}{\sum_{j \in N_i^d} [max(\tau_{ij}^d, \omega_{ij}^d)]^{\beta_2}}, \quad \beta_2 \geq 1, \tag{4.5}$$

When a proactive forward ant arrives at its destination, it is converted into a proactive backward ant, which is sent back to the source. Proactive backward ants have the same behavior as reactive backward ants: they follow the exact path $\mathcal{P}$ recorded by their corresponding forward ant back to the source, and update regular pheromone entries at intermediate nodes and at the source. For details about this behavior, we refer to the description of reactive backward ants in subsection 4.2.2.

An important aspect to note here is that while the proactive forward ants can follow both regular and virtual pheromone, proactive backward ants always deposit regular pheromone. This way, the proactive ant sampling process can investigate promising virtual pheromone, and if the investigation is successful turn it into a regular route that can be used for data. In the example of figure 4.2, a proactive forward ant following the virtual pheromone from node 3 over node 5 to node 8 is at its arrival in 8 converted into a backward ant, which deposits regular pheromone on the link from node 3 to node 5. The process of proactive ant sampling increases in this way the number of routes available for data routing, which can grow to a full mesh, and allows the algorithm to

exploit new routing opportunities in the ever changing AHWMN topology. As stated earlier, the proactive ants provide through their full path sampling the necessary control to verify the reliability of new routes obtained through the information bootstrapping process of pheromone diffusion.

4.2.4 Data packet forwarding

Data packets are forwarded from their source to their destination in hop-by-hop fashion, taking a new routing decision at each intermediate node. Routing decisions for data packets are based only on regular pheromone. This means that they only follow the reliable routes that are the result of ant based sampling, and leave the virtual pheromone information that is the result of information bootstrapping out of consideration. The combination of the reactive route setup and the proactive route maintenance processes leads to the availability of a full mesh of such reliable routes between the source and destination of each session.

Nodes in AntHocNet forward data packets stochastically, based on the relative values of the different regular pheromone entries they have available for the packet's destination. The probability P_{in}^d for a node i to pick next hop n when forwarding a packet with destination d is given in the formula of equation 4.6. This formula is very similar to the one used for reactive forward ants (see equation 4.1), but uses a different parameter, β_3, for the power function of the pheromone values. This way, the relative preference for the best routes can be adapted separately for data and for ants (as is common practice in ACO routing algorithms, see section 3.2).

$$P_{nd} = \frac{(\tau_{in}^d)^{\beta_3}}{\sum_{j \in N_i^d} (\tau_{ij}^d)^{\beta_3}}, \quad \beta_3 \geq 1 \tag{4.6}$$

By adapting the β_3 parameter, one can make data forwarding less or more greedy with respect to the best available routes. By setting β_3 low, data is spread over multiple routes, considering also low quality ones. Using multiple routes for data forwarding can improve throughput, as the data load is spread more evenly over the available network resources (see also subsection 2.4.3). By setting β_3 high, on the other hand, data is concentrated on the best routes. This can also be a good choice, since the routes that according to the ant sampling give the best performance, are exploited as much as possible. In our experiments, we normally keep β_3 on 20, which is relatively high and only allows data load spreading when there are several good routes of more or less equal quality. In chapter 5, we also compare results when using different values for β_3.

4.2.5 Link failures

In AHWMNs, link failures can occur due to physical changes such as the movement or disappearance of a node, or due to changes that influence the connectivity of the wireless communication, such as an increase of radio interference or

a decrease in the used transmission power. Since AHWMNs are usually highly dynamic, such events are expected to occur frequently, and AHWMN routing algorithms should be prepared to deal with them effectively. The components of AntHocNet described so far already offer some basic protection against link failures. The reactive route setup process allows source nodes to rebuild entire routes if needed, and the proactive route maintenance process offers protection in a proactive way through the creation of new paths, which can serve as backup routing possibilities. In this subsection, we outline further mechanisms in AntHocNet that are specifically aimed at dealing with link failures.

