Eidgenössische Technische Hochschule Zürich
Ecole polytechnique fédérale de Zurich
Politecnico federale di Zurigo
Swiss Federal Institute of Technology Zurich

Departement D-ITET

Distributed Computing Group

Semester Thesis

Ant Algorithm

Claudia Lorenz
lorenzc@ee.ethz.ch

Patrizia Mottl
mottlp@ee.ethz.ch

Prof. Dr. Roger Wattenhofer
Distributed Computing Group

Advisor: Roland Flury

Distributed Computing Group

Contents

Chapter 1

Abstract

In this report we present a routing algorithm for mobile ad hoc networks (MANETs). The algorithm uses techniques of route discovery that was observed by ants. FAnts walk randomly around the network to find the target. These FAnts leave their track in writing routing table entries in each node they pass. If the target is found, another ant (BAnt) can walk back along this route and also writes the routing table entries. When the BAnts reaches the source the routing table of all nodes between the source and the destination carry the necessary routing information and data packets can be sent.
The routing algorithm based on ants was developed by G. Di Carlo and M. Dorigo [3] and M. Günes, U. Sorges and I. Bouazizi in [7] and further discussed in [6]. However, none of these works investigated the problem of mobile networks, where nodes change their position over time. In such a mobile network, some nodes may be connected during route discovery, but are disconnected when the data should be transferred. If this happen a mechanism called Route Maintenance will start to find the node the messages should be sent to. That is how we guaranteed that the route is stable and doesn't break down.
Simulation results show that the total number of messages to find the target can be reduced compared to a basic flooding algorithm.

Chapter 2

Introduction

In this report we introduce a routing protocol we developed. It is based on ideas of different ant algorithms. In particular, we consider mobile ad hoc networks (MANETs), where the network nodes are able to change their position and the communication between the network nodes is established over a wireless medium. Also, we consider homogenous networks with no additional infrastructure. There is no difference between the nodes. This has among other things the following consequences:

- Nodes can leave and join the network at any time
- There is no centralized control or overview
- Packets have to be forwarded form node to node

Routing in MANETs is a challenge due to the fact that a good path can suddenly become an ineffcent or even an infeasible one. To succeed, a routing algorithm for such an environment needs to be adaptive and to be able to deal with sudden changes in the topology of the network. These properties can also be found in nature. Insect populations show us a robust and efficient way to adapt to the changing environment. This fact inspired us to design a routing algorithm based on simple biological agents, in our case ants.

2.1 An Ant Colony

It was observed that in a colony of ants the individuals are able to move over a shortest among different path, which connect their nest with a source of food.
The trick is the use of a volatile chemical substance called pheromone: while moving between the nest and a source of food they deposit pheromone. The ants choose their path in the direction of higher pheromone intensity. Shorter paths can be completed quicker and more frequently by the ants and therefore the intensity is higher for such routes and more ants are going to take this connection between the nest and the source of food.
First bio inspired routing protocols were designed for wired networks as for example AntNet for packet switching network [3]. In this protocol a node sends periodically an agent ("forward ant") to a randomly chosen target node. On its way the ant saves the IDs of the visited on his stack. At the target node an other agent ("backward ant") is routed back to the source node over the same way the forward ant took. On their way both agents update the routing table informations in the nodes. In the next section we have a look at two other protocols. In contrary to AntNet these two protocols are designed for mobile networks.

2.2 Existing Protocols

Quite a large number of routing algorithm for MANETs have been proposed. There exist several ways of dealing with the mobility of a network. The problem with mobility is that a found route is not valid forever, because the nodes move around and after some time two nodes may not be any more in the connection radius of each other.
In general three categories can be made up:

- reactive protocols: Information is only gathered in response to an event. Such a protocols tries to find a route form a source to a destination on-demand. The main advantage of reactive protocols is ceratinly the low overhead of control message. The main disadvantage is, that the route discovery has to deal with higher latency.

- proactive protocols: In a proactive protocol extensive routing tables are maintained for the entire network. As an advantage a route is found as soon it is requested and a low latency is the case. Clearly such a protocol need quite a high volume of control messages especially when the topology changes over time.

- hybrid protocols: A protocol having reactive and proactive components is named hybrid.

A list of a few different routing can be found for example in the following wiki [4].

2.2.1 Basic Ant Algorithm

We figured out that one basis almost all ant based algorithms have can be found in [6]. However, the authors use a more restrictive definition of mobile network: They only consider temporal changes due to link failure and congestion, but not the physical relocation of network nodes. Nevertheless the basic ideas can be refound in different papers.
Subramanian, Druschel and Chen introduce two different algorithms: the regular ant algorithm and the uniform ant algorithm. There are three different key ideas underlying the two algorithms:

- ants that explore the network: Periodically, each host generates an agent (ant) to another randomly chosen host. On its way, the ant updates a cost variable according to the cost C the taken route has and writes routing table entries in each node it visits. It will become clear in the third point that this costs are needed to update the routing tables.

- probabilistic routing tables: The forwarding tables are probabilistic. We have a look at a routing table entry for a certain target (T) each neighbor node is listed and rated with a probability. When a message arrives in this node for target T it is forwarded to a neighbor with this listed probability. How this probability is updated is showed in the next point.

- probabilistic updates of routing tables by ants: If a forwarding node (A) receives an ant from node B with source S, target T and cost C (the costs the ant had until it reached A), it updates the costs C by adding the ants cost to go from B to A and then updates the routing table entry for the source S. At last the probabilities are a function of the number of ants arrived over a link from a specific node: In our case after the ant arrived the probability to take the edge from A to B when S is the target is increased while the probability for the other is decreased.

The only difference between the uniform and the regular ant algorithm is in the behavior of the ants that explore the network. The uniform ants choose the next hop uniformly over the connected neighbors of the node it is in, while the regular ant algorithm chooses the next hop according to the probability entries in the routing table.

2.2.2 The Ant-Colony-Based Routing Algorithm for MANETs (ARA)

Another algorithm is the Ant-Colony-Based Routing Algorithm for MANETs [7]. This algorithm is built up in three phases:

- Route Discovery Phase: Forward Ants (FANTs) are flooded over the network. The routing table entry is based on the concept of a pheromone track. An entry for a certain node A contains all neighbors with a certain pheromone value, which is increased, if a FANT arrives from the specific neighbor and is decreased over time. Once a FANT reaches its target T backward ants (BANTs) are sent out. They are sent back over all edges with a pheromone value > 0. At the same time they make entries in the nodes for the target T of the communication.

- Route Maintenance: Once the FANTs and BANTs have established the pheromone tracks for the source and the destination node, subsequent data packets also increase the pheromone value.

- Route Failure Handling: ARA recongizes a route failure through a missing acknowledgement. In our eyes this is a quite a bad way of dealing with route failures. If a message has to take a long path towards a target it lasts long until the sender is able to realize that the packet