USING ANT AGENTS TO COMBINE REACTIVE AND PROACTIVE STRATEGIES FOR ROUTING IN MOBILE AD HOC NETWORKS

FREDERICK DUCATELLE,* GIANNI DI CARO AND LUCA MARIA GAMBARDELLA
Istituto Dalle Molle sull'Intelligenza Artificiale (IDSIA), Galleria 2
CH-6928 Manno-Lugano, Switzerland

This paper describes AntHocNet, an algorithm for routing in mobile ad hoc networks based on ideas from the Ant Colony Optimization framework. In AntHocNet a source node reactively sets up a path to a destination node at the start of each communication session. During the course of the session, the source node uses ant agents to proactively search for alternatives and improvements of the original path. This allows to adapt to changes in the network, and to construct a mesh of alternative paths between source and destination. The proactive behavior is supported by a lightweight information bootstrapping process. Paths are represented in the form of distance-vector routing tables called pheromone tables. An entry of a pheromone table contains the estimated goodness of going over a certain neighbor to reach a certain destination. Data are routed stochastically over the different paths of the mesh according to these goodness estimates. In an extensive set of simulation tests, we compare AntHocNet to AODV, a reactive algorithm which is an important reference in this research area. We show that AntHocNet can outperform AODV for different evaluation criteria under a wide range of different scenarios. AntHocNet is also shown to scale well with respect to the number of nodes.

Keywords: Mobile Ad Hoc Networks; Routing; Ant Colony Optimization.

1. Introduction

Mobile Ad Hoc Networks (MANETs)[1] are communication networks in which all nodes are mobile and communicate with each other via wireless connections. There is no fixed infrastructure. All nodes are equal and there is no centralized control or overview. There are no designated routers: all nodes can serve as routers for each other, and data packets are forwarded from node to node in a multi-hop fashion. MANETs can be applied in situations where no fixed network infrastructure is available, such as military activities in enemy territory and disaster recovery operations.

Routing is the task of directing data flows from sources to destinations maximizing network performance. This is particularly difficult in MANETs. Due to the mobility of the nodes, the topology of the network changes constantly, and paths which were initially efficient can quickly become inefficient or even infeasible. This

*This work was supported by the Future & Emerging Technologies unit of the European Commission through project "BISON: Biology-Inspired techniques for Self Organization in dynamic Networks" (IST-2001-38923) and by the Swiss Hasler Foundation through grant DICS-1830.

means that routing information should be updated more regularly than for instance in wired networks. However, this can be a problem in MANETs, since the bandwidth of the network is limited by the fact that the wireless medium is shared: nodes can only send or receive data if no other node is sending in their immediate neighborhood (see[2] for a theoretical analysis of this problem). The access to the shared channel is controlled by protocols at the Medium Access Control layer (MAC), such as ANSI/IEEE 802.11 DCF[3] (the most commonly used in MANETs), which create extra overhead, lowering the effective available bandwidth.

Many MANET routing algorithms have been proposed.[4] These algorithms deal with the dynamic aspects of MANETs in their own way, using reactive or proactive behavior or a combination of both. *Reactive behavior* means that an algorithm only gathers routing information in response to an event, usually one which triggers the need for new routes, such as the start of a data session or the failure of an existing route. *Proactive behavior* means that the algorithm also gathers information at other times, so that routing information is readily available when the event happens.

In this work we present *AntHocNet*, a MANET routing algorithm based on the framework of *Ant Colony Optimization* (ACO)[5]. ACO reverse-engineers the pheromone laying-following behavior of ants, which allows the colony as a whole to find a shortest path between the nest and a food source.[6] For wired networks, a number of successful ACO routing algorithms exist (e.g. ABC[7] and AntNet[8]). The main idea is to repeatedly sample paths with small control packets, called *ants*, in order to adaptively estimate the quality of each local routing choice. This results in a collective and distributed learning of routing tables. ACO routing algorithms exhibit some desirable properties for MANETs: they work in a distributed way, are highly adaptive and robust, and provide automatic load balancing. AntHocNet is an attempt to create an ACO routing algorithm which works efficiently in MANETs, combining reactive path finding and repairing with proactive path maintenance and improvement. AntHocNet also features the use of multiple paths and stochastic data load spreading. The AntHocNet algorithm presented here is a revised and improved version of the algorithms described in[9,10]. The differences mainly concern proactive path maintenance and improvement, for which we use for the first time in ACO a combination of ant-based path sampling and information bootstrapping (see 3.2).

This paper is organized as follows. Section 2 presents related work in MANET and ACO routing. In section 3 our algorithm is described in detail and in section 4 it is compared to a state-of-the-art algorithm in a number of simulation studies.

2. Related work

This section contains an overview of some of the literature which is related to the work presented in this paper. Subsection 2.1 provides a short introduction to the vast amount of research in the area of MANET routing. Subsection 2.2 contains a description of ACO routing and of some of its earlier implementations in MANETs.

2.1. *Routing in MANETs*

In the MANET literature, the classical distinction is between table-driven, demand-driven and hybrid algorithms.[1] *Table-driven* algorithms, such as e.g. DSDV[11], are purely proactive: all nodes try to maintain routes to all other nodes at all times. This means that they need to keep track of all topology changes, which can become difficult if there are a lot of nodes or if they are very mobile. *Demand-driven* algorithms, such as AODV[12] and DSR[13], are purely reactive: nodes only gather routing information when a data session to a new destination starts, or when a route which is in use fails. Reactive algorithms are in general more scalable[14] since they greatly reduce the routing overhead, but they can suffer from oscillations in performance because they are never prepared for disruptive events. In practice, many algorithms are *hybrid algorithms* (e.g. ZRP[15]), using both proactive and reactive components in order to try to combine the best of both worlds.