The first step in dealing with link failures is their detection. In AntHocNet, link failures are detected if lower layer protocols report the failure of the unicast transmission of a control or data packet, or if a node fails to receive periodic hello messages from its neighbors. Once a failure is detected, the next step is to take action to neutralize its effect. In AntHocNet, the action to be taken depends on the way the failure was detected. If the detection was through the failed transmission of a control packet or through the missed reception of hello messages, the node that detected the link failure broadcasts a link failure notification message, in which it warns downstream nodes about changed routes. If the detection was through the failed transmission of a data packet, the node starts a local route repair process in order to repair the route to the destination of the failed data packet. A final action that can be taken is the sending of a unicast warning message. These are messages that are used when an earlier broadcast link failure notification message got lost. Below, we first discuss the detection of link failures, then the use of link failure notification messages, next the process of local route repair, and finally the use of unicast warning messages.

Detecting link failures

Link failures can be detected through the failed unicast transmission of control or data packets, or via the use of hello messages. Detection through a failed unicast transmission is straightforward. MAC layer protocols usually have mechanisms that inform it about the success or failure of a unicast transmission. For instance, the IEEE 802.11 DCF protocol, which is often used in AHWMNs (see subsection 2.3.3), requires receiving nodes to send an acknowledgement upon successful reception of a unicast transmission. AntHocNet gives MAC layer protocols the possibility to report the failure of a transmission, and assumes in that case that the corresponding link has failed.

Relying solely on this mechanism to detect link failure is not satisfactory however. First of all, it does not allow to detect a link failure in advance, but only at the moment that it causes damage. Second, on a more technical note, many implementations of MAC layer protocols do not include the possibility to warn higher layers about a failed transmission. Therefore, AntHocNet also uses hello messages to detect link failures. These are messages that are sent out by all the nodes of the network asynchronously at a fixed interval of t_{hello} seconds. When a node i receives a hello message from a new node j, it can assume that j is its neighbor, and create an entry $\mathcal{N}_{ij}$ for j in its neighbor

table, indicating in it the last time that it has heard from j. It also makes an entry $\mathcal{T}_{ij}^{j}$ in its pheromone table, indicating that there is a one-hop route from i to j over next hop j. After this, i expects to receive a message from j at least every t_{hello} seconds. If i does not hear from j for a certain number of t_{hello} second intervals (set to 2 intervals here), i assumes that the wireless connection to j has disappeared.

In AntHocNet, hello messages are not only used to detect link failures, but also to carry pheromone information in the pheromone diffusion process (see subsection 4.2.3). This means that hello messages in AntHocNet are larger than those used in many other routing algorithms (such as e.g. in AODV [213], see subsection 2.4.2). For link failure detection, this can actually be an advantage. It has been pointed out in [56] that since hello messages are usually smaller than data packets, they can more easily be received correctly over shaky wireless connections, and therefore give a false image of link availability for data packets. The authors propose exactly the use of larger hello messages to get a better image of the real network topology.

Link failure notifications

When a node i detects that the link with a neighboring node j is lost, it removes j from its neighbor table. Then, it updates its pheromone table $\mathcal{T}_i$, building a link failure notification message in the process. It scans its pheromone table to control which destinations d have a non-zero regular pheromone value τ_{ij}^{d} (i.e., for which destinations d neighbor node j is used as a next hop from i). For each such destination, i sets τ_{ij}^{d} to 0. Furthermore, it checks whether the lost pheromone τ_{ij}^{d} was the best or only regular pheromone value available for d. If this is the case, it adds the address of the destination d to the link failure notification message, together with the new best regular pheromone it has available for d. If τ_{ij}^{d} was the only non-zero regular pheromone entry for d, this is also indicated in the link failure notification message.

Once the link failure notification message is fully constructed, it is broadcast. All of i's neighbors receive the message, and update their routing tables for the routes going over i to the involved destinations, using the new estimates reported in the message. To this end, they use the same formula that is applied for information bootstrapping in the pheromone diffusion process, as given in equation 4.4. In case they in turn loose their best or only route to one of the involved destinations due to the reported failure, they in turn construct their own link failure notification message, in the same way as i did, and broadcast it further. This way, all involved nodes eventually get warned and can update their pheromone tables.

Local route repair

When a node i detects a link failure through the failed transmission of a data packet, and i does not have any alternative routing information available for the destination d of this data packet, i does not include d in the link failure

notification it sends out. Instead, it starts a local route repair process to try to repair the route to d, so that the data packet can still be delivered.

At the start of the local route repair process, i creates a repair forward ant. Repair forward ants are identical to reactive forward ants, and are forwarded in the same way: they are broadcast when no routing information is available, and are otherwise unicast to a stochastically chosen next hop using the formula of equation 4.1. The only real difference with reactive forward ants is that repair forward ants can only be broadcast a limited maximum number of times (we set this number to 2). Therefore, they can only travel far if they are unicast over existing pheromone. Concretely, this means that repair forward ants need to stay close to existing routes in order to reach the destination, so that they really focus on the repair of the lost route. Upon arrival at the destination d, the repair forward ant is converted into a repair backward ant, which travels back to the node i that launched the local route repair process. It does so in exactly the same way as a reactive backward ant traveling back to its source (see subsection 4.2.2), updating regular pheromone entries on the way. Once the repair backward ant is back at the original node i, this node can send its stored data packet to d.

Node i uses a timer to decide how long to wait for a repair backward ant. The value of this timer is an estimate of the time it takes to go from i to d and come back, and is calculated as shown in equation 4.7. In this equation, t_{hop} is a fixed delay value per hop (set to 50 milliseconds), and h_{ij}^{d} is the number of hops to the destination as reported by the backward ants and stored in i's pheromone table (see subsections 4.2.1 and 4.2.2). The multiplication with 2 is to account for the way to go to the destination and come back. If no backward ant has been received before the timer runs out, i discards the stored data packet, and broadcasts a link failure notification about destination d.

$$timer = 2 * t_{hop} * h_{ij}^{d} \tag{4.7}$$

Unicast warning messages

A final aspect of dealing with link failures in AntHocNet is the use of unicast warning messages. These are emergency messages that are needed when link failure notification messages are not delivered correctly. This can happen because link failure notifications are broadcast. The IEEE 802.11 DCF MAC layer protocol, which is very often used in AHWMNs, does not provide any guarantees for the delivery of broadcast messages. This makes broadcast transmission a lot less reliable than unicast transmissions, which are supported with extra mechanisms to improve reliability (see subsection 2.3.3). Suppose now that a node i has lost its only route to a destination d due to a link failure, and warns other nodes about this in a link failure notification message as described above. If a neighboring node n, which has a route to d using i as next hop, does not receive this message correctly, it will continue sending data packets for d to i. At this point, i cannot forward the data packets. It therefore answers to the data packet by unicasting a short warning message to n, indicating that it has

no routing information for d. Upon reception of this message, n removes the erroneous routing information from its pheromone table.

4.2.6 Routing metrics

So far, we have not given any details about how paths are evaluated in AntHocNet, and have instead talked in terms of a generic cost value. In principle, this generic cost value can be replaced by any possible route cost metric. Concretely, we have explored the use of the following ones: 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. While the calculation of the number of hops is trivial, the other three are a bit more complicated. Therefore, we explain in what follows for each of these metrics how a node i locally estimates the cost c_i^j of the link to its neighbor j. How local estimates of link costs are then combined into full route costs has been explained earlier in the description of the working of reactive backward ants (see subsection 4.2.2). In the experimental results reported in chapters 5 and 6, we normally use the signal-to-interference-and-noise ratio metric, as this gave the best results. Comparisons with versions of AntHocNet using the other metrics will also be reported in chapter 5.

End-to-end delay

When using the end-to-end delay cost metric, the cost estimate c_i^j maintained locally by node i reflects the expected delay incurred by a data packet when following the wireless link from i to its neighbor j. Concretely, c_i^j is calculated by the formula given in equation 4.8. In this formula, q_{mac}^i is the number of packets that are currently in queue at the node to be sent, and $\hat{t}_{mac}^i$ is an estimate of the time it takes to send one packet via unicasting. $\hat{t}_{mac}^i$ is calculated as a moving average of the time elapsed between the arrival of a packet at the MAC layer and the end of a successful transmission, which is indicated by an acknowledgement received from the next hop. This is shown in equation 4.9, where t_{mac}^i is the latest observed send time, and η is a parameter defining how quickly the moving average adapts to new observations (η is kept on 0.7).

$$c_i^j = (q_{mac}^i + 1)\hat{t}_{mac}^i \tag{4.8}$$

$$\hat{t}_{mac}^i \leftarrow \eta\hat{t}_{mac}^i + (1-\eta)t_{mac}^i, \quad \eta \in [0,1] \tag{4.9}$$

As can be seen from equations 4.8 and 4.9, the calculation of c_i^j here is in fact independent of j. This is because we assumed network nodes that have only one wireless interface which is an omnidirectional antenna. In this case, the data traffic for all next hops needs to go over the same outgoing queue, and needs to access the same wireless channel, so that a packet queuing to be sent to a node j might need to wait behind packets for any other neighboring node, experiencing also their delays.