AntHocNet is a hybrid routing algorithm, using ant agents to combine a reactive path setup phase with proactive path improvement efforts based on *bootstrapping* techniques. Bootstrapping is a characteristic of dynamic programming (see[16]), in which nodes calculate the estimated quality of a path based on estimates made by neighboring nodes. This is the typical mode of operation in Bellman-Ford routing algorithms (e.g. see[17]), to which table-driven algorithms like DSDV belong, and some reinforcement learning inspired approaches to routing, like SAMPLE[18]. The idea of continuously improving existing paths is quite rare in MANETs, although it has been explored to some extent in[19]. On the other hand, the use of multiple paths for data forwarding is an idea which has received a lot of attention in the MANET literature (see[20] for an overview). Very few algorithms however try to keep an updated estimate of the path qualities, and adapt their data load spreading to this.[21] Stochastic data spreading is to the best of our knowledge unexplored outside the area of ACO routing algorithms.

2.2. *ACO routing*

The basic idea behind ACO algorithms for routing[22,5] is the acquisition of routing information through path sampling using ant agents. These lightweight agents are generated concurrently and independently by the nodes, with the task to try a path to an assigned destination. An ant going from source s to destination d collects information about the quality of the path (e.g. end-to-end delay and number of hops), and, retracing its way back from d to s, uses this information to update the routing tables at intermediate nodes. Ants always sample complete paths.

The routing tables, called *pheromone tables*, contain for each destination a vector of real-valued entries, one for each known neighbor node. These entries are a measure of the goodness of going over that neighbor on the way to the destination. These pheromone variables are continually updated according to the quality of the paths sampled by the ants. The repeated and concurrent generation of ants results in the availability at each node of a bundle of paths, each with an estimated

measure of quality. In turn, the ants use the pheromone tables to find their way to their destination: at each node they stochastically choose a next hop, giving higher probability to those next hops which are associated with higher pheromone values.

Data packets are routed more or less in the same way as ants: packets are routed *stochastically*, choosing with a higher probability those links associated with higher pheromone values. This way data for a same destination are adaptively spread over *multiple paths* (but with a preference for the best paths), resulting in *load balancing*. For data packets, mechanisms are usually adopted to avoid low quality paths, while ants are more explorative, so that also less good paths are occasionally sampled and maintained as *backup paths* in case of failure or sudden congestion. In this way *path exploration* is kept separate from the use of paths by data. If enough ants are sent to the different destinations, nodes can keep up-to-date information about the best paths, and automatically adapt their data load spreading to this.

Ant-based routing algorithms have a number of properties which are desirable in MANETs: they are highly adaptive to network changes, use active path sampling, are robust to agent failures, provide multipath routing, and take care of data load spreading. However, the fact that they crucially rely on repeated path sampling can cause significant overhead if not dealt with carefully. There have been a number of attempts to design ant-based routing algorithms for MANETs. Examples are ARA[23], PERA[24], ANSI[25] and Termite[26]. In general, however, most of all these algorithms move quite far away from the original ACO routing ideas trying to obtain the efficiency needed in MANETs, and many of them are not very different from single-path on-demand algorithms.

3. Description of the algorithm

AntHocNet is a multipath routing algorithm consisting of both reactive and proactive components. The algorithm is reactive in the sense that nodes only establish paths when there is a need for them. When a communication session is started at a source node s, this node starts a *reactive path setup* phase, in which ant agents called *reactive forward ants* are spread over the network in order to find the destination d of the session. They follow existing routing information if available, and are otherwise broadcast. The first ant to find d becomes a *reactive backward ant* which returns to s, setting up entries in the pheromone tables of intermediate nodes indicating a path between s and d. They indicate in a node the quality of going over a certain next hop to reach a certain destination. During the duration of a communication session, the algorithm tries to *proactively improve the existing paths*. To facilitate this process, nodes include pheromone information they have about *active destinations* in their *hello messages*. A node is considered an active destination if data have recently been forwarded to it. Hello messages are short messages broadcast to all neighbors at regular intervals. The pheromone information forwarded from node to node in hello messages spreads over the network in a process which we call *pheromone diffusion*, in analogy with the volatile and diffusive character of ant

pheromone in nature (e.g. see[27]). The pheromone diffusion creates a field indicating possible paths to the destinations. This field, which is built up of bootstrapped information, contains potentially unreliable information, and it is important to verify it before it can be used safely by data. To this end, the source node of each communication session periodically sends out *proactive forward ants* during the course of the session, which follow the diffused pheromone in order to find new and better paths. In this way, the single path which is set up during the route setup phase is extended to a mesh of multiple paths. *Data are spread* concurrently across these paths, with a strong preference for the best path. *Link failures* are dealt with either by using local route repair or by warning preceding nodes on the paths.