End-to-end delay combined with number of hops

We also considered the possibility to combine the end-to-end delay with the number of hops. While the end-to-end delay is adaptive to the local traffic situation, it can be quite unstable, showing large oscillations due to variations in the quality of the wireless channel and local interference. The number of hops, on the other hand, is not adaptive, but is a stable metric. The goal of combining both is to have a metric that is both adaptive and stable.

The formula for the calculation of c_i^j using the combined metric is given in equation 4.10. The first part of this formula corresponds to the calculation of the delay, and is identical to the formula of equation 4.8. The second part of the formula reflects the number of hops. While the number of hops to reach neighbor j from node i is obviously always 1, here we use a different constant value, namely t_{hop}. This is a fixed estimate of the time needed to take one hop in unloaded conditions (we kept t_{hop} on 0.003 sec). Using the constant t_{hop}, rather than 1, allows to scale the number of hops to the same order of magnitude as the time estimation.

$$c_i^j = (q_{mac}^i + 1)\hat{t}_{mac}^i + t_{hop} \tag{4.10}$$

Signal-to-interference-and-noise ratio

The signal-to-interference-and-noise ratio (SINR) between a node i and a node j is the ratio between the strength of the signal received by node i from node j and the general noise and interference from other radio signals present around i. This value can be calculated at the physical layer of the node. SINR is a crucial factor defining the success of a wireless reception. When SINR is high, reception has a high probability of being successful, whereas when it is low, reception is impossible. In between there is a range for which reception is possible with some probability. Note that also factors other than SINR can influence reception, so that it is sometimes also possible to have bad reception when SINR is high (see e.g. [15]). Nevertheless, SINR is an important indicator of link quality.

When using SINR to define c_i^j in AntHocNet, we are not interested in fine variations in the SINR level, but rather in a coarse grained distinction between "good" and "bad" wireless links: we want to capture the difference between links on which reception has a high probability of being successful, and links on which reception is possible but with a lower probability of success. Therefore, we apply a simple approach using a critical SINR value $SINR_c$ as threshold, in which links with an SINR value lower than $SINR_c$ are penalized. Concretely, we use the formula of equation 4.11, where c_i^j is set to 1 for links with SINR higher than $SINR_c$ (this corresponds to using normal hop count as a metric), and to a constant value $c_{const} > 1$ for links with SINR lower than $SINR_c$. In simulation tests using IEEE 802.11b radios sending at 2Mbps, we empirically set $SINR_c$ to 17dB and c_{const} to 3. The value of 17dB for $SINR_c$ is in line with critical values of SINR found in empirical research on wireless LANs using the same radio technology [200].

$$c_i^j = \begin{cases} 1, & \text{if } SINR > SINR_c \\ c_{const}, & \text{if } SINR \leq SINR_c \end{cases} \tag{4.11}$$

4.3 Further Discussions

In this section, we provide further discussions related to the AntHocNet routing algorithm. First, we consider AntHocNet in the light of the RL framework presented in chapter 3, and discuss its particular strategies for information gathering from this point of view. Then, we take a look at the different challenges for AHWMN routing that were pointed out in chapter 2, and investigate qualitatively how AntHocNet deals with these. Next, we discuss how AntHocNet relates to other existing routing algorithms. In particular, we search in how far components of AntHocNet are also used elsewhere. Finally, we write a few words about mechanisms that were present in older versions of AntHocNet, and about why they were discarded.

4.3.1 AntHocNet and reinforcement learning

In chapter 3 we have described RL, an important class of problems in machine learning, and we have explained how the problem of routing fits into this framework. Then, we have discussed two basic solution methods for RL problems, namely dynamic programming and Monte Carlo sampling, each with their own advantages and disadvantages, and we have shown how existing routing algorithms relate to them, with distance vector routing being a direct implementation of dynamic programming, and ACO routing relying mainly on Monte Carlo sampling. Subsequently, we have also described a more advanced learning method, temporal difference learning, which combines elements of both basic methods, and we have shown the Q-routing algorithm that was based on it. In what follows, we investigate how the AntHocNet routing algorithm proposed in this chapter relates to all of this. In particular, we describe how AntHocNet uses both the Monte Carlo sampling and dynamic programming learning methods, but combines them in a way that is different from temporal difference learning.

Monte Carlo sampling is used extensively in AntHocNet, since it is the learning method applied in ACO routing, which was the main source of inspiration of our algorithm. This is most evident in the proactive ant sampling subprocess of the proactive route maintenance process (see subsection 4.2.3). In this subprocess, ants are regularly sent out by the source node of each session in order to sample a path towards the destination of the session. This is very much in line with the continuous path sampling done in other ACO routing algorithms. Apart from this, also the reactive route setup process (see subsection 4.2.2) and the local route repair process (see subsection 4.2.5) use a form of Monte Carlo sampling: the reactive and repair forward ants used in these processes each take a single sample of a path through the network, and use it to set up a new route. As was explained earlier in subsection 3.3.2, this approach of using Monte Carlo sampling by taking a single sample of a path through the network

is also applied in many existing reactive routing algorithms. An important point to note here is that AntHocNet is more consistent in its use of sampling, since both reactive and repair forward ants always go all the way till the destination. In most existing reactive routing algorithms, including the AODV and DSR protocols described in subsection 2.4.2, this is not the case: RREQ messages that are searching for a path to the destination can be returned by intermediate nodes that have routing information about the destination available. At that point, the obtained routing information relies on the estimate provided by the intermediate node, so that a form of information bootstrapping is applied.

The most important advantage of using Monte Carlo sampling here is that it provides a high level of reliability. This is because all routing information is the result of direct experiences, giving a certain guarantee about its correctness. A disadvantage is that it can be inefficient. The need to send sampling packets from source to destination can lead to high levels of overhead. The use of IEEE 802.11 DCF as MAC layer mechanism can deteriorate this problem, because this protocol creates a lot of overhead for each sent packet, making the transmission of multiple small packets particulary problematic. Traditional reactive routing algorithms deal with the efficiency issue by using just a single sample. AntHocNet, on the other hand, improves efficiency by combining Monte Carlo sampling with a supporting dynamic programming process.

AntHocNet uses dynamic programming in the pheromone diffusion subprocess of its proactive route maintenance process (see subsection 4.2.3). This subprocess works more or less in the same way as distance vector routing algorithms do, using information bootstrapping in each node to derive routing information from estimates calculated by neighboring nodes. Such an approach has as an advantage that it is highly efficient, as the information obtained by each node is optimally reused in the calculation of the information needed by other nodes. In AntHocNet, this efficiency is further increased by piggybacking routing updates on top of hello messages, which avoids the transmission of multiple small control packets. An important disadvantage of using dynamic programming and information bootstrapping is that it can lead to processes that are slow to converge, so that routing information can be temporarily unreliable. This is especially a problem in dynamic situations. In existing distance vector routing algorithms for AHWMNs, such as DSDV (see subsection 2.4.2), extra techniques are applied to ensure reliability in the face of the highly dynamic network environment. Unfortunately, these techniques introduce extra overhead, and can severely reduce the efficiency of the process. Therefore, in AntHocNet, we have chosen a different strategy: the dynamic programming part of AntHocNet is kept very simple and lightweight, and is not expected to provide information that is 100% reliable. This way, we focus maximally on its efficiency, and do not worry about its unreliability. Instead, we are aware that the information produced by the process can contain errors and therefore we do not use it directly for routing. We use it as a guideline for the Monte Carlo sampling of the proactive ant sampling subprocess. Using this guideline, the ants do not have to explore the whole network, but can concentrate on routes that are suggested by pheromone diffusion. This reduction in the need

for exploration makes the sampling process more effective, so that less ants are needed, leading to better efficiency. This way, we obtain an adaptive algorithm that combines the efficiency of dynamic programming with the reliability and robustness of Monte Carlo sampling.