3.1. *Reactive path setup*

When a source node s starts a communication session with a destination node d, and it does not have routing information for d available, it broadcasts a reactive forward ant F_d^s. Due to this initial (and further) broadcasting, different instances of the same original ant will travel through the network, and we will refer to the set of ants which originated from the same initial ant as an *ant generation*. The task of the ants of one generation is to find a path connecting s and d. At each node, an ant is either unicast or broadcast, according to whether or not the current node has routing information for d. The routing information of a node i is represented in a pheromone table $\mathcal{T}^i$. The entry $\mathcal{T}_{nd}^i \in \mathbb{R}$ of this table is the pheromone value indicating the estimated goodness of going from i over neighbor n to reach destination d. If pheromone information is available, the ant chooses its next hop n with the probability P_{nd}:

$$P_{nd} = \frac{(\mathcal{T}_{nd}^i)^{\beta_1}}{\sum_{j \in \mathcal{N}_d^i} (\mathcal{T}_{jd}^i)^{\beta_1}}, \quad \beta_1 \geq 1, \tag{1}$$

where $\mathcal{N}_d^i$ 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 (although in current experiments β_1 is kept to 1).

If an ant arrives in a node where there is no pheromone information available for d, it is broadcast. Due to this broadcasting, an ant generation can proliferate quickly over the network, with different ant instances following different paths to the destination. If an ant arrives in a node which was already visited by a different ant of the same generation, it is discarded, and at the destination d only the first ant of a generation is processed. This way, only one path is set up. In previous versions of AntHocNet more than one reactive forward ant could be accepted at the destination, creating a mesh of multiple paths in this phase.[9,10] But for efficiency reasons (due to the overhead involved in the parallel forwarding of multiple ant instances) it turned out to be better to construct only one path during the reactive setup phase and leave the creation of multiple paths to the proactive path maintenance and improvement phase (see subsection 3.2).

Each forward ant keeps a list $\mathcal{P} = [1, 2, \ldots, d]$ of the nodes it has visited. Upon arrival at the destination d, it is converted into a *backward ant*, which travels back to the source retracing $\mathcal{P}$. At each intermediate node $i \in \mathcal{P}$ $(i < d)$, the backward ant calculates the local estimate $\hat{T}^i_{i+1}$ of the time it takes to reach the neighbor $i+1$ the ant is coming from. This local estimate is used to incrementally compute an estimate of the time $\hat{T}^i_d$ it would take a data packet to reach d from i over $\mathcal{P}$, which is in turn used to update the pheromone tables:

$$\hat{T}^i_d = \sum_{i \leq j < d,\ j \in \mathcal{P}} \hat{T}^j_{j+1} \ . \tag{2}$$

The value of the estimate $\hat{T}^i_{i+1}$ is defined as the product of the estimate of the average time to send one packet, $\hat{T}^i_{mac}$, and the current number of packets in queue (plus one, for the current ant packet) to be sent at the MAC layer, Q^i_{mac}:

$$\hat{T}^i_{i+1} = (Q^i_{mac} + 1)\hat{T}^i_{mac} \ . \tag{3}$$

$\hat{T}^i_{mac}$ is calculated as a running average of the time elapsed between the arrival of a packet at the MAC layer and the end of a successful transmission. So if t^i_{mac} is the time it took to send a packet from node i, then node i updates its estimate as:

$$\hat{T}^i_{mac} = \alpha \hat{T}^i_{mac} + (1 - \alpha) t^i_{mac}, \tag{4}$$

with $\alpha \in [0, 1]$. Since $\hat{T}^i_{mac}$ is calculated at the MAC layer it includes channel access activities, so it takes into account local congestion of the shared medium.

At each intermediate node $i \in \mathcal{P}$, the backward ant sets up a path towards the destination d, creating or updating the pheromone table entry $\mathcal{T}^i_{nd}$ in $\mathcal{T}^i$. The pheromone value in $\mathcal{T}^i_{nd}$ represents a running average of the inverse of the cost, in terms of both estimated time and number of hops, to travel from i to d through n. If $\hat{T}^i_d$ is the travelling time estimated by the ant, and h is the number of hops, the value τ^i_d used to update the running average is defined as:

$$\tau^i_d = \left(\frac{\hat{T}^i_d + hT_{hop}}{2} \right)^{-1}, \tag{5}$$

where T_{hop} is a fixed value representing the time to take one hop in unloaded conditions (e.g. for the 2 Mbps experiments, T_{hop} was set to 0.003 sec). Defining τ^i_d like this is a way to avoid possibly large oscillations in the time estimates gathered by the ants (e.g., due to local bursts of traffic) and to take into account both end-to-end delay and number of hops. The value of $\mathcal{T}^i_{nd}$ is updated as follows:

$$\mathcal{T}^i_{nd} = \gamma \mathcal{T}^i_{nd} + (1 - \gamma)\tau^i_d, \quad \gamma \in [0, 1]. \tag{6}$$

Using this formula, pheromone has the dimension of an inverted time. In the experiments, the parameters γ and α were both set to 0.7.

Once the backward ant makes it back to the source, a full path is set up and the source can start sending data. If the backward ant for some reason does not arrive, a timer (set to 1 second in the experiments) will run out at the source, and the whole process is started again.

3.2. *Proactive path maintenance and exploration*

During the course of a communication session, a source node proactively updates the information about the currently used paths to the destination, and tries to find new and better paths. An important role in this process is played by *hello messages*. These are short messages broadcast every t_{hello} seconds (in our case $t_{hello} = 1sec$) by all nodes. This kind of messages are used in many existing protocols (see e.g.[28,29]), to allow nodes to figure out which are their immediate neighbors. When a node i receives a hello message from a new node n, it can assume that n is its neighbor. After that, i expects to receive a hello message from n every t_{hello} seconds. After missing a certain number of expected hello's (2 in our case), i assumes that n has moved out of range, and no longer considers it a neighbor.