Note that the way Monte Carlo sampling and dynamic programming are combined here is very different from temporal difference learning methods. In n-step temporal difference learning (see subsection 3.3.3), the learning agent takes a sample of a few steps, after which it arrives in an intermediate state i, where it reads the local value estimate, which it uses to bootstrap on. This approach can be highly efficient, but does not avoid the potential unreliability of dynamic programming, since it still uses information bootstrapping. It is interesting to see that the temporal difference learning approach to information gathering is similar to the returning of RREQ messages by intermediate nodes in reactive routing algorithms as described above: the RREQ samples a path till an intermediate node that has routing information available about the destination, at which point it bootstraps on this information and returns to the source. In AntHocNet, we aimed to be more consistent, and keep sampling and information bootstrapping strictly separate.

4.3.2 Challenges for routing in AHWMNs

In chapter 2, we have defined a number of challenges for routing in AHWMNs. These included adaptivity, robustness, efficiency and scalability. Here, we investigate qualitatively how AntHocNet is equipped to deal with each of these.

Adaptivity is very important in AHWMNs, as the dynamic network environment constantly presents the routing algorithms with new changes. Adaptivity is in AntHocNet provided in two different ways. On the one hand, the algorithm has a wide range of reactive mechanisms at its disposal. These include the reactive route setup process and the different mechanisms to deal with link failures, such as link failure notifications, local route repair, and unicast warning messages. These provide the algorithm with tools to react immediately in case a disruptive event takes place. On the other hand, AntHocNet applies also proactive mechanisms, in its proactive route maintenance process. These allow the algorithm to take adaptive actions without the need for being prompted by an event. Proactive actions can avoid problems with disruptive events in the future, by providing backup routes, or exploit new possibilities that arise from the changes in the environment.

Robustness is in general obtained from the extensive use of ant based full path sampling, a practice that is taken over from ACO routing. As was explained in chapter 3, using full path sampling as a method to gather routing information leads to increased robustness in two ways. First of all, each individual sample is unimportant, so that the loss of control packets only has a marginal effect, not immediately leading to erroneous routing information. Second, since ants always sample full paths, they provide a certain guarantee that the path actually exists and the reported information is correct. This is in contrast with distance vector routing and link state routing, which are both in principle more vulnerable in

a highly dynamic environment.

Efficiency is taken care of in AntHocNet in two ways. First of all, the algorithm's hybrid architecture combining reactive and proactive components allows to concentrate on the routing information that is most needed. Second, the use of a highly efficient dynamic programming approach, which uses piggybacking on top of hello messages, as a way to guide ant based sampling in the proactive route maintenance process allows to improve efficiency during the gathering of routing information.

Good scalability is expected to arise from the provided efficiency, adaptivity and robustness. We refer here to the empirical results in chapter 5 that show the scalability of the algorithm compared to other, state-of-the-art routing algorithms for AHWMNs.

4.3.3 AntHocNet related to other routing algorithms

In this subsection we take a look at routing algorithms that are related to AntHocNet. We isolate mechanisms that are used in AntHocNet, and investigate to what extent they are also applied in other AHWMN routing algorithms. In particular we will talk about multipath routing, the use of path sampling, and the approach to combine reactive route setup with proactive route improvement.

Multipath routing is used in many AHWMN routing algorithms. An overview has been given earlier in subsection 2.4.3. The main objectives when using multipath routing is to improve failure resilience by providing backup paths, and to improve throughput. The former is inherent to all algorithms that set up multiple paths. The latter can only be obtained when data traffic is spread over the multiple paths. In AntHocNet, this is possible by choosing a low value for the parameter β_3 in equation 4.6, which leads to a stochastic spreading of the data load according to the relative quality of the different available paths. Such an approach is typical for ACO routing algorithms, and can therefore also be found in some of the other algorithms that apply ACO routing for AHWMNs, such as ARA [119] and Termite [225] (nevertheless, many ACO routing algorithms also forward data deterministically over the best path; see subsection 3.2.6 for an overview). Outside the area of ACO routing, adaptive data load spreading is far less common. Most algorithms that do spread data over multiple paths do so in a simple, even way (see e.g. [164]). A few algorithms explore the idea of adaptive data load spreading depending on the estimated quality of the paths (e.g. [108, 269]). Stochastic data load spreading is to the best of our knowledge unexplored outside the area of ACO routing.

The use of path sampling as a strategy for obtaining routing information has been discussed amply in chapter 3 and in subsection 4.3.1. AntHocNet applies sampling both reactively, using a single sample to set up a route between source and destination, and proactively, using repeated samples to update existing routing information and explore new possibilities. As was mentioned before, reactive use of single path samples is quite common in AHWMNs. It is at the basis of the working of some important algorithms, such as AODV [213] and DSR [140] (see subsection 2.4.2). Proactive use of repeated path sampling is

far less common. Some algorithms apply it in a limited way, using sampling to get up-to-date information about existing routes, but not to explore new ones. This is the case in [108, 269], and in many of the ACO routing algorithms for AHWMNs (see subsection 3.2.6). The use of sampling to proactively find new routing information is quite rare in AHWMNs, even for ACO routing algorithms. Exceptions are the Termite [225] and EARA [177] algorithms, in which ants (or in the case of Termite any kind of packets) can take random routing decisions, so that they leave existing routes, and start exploring new ones. Such random exploration is quite blind. A system where the exploration is guided such as in AntHocNet's proactive route maintenance process has to our knowledge not been explored outside the work presented in this thesis.

A hybrid strategy of combining reactive route setup with proactive route improvement like the one used in AntHocNet is not very common in AHWMN routing. It is applied in some ACO routing algorithms such as Termite and EARA through the use of random exploration decisions during the path sampling process, as is explained above. Outside the area of ACO routing, the approach can to some extent be found in the reactive routing protocol DSR. As was explained in subsection 2.4.2, DSR uses source routing, which means that each data packet carries the full route from its source to its destination, as a list of addresses. Nodes that are not on this route can overhear the data packet, and extract the routing information it is carrying. This allows these nodes to discover new routes. Unlike AntHocNet, however, DSR does not include any mechanism to verify the reliability of these new routes, and in experiments this mechanism has often been found ineffective, as it allows erroneous routing information to be copied by other nodes, leading to a quick "pollution" of routing tables throughout the network. Nevertheless, DSR's approach to route improvement has recently been taken over in two new routing algorithms: LQSR [92,93] and Srcr [30]. Quite interestingly, both algorithms have been developed specifically for use in real WMN deployment projects: LQSR for Microsoft's MCL architecture, and Srcr for MIT's Roofnet project. A very different approach to proactive route improvement is found in the LUNAR algorithm [259], which is in essence a reactive algorithm in which improvements are obtained by repeating the route setup phase every 3 seconds. For efficiency reasons, the algorithm is limited to networks of maximally 3 hops. Like LQSR and Srcr, also LUNAR was developed based on experiences with a real WMN deployment project. The fact that proactive route improvement was chosen as the approach in several WMN deployment studies is an indication of its usefulness in realistic settings.

4.3.4 Older versions of AntHocNet

In the first papers about AntHocNet [73, 74], an older version of the algorithm was described. This version contains some important differences compared to the version described in this chapter. Specifically, it uses different mechanisms in the reactive route setup process and in the proactive route maintenance process. Here, we describe these different mechanisms, and explain why we dropped them.

The main difference in the reactive route setup process of the older version of AntHocNet is that not one but multiple routes are set up. To this end, nodes that receive multiple reactive forward ants belonging to the same route setup process do not immediately discard these duplicate ants. Instead, they compare the path traveled by each ant to that of the previously received ants of this route setup. If the number of hops and travel time of a newly received ant are both within an acceptance factor a_1 of those of the best previously received ant of the same route setup, the new ant is accepted and forwarded; otherwise, it is discarded. a_1 is set quite low (to 0.9), in order to only allow the best ants through and avoid too much proliferation of forward ants in the network. The multiple reactive forward ants arriving in the destination are converted into backward ants, which return to the source, so that a number of multiple, good paths are set up simultaneously. A problem with the approach is that due to the use of strict acceptance criteria when comparing to the best previously received ant (using the low acceptance factor a_1), the process can lead to a situation where the different resulting paths are all just small variations of the best one. In general, it is better to have more disjoint paths, as this gives better protection in case of link failures. To boost the creation of disjoint paths, a different mechanism is applied, which takes into account the first hop taken by each ant. If this first hop is different from those taken by previously accepted ants, we apply a higher (less restrictive) acceptance factor a_2 than in the case the first hop was already seen before (a_2 was set to 2). A similar strategy can be found in [186].