In AntHocNet, nodes include in the hello messages they send out routing information they have about active destinations (also in some other algorithms hello messages are used to convey extra routing information[29]). A node n constructing a hello message consults its pheromone table, and picks a maximum number k (set to 10 in the experiments) of destinations it has routing information for (if more than k active destinations are available, k of them are picked randomly). For each one of these destinations d, the hello message contains the address of d and the best pheromone value $\mathcal{T}^n_{m^*d}, m^* \in \mathcal{N}^n_d$, which n has available for d.

A node i receiving the hello message from n first of all registers n as a neighbor in its neighbor table and adds a one-hop route to n in its routing table. Then it goes through the list of destinations reported in the hello message. For each listed destination d, it uses the received pheromone value $\mathcal{T}^n_{m^*d}$ to build up a new estimate for the goodness of going from i to d over n by adding the cost of hopping from i to n. We call this new estimate the *bootstrapped pheromone* $\mathcal{B}^i_{nd}$, since it is built up using an estimate which is non-local to i. To calculate $\mathcal{B}^i_{nd}$, we first invert $\mathcal{T}^n_{m^*d}$ (since pheromone has the dimensions of an inverted time), and then add the estimated time to perform the single hop from i to n, which we calculate like the term between brackets in equation 5:

$$\mathcal{B}^i_{nd} = \left((\mathcal{T}^n_{m^*d})^{-1} + \frac{\hat{T}^i_n + T_{hop}}{2} \right)^{-1} \tag{7}$$

What i does with the bootstrapped pheromone information $\mathcal{B}^i_{nd}$ depends on the information about d and n in i's routing table. If i has a pheromone entry $\mathcal{T}^i_{nd}$ for destination d going over neighbor n, this means that there is a path from i to d over n which has been sampled completely by one ant, and can therefore be considered reliable. So i treats $\mathcal{B}^i_{nd}$ as an update of the goodness estimate of this existing path, and replaces $\mathcal{T}^i_{nd}$ by $\mathcal{B}^i_{nd}$. This way it keeps pheromone on current paths up-to-date.

If, on the other hand, i did not have a value for $\mathcal{T}^i_{nd}$, the bootstrapped pheromone can indicate a possible new path from i over n to d. If i would then be allowed to include $\mathcal{B}^i_{nd}$ in its routing table as a new entry, and add it in its own hello messages for further broadcasting, pheromone information could be spread over the

whole network. This forwarded pheromone information would be for a large part based on pure bootstrapping though (estimates based on other node's estimates), and could therefore be unreliable. Combining old and new routing information in bootstrapping systems can easily lead to loops for example. This is quite likely in our case here, since the MANET environment changes quickly, and nodes only forward bootstrapped information in periodic hello messages. Therefore, i should not use $\mathcal{B}^i_{nd}$ to update the regular pheromone table for data routing, since it is potentially wrong: it needs to be checked before being used. So $\mathcal{B}^i_{nd}$ is stored in a second pheromone table $\mathcal{V}^i$ at i, called *virtual pheromone table*. The information of $\mathcal{V}^i$ can in turn be forwarded further in hello messages as bootstrapped pheromone. In that case however, a bit is set in the entry in the hello message to indicate that this pheromone should not be used to update the pheromone estimate of existing paths as described above (so that virtual and sampled paths do not mix). Using this system, virtual pheromone can be spread from node to node without causing too much overhead, creating over the whole MANET a field of virtual pheromone. The paths indicated by this field are checked using *proactive ants*.

Each node which is the source of a communication session periodically (every t_{hello} seconds) compares the virtual and regular pheromone information it has available for the destination of each of its session. If the best virtual pheromone is significantly better (in the experiments: at least 10% better) than the best regular pheromone, a *proactive forward ant* is sent out. Proactive forward ants are unicast towards the destination, using the same probabilistic routing rule as reactive forward ants (but with the same routing exponent as data, see eq. 8), but considering both regular and virtual pheromone (while data only follow regular pheromone). Unlike reactive ants, however, proactive ants are never broadcast, and when they arrive at a place where there is no pheromone available they are just discarded. When a proactive ant arrives at the destination, a backward ant identical to the ones used during the reactive path setup is sent back to the source. This way, promising virtual pheromone is investigated, and if the investigation is successful it is turned into a regular path which can be used for data. This increases the number of paths available for data routing, which slowly grows to a full mesh, and allows the routing algorithm to exploit new routing opportunities in the ever changing topology.

While the reactive path setup phase described in subsection 3.1 is similar to common practices in traditional reactive routing algorithms, the forwarding of bootstrapped routing information over the whole network reminds of the typical mode of operation in proactive algorithms like DSDV, which is based on Bellman-Ford routing algorithms for wired networks. The problem with such algorithms is that in general they are only guaranteed to work correctly if all the routing tables are up-to-date. In dynamic MANETs, this means that nodes should send a lot of routing updates around, which can quickly congest the network. In the approach we take, the proactive part is kept lightweight, forwarding the bootstrapped information in a delayed fashion, and no guarantees about correctness are given. Therefore we cannot use the proactive information for data routing, and instead we use it as a

guideline for finding new and better paths. The proactive ants form a link between the unreliable proactive information and trusted established data forwarding paths.