The strategy of setting up multiple routes during the reactive route setup process has as an obvious advantage that multiple routes are available from the start of the communication session, so that the session is better protected against link failures and can start data load spreading immediately. However, through experiments we experienced that it is hard to get a good balance between the number of routes that are obtained and the overhead that is created. High levels of overhead were often experienced. Therefore, we decided to restrict reactive route setup to the creation of just one single route, and to rely on proactive route maintenance to obtain multiple routes.

The proactive route maintenance process of the older version of AntHocNet is considerably different from the new one. It consists of only the proactive ant sampling subprocess, and does not apply pheromone diffusion. Proactive forward ants can be forwarded through unicasting or through broadcasting. The unicasting is done in the same way as in the current version of AntHocNet: a next hop is chosen probabilistically according to available pheromone information. However, since no pheromone diffusion is done, no virtual pheromone is spread out, and the only available pheromone information for the proactive forward ants is regular pheromone. This means that through pheromone guided unicasting, ants can check out existing routes, but not explore new ones. This is where the broadcasting comes into play. At each node, proactive forward ants have a small probability (set to 10%) of being broadcast. After such a broadcast the ant arrives in all the neighbors of the broadcasting node. This way, it can leave the currently existing paths and start the exploration of new ones. It is

possible that in these neighbors it does not find pheromone pointing towards the destination, in which case it is broadcast again. The ant will then quickly proliferate and flood the network, like reactive forward ants do. In order to avoid this, the number of broadcasts is limited to n_b (n_b was set to 2). If the ant does not find routing information within n_b hops, it is deleted.

Compared to the route exploration done in the current version of AntHocNet, this older mechanism has some important shortcomings. First of all, its exploration is completely blind: the broadcasts are done randomly, without using any information about whether it would be possible to find new good routes there. Second, it creates a lot of overhead. Due to the n_b possible broadcasts, each proactive forward ant can multiply quickly, leading to a lot of extra control packets in the network. Therefore, n_b needs to be kept quite low. As a consequence, however, exploration is more limited, and the number of exploratory moves cannot be more than n_b. The proactive route maintenance process adopted in the current version of AntHocNet is both more effective and more efficient.

4.4 Conclusion

In this section we have presented the AntHocNet routing algorithm for AHWMNs. AntHocNet is a hybrid algorithm that combines a reactive route setup process with a proactive route maintenance process. The reactive route setup is carried out at the start of a communication session or whenever the source of a current session has no more routing information available for the destination. It creates a single route for the session. The proactive route maintenance is run for the entire duration of the session. Its aim is to keep information about existing routes up to date and to explore new routes. Under impulse of this process, the initial single route created during reactive route setup is extended to a full mesh, and improved to exploit new possibilities in the changing AHWMN environment. Other features of AntHocNet are the possibility to do probabilistic data load spreading, and the use of a number of reactive components to deal with link failure, such as the transmission of failure notification messages and the possibility to execute local route repair.

Considered from a machine learning point of view, AntHocNet relies on two distinct strategies for information gathering, namely Monte Carlo sampling and dynamic programming. The Monte Carlo sampling approach is inherited from ACO routing, which was the main source of inspiration for AntHocNet. It is applied extensively throughout the different components of the algorithm. Collecting routing information through the sampling of full paths leads in general to reliable routing information. Dynamic programming is only used during proactive route maintenance. It is the basis of the pheromone diffusion subprocess, which uses information bootstrapping to spread earlier obtained routing information over the network. Using a dynamic programming approach allows to gather routing information in an efficient way. However, it can sometimes temporarily lead to erroneous information. Therefore, it is in AntHocNet combined

with full path sampling in order to improve reliability. The way ideas from dynamic programming and Monte Carlo sampling are combined in AntHocNet is novel in the area of machine learning.

The different mechanisms used in AntHocNet help the algorithm to deal with some of the important challenges of AHWMN routing that were pointed out in chapter 2, such as adaptivity, robustness and efficiency. Adaptivity is on the one hand provided by the availability of reactive algorithms such as the reactive route setup and the different mechanisms to deal with link failures, which make sure that disruptive events can always be dealt with. On the other hand, the use of proactive route maintenance allows to adapt to change in the environment before they cause disruptions. Robustness is in general provided through the use of full path sampling to establish routes for data traffic. Efficiency is obtained by combining the path sampling with a dynamic programming approach which allows to spread routing information in an efficient way.