From an ACO routing point of view, we have decoupled path maintenance from path discovery and improvement. While initial paths are set up using ants which can be broadcast or follow pheromone (see subsection 3.1), the updating of pheromone on existing paths is no longer done by repeated path sampling with ants, but with a lightweight bootstrapping process which can dramatically reduce the amount of overhead in MANETs. Exploring new paths, on the other hand, is again done using ants. They are guided however by the virtual pheromone field set up using bootstrapping. The bootstrapped pheromone field can be seen as the result of natural diffusion of previously placed pheromone.

3.3. *Stochastic data routing*

The path setup phase together with the proactive path improvement actions creates a mesh of good paths between source and destination, indicated in the pheromone tables of the nodes. Data are forwarded according to the values of the pheromone entries. Nodes in AntHocNet *forward data stochastically*. When a node has multiple next hops for the destination d of the data, it randomly selects one of them, with probability P_{nd}. P_{nd} is calculated in the same way as for the reactive forward ants, but with a much higher exponent, in order to be greedy with respect to the better paths:

$$P_{nd} = \frac{(\mathcal{T}^i_{nd})^{\beta_2}}{\sum_{j \in \mathcal{N}^i_d} (\mathcal{T}^i_{jd})^{\beta_2}}, \quad \beta_2 \geq \beta_1 \ . \tag{8}$$

In the experiments, β_2 was set to 20. Setting the routing exponent so high means that if several paths have similar quality, data will be spread over them. However, if one path is clearly better than another, it will almost always be preferred. According to this strategy, we do not have to choose a priori how many paths to use: their number will be automatically selected in function of their quality.

The probabilistic routing strategy leads to data load spreading according to the estimated quality of the paths. If the estimates are kept up-to-date (which is done using the bootstrapped information from the hello messages as described in subsection 3.2), this leads to *automatic load balancing*. When a path is clearly worse than others, it will be avoided, and its congestion will be relieved. Other paths will get more traffic, leading to higher congestion, which will make their end-to-end delay increase. By adapting the data traffic, the nodes try to spread the data load evenly over the network.

3.4. *Link failures*

Nodes can detect link failures (e.g., a neighbor has moved far away) when unicast transmissions (of data packets or ants) fail, or when expected hello messages were not received (see subsection 3.2). When a neighbor is assumed to have disappeared,

the node takes a number of actions. In the first place, it removes the neighbor from its neighbor list and all the associated entries from its routing table.

Further actions depend on the event which was associated with the discovered disappearance. If the event was a failed transmission of a control packet, the node broadcasts a *link failure notification* message. Such a message contains a list of the destinations to which the node lost its best path, and the new best estimated end-to-end delay and number of hops to this destination (if it still has entries for the destination). All its neighbors receive the notification and update their pheromone table using the new estimates. If they in turn lost their best or their only path to a destination due to the failure, they will broadcast the notification further, until all concerned nodes are notified of the new situation.

If the event was the failed transmission of a data packet, the node does not include the destination of the data packet in question in the link failure notification. For this destination, the node starts a *local route repair*. The node broadcasts a *route repair ant* that travels to the involved destination like a reactive forward ant: it follows available routing information when it can, and is broadcast otherwise. One important difference is that it has a maximum number of broadcasts (which we set to 2 in our experiments), so that its proliferation is limited. The node waits for a certain time (in the experiments set to 5 times the estimated end-to-end delay of the lost path), and if no backward repair ant is received by then, it concludes that it was not possible to find an alternative path to the destination. Packets which were in the meantime buffered for this destination are discarded, and the node sends a new link failure notification about the lost destination.

Link failure notifications keep routing tables on paths up-to-date about upstream link failures. However, they can sometimes get lost and leave dangling links. A data packet following such a link arrives in a node where no further pheromone is available. The node will then discard the data packet and unicast a warning back to the packet's previous hop, which can remove the wrong routing information.

4. Experimental results

We evaluate our algorithm in a number of simulation tests. We compare its performance with AODV (with route repair), a state-of-the-art MANET routing algorithm and a de facto standard. In 4.1 we describe the simulation environment and the test scenarios, and in 4.2 we show and discuss the results.

4.1. *Simulation Environment*

As simulation software, we use Qualnet, a commercial simulation package.[30] We run simulation tests on three different scenarios. All three scenarios are run for 900 seconds. Nodes are initially placed randomly on the surface area and then they move according to the random waypoint mobility model (RWP),[13] with speed ranging between 0 and 20 m/s (under RWP, nodes iteratively choose a random destination,

move there in a straight line with a randomly chosen speed, and then wait there for a certain pause time). The pause time is set differently in different experiments. Data traffic is generated by 20 constant bit rate (CBR) sources using UDP at the transport layer. The sources start sending at a random time between 0 and 180 seconds after the start of the simulation, and keep sending until the end. At the physical layer a two-ray signal propagation model is used. At the MAC layer we use the popular 802.11b DCF protocol, with only one channel for communications.

In the first scenario, 100 nodes move in an area of $3000 \times 1000\ m^2$. Each data source sends one 64-byte packet per second. The data rate at the physical layer is 2Mbit/s and the radio range 300 meters. We ran tests for different pause times, ranging between 0 and 480 seconds (under RWP, higher pause time means higher mobility). In the second scenario, we investigate what happens under more intense data traffic, using higher bandwidth: each data source sends eight 64-byte packets per second. The bandwidth of the wireless interface is raised to 11Mbit/s. Since higher bandwidth implies shorter radio ranges (now 110 meters), we reduce the area of the MANET: 100 nodes move in an area of $1000 \times 1000\ m^2$. We use the same pause times as before. In the third scenario, we investigate the scalability of the algorithm. We keep the same data rate and bandwidth as in the second scenario, and vary the number of nodes, ranging from 50 to 500. The MANET area is adapted accordingly, ranging from $750 \times 750\ m^2$ to $2250 \times 2250\ m^2$, to keep the node density constant on 1 node per 100 m^2. The pause time is kept constant on 30 seconds.

4.2. *Simulation Results*

In these tests, we measure the *average end-to-end delay* for data packets and the *ratio of correctly delivered versus sent packets*. These are standard measures of effectiveness in MANETs. We also report *delay jitter*, the average difference in inter-arrival time between packets. This is an important metric in quality-of-service networks and also provides a measure of the stability of the algorithm's response to topological changes in MANETs. Finally, we consider *routing overhead*, as measure of efficiency. We calculate it as the number of control packet transmissions (counting every hop) per data packet delivered. In all tests, each reported data point represents an average over 10 randomly generated initial node placements. The parameters of AntHocNet were set to the values mentioned in section 3 for all test settings. These values were obtained from empirical experience, without tuning them separately for each considered scenario (the algorithm is quite robust with respect to the setting of most parameters). Figures 1-3 show the average end-to-end delay and the delivery ratio for AntHocNet and AODV in each of the three different scenarios, while table 1 reports the average delay jitter for both algorithms in all three scenarios, and table 2 reports the overhead.

In the first scenario, with 100 nodes and light traffic load (figure 1 and tables 1a and 2a), AntHocNet shows an average end-to-end delay which is about half that of AODV for low pause times, and around one third for high pause times. In terms

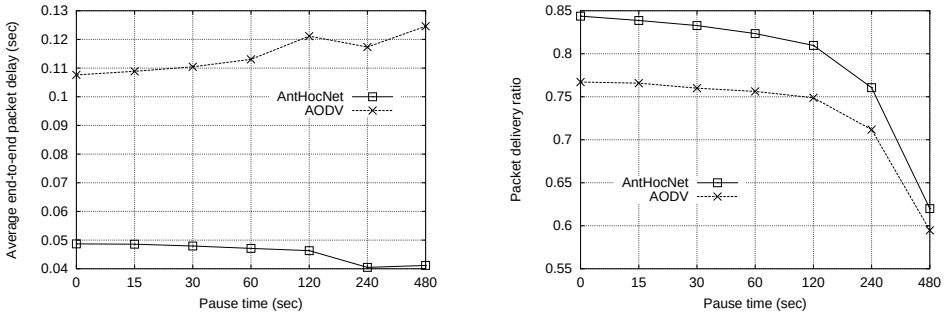

Fig. 1. Average delay and delivery ratio for various pause times in a scenario with 100 nodes in a $3000 \times 1000\ m^2$ area under light traffic load (20 sources sending 1 packet per second)

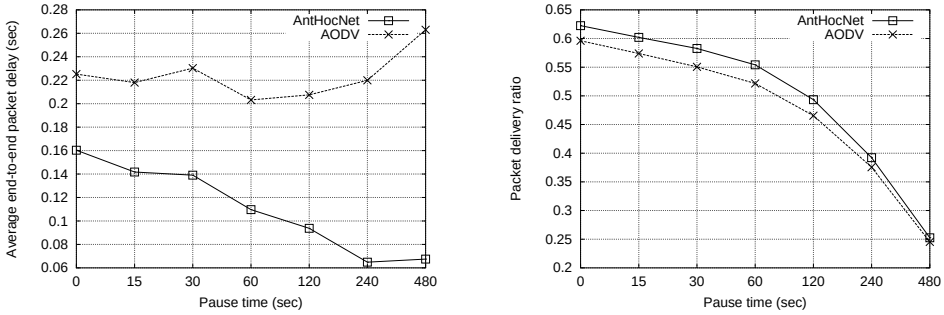

Fig. 2. Average delay and delivery ratio for various pause times in a scenario with 100 nodes in a $1000 \times 1000\ m^2$ area under heavy traffic load (20 sources sending 8 packets per second)

of delivery ratio, the difference is less striking but still significant. The drop in delivery ratio for the highest pause times is due to connectivity issues. The network we consider is quite sparse, so that nodes can become disconnected from the network for a certain time. For the highest pause times, these periods of lost connectivity, in which no packets can be delivered, can be long. When we look at delay jitter, we can again see a large advantage of AntHocNet over AODV. The better performance of AntHocNet is partly paid for with more overhead, although the difference is quite small. For both jitter and overhead, the loss of connectivity for high pause times has a strong effect. In terms of overhead, this effect is less strong for AODV since AntHocNet waits for a shorter time before retrying a path setup in case the previous attempt failed (see subsection 3.1).

In the second scenario, with 100 nodes and heavier traffic load (figure 2 and tables 1b and 2b), we can see the same trends. The difference in delay is smaller for low pause times, but for high pause times AntHocNet's average delay is again three times lower than AODV's. The drop in AntHocNet's average delay is likely due to

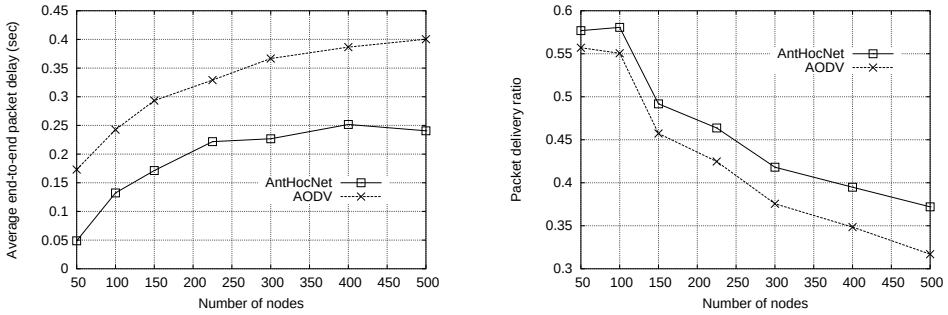

Fig. 3. Average delay and delivery ratio for an increasing number of nodes (and increasing area keeping node density constant) under heavy traffic load (20 sources sending 8 packets per second)

Table 1. Average delay jitter in the different scenarios: table (a) refers to the scenario with 100 nodes and light traffic load, table (b) to the one with 100 nodes and heavy traffic load, and table (c) to the scenario with increasing number of nodes

(a)

Pause	AntHocNet	AODV
0	0.439	0.754
15	0.452	0.759
30	0.472	0.794
60	0.504	0.822
120	0.536	0.851
240	0.695	1.009
480	1.498	1.754

(b)

Pause	AntHocNet	AODV
0	0.265	0.271
15	0.272	0.286
30	0.277	0.304
60	0.278	0.306
120	0.345	0.377
240	0.500	0.566
480	0.853	0.894

(c)

Nodes	AntHocNet	AODV
50	0.233	0.258
100	0.280	0.304
150	0.385	0.430
225	0.441	0.510
300	0.550	0.616
400	0.605	0.686
500	0.723	0.890

Table 2. Overhead in the different scenarios, with tables referring to scenarios like in table 1

(a)

Pause	AntHocNet	AODV
0	23.52	20.69
15	24.11	20.65
30	24.24	20.42
60	24.21	19.9
120	24.63	19.42
240	26.69	18.43
480	36.72	21.12

(b)

Pause	AntHocNet	AODV
0	6.66	5.83
15	6.77	5.69
30	6.83	5.58
60	6.72	5.27
120	7.02	4.98
240	7.83	4.69
480	11.66	6.10

(c)

Nodes	AntHocNet	AODV
50	3.01	2.31
100	6.55	5.58
150	12.14	11.09
225	20.19	20.30
300	30.26	31.32
400	44.99	48.81
500	58.45	66.36

the proactive path improvements in combination with the fact that pheromone is based on path delay, which is more important in more congested traffic conditions. In terms of delivery ratio and jitter, the difference between both algorithms is smaller, although AntHocNet retains an advantage. In terms of overhead, the difference is more or less the same as before.

The third scenario (figure 3 and tables 1c and 2c) shows the scalability of AntHocNet: with increasing network size, the differences in terms of average end-to-end delay, delivery ratio and jitter grow. More importantly, while AntHocNet has a slightly higher overhead than AODV for the small scenarios, it actually generates less overhead for the larger scenarios. This is an indication that AntHocNet can

scale better with respect to the number of nodes than AODV. It seems that in larger MANETs AntHocNet's mechanisms of path maintenance, path improvement and local repair pay off more.

5. Conclusions

We have described AntHocNet, an ACO routing algorithm for MANETs. AntHocNet combines a reactive path setup phase with proactive path maintenance and improvement, and also makes use of a local repair mechanism. In the proactive behavior, ant based sampling is combined with a lightweight information bootstrapping mechanism to obtain an efficient and reliable process. In a series of simulation experiments we compare AntHocNet with AODV, a reactive algorithm which is an important reference. We show that AntHocNet can outperform AODV in terms of delivery ratio, average delay and jitter, without causing much more overhead. AntHocNet also seems more scalable than AODV: increasing the number of nodes its performance advantage increases and its overhead grows slower than AODV's.

In future work, we plan to use more complex metrics as pheromone, taking into account more than path delay and number of hops. In particular, we think it is important to include information about the quality of wireless links. This could be obtained from information about the relative position and movement of nodes, and from measures taken at the physical and MAC layer. Another aspect we plan to work on is the reliability of the bootstrapping process, e.g. by including a mechanism of destination sequence numbers like in DSDV.

References

1. E.M. Royer and C.-K. Toh, A review of current routing protocols for ad hoc mobile wireless networks, *IEEE Personal Communications* **6** (1999) 46–55.
2. P. Gupta and P.R. Kumar, The capacity of wireless networks, *IEEE Transactions on Information Theory* **46** (2) (2000).
3. IEEE 802.11 working group, ANSI/IEEE std. 802.11, 1999 edition: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications, Technical report, ANSI/IEEE, (1999).
4. M. Abolhasan, T. Wysocki and E. Dutkiewicz, A review of routing protocols for mobile ad hoc networks, *Ad Hoc Networks* **2** (2004) 1–22.
5. M. Dorigo, G. Di Caro and L.M. Gambardella, Ant algorithms for distributed discrete optimization, *Artificial Life* **5** (1999) 137–172.
6. S. Camazine, J.-L. Deneubourg, N. Franks, J. Sneyd, G. Theraulaz and E. Bonabeau, *Self-Organization in Biological Systems*, (Princeton University Press, 2001).
7. R. Schoonderwoerd, O. Holland, J. Bruten and L. Rothkrantz, Ant-based load balancing in telecommunications networks, *Adaptive Behavior* **5** (1996) 169–207.
8. G. Di Caro and M. Dorigo, AntNet: Distributed stigmergetic control for communications networks, *Journal of Artificial Intelligence Research (JAIR)* **9** (1998) 317–365.
9. G. Di Caro, F. Ducatelle and L.M. Gambardella, AntHocNet: an ant-based hybrid routing algorithm for mobile ad hoc networks, in *Proceedings of Parallel Problem Solving from Nature (PPSN VIII)*, LNCS 3242 (Springer-Verlag, 2004).

10. F. Ducatelle, G. Di Caro and L.M. Gambardella, Ant agents for hybrid multipath routing in mobile ad hoc networks, in *Proceedings of The Second Annual Conference on Wireless On demand Network Systems and Services (WONSS)*, January 2005.
11. C. Perkins and P. Bhagwat, Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers, in *Proceedings of ACM SIGCOMM'94 Conference on Communications Architectures, Protocols and Applications*, 1994, 234–244.
12. C.E. Perkins and E.M. Royer, Ad-hoc on-demand distance vector routing, in *Proc. of the Second IEEE Workshop on Mobile Computing Systems and Applications*, 1999.
13. D.B. Johnson and D.A. Maltz, Dynamic Source Routing in Ad Hoc Wireless Networks, in *Mobile Computing* Chapter 5 (Kluwer, 1996), 153–181.
14. J. Broch, D.A. Maltz, D.B. Johnson, Y.-C. Hu and J. Jetcheva, A performance comparison of multi-hop wireless ad hoc network routing protocols, in *Proc. of the* 4^{th} *Annual ACM/IEEE Int. Conf. on Mobile Computing and Networking*, 1998.
15. Zygmunt J. Haas, A new routing protocol for the reconfigurable wireless networks, in *Proc. of the IEEE Int. Conf. on Universal Personal Communications*, 1997.
16. R.S. Sutton and A.G. Barto, *Reinforcement Learning* (MIT Press, 1998).
17. D. Bertsekas and R. Gallager, *Data Networks* (Prentice–Hall, 1992).
18. J. Dowling, E. Curran, R. Cunningham and V. Cahill, Collaborative Reinforcement Learning of Autonomic Behaviour, in *Proc. of the 2nd Int. Workshop on Self-Adaptive and Autonomic Computing Systems*, 2004.
19. C. Gui and P. Mohapatra, SHORT: Self-healing and optimizing routing techniques for mobile ad hoc networks, in *Proceedings of MobiHoc*, 2003.
20. S. Mueller, R. Tsang and D. Ghosal, Multipath routing in mobile ad hoc networks: Issues and challenges, *Performance Tools and Applications to Networked Systems*, LNCS 2965 (Springer-Verlag, 2004).
21. D. Ganesan, R. Govindan, S. Shenker and D. Estrin, Highly-resilient, energy-efficient multipath routing in wireless sensor networks, *Mobile Computing and Communications Review* **1** (2002).
22. G. Di Caro, *Ant Colony Optimization and its application to adaptive routing in telecommunication networks*, PhD thesis, Faculté des Sciences Appliquées, Université Libre de Bruxelles, (Brussels, Belgium, 2004).
23. M. Günes, U. Sorges and I. Bouazizi, ARA - The ant-colony based routing algorithm for MANETs, In *Proc. of the Int. Workshop on Ad Hoc Networks (IWAHN)*, 2002.
24. J.S. Baras and H. Mehta, A probabilistic emergent routing algorithm for mobile ad hoc networks, in *Proceedings of WiOpt03: Modeling and Optimization in Mobile, Ad Hoc and Wireless Networks*, 2003.
25. C.-C. Shen, C. Jaikaeo, C. Srisathapornphat, Z. Huang and S. Rajagopalan, Ad hoc networking with swarm intelligence, in *Ants Algorithms - Proceedings of ANTS 2004*, LNCS 3172, (Springer-Verlag, 2004).
26. M. Roth and S. Wicker, Termite: Ad-hoc networking with stigmergy, in *Proceedings of Globecom*, 2003.
27. R.W. Mankin, R.T. Arbogast, P.E. Kendra and D.K. Weaver, Active spaces of pheromone traps for *Plodia interpunctella* in enclosed environments, *Environmental Entomology* **28** (1999) 557–565.
28. C.-K. Toh, Associativity-based routing for ad-hoc mobile networks, *Wireless Personal Communications* **4** (2) (1997) 1–36.
29. T. Clausen, P. Jacquet, A. Laouiti, P. Muhlethaler, A. Qayyum and L. Viennot, Optimized link state routing protocol, in *Proceedings of IEEE INMIC*, 2001.
30. Scalable Network Technologies, Inc., Culver City, CA, *Qualnet Simulator, Version 3.6* (2003), `http://stargate.ornl.gov/trb/tft.html`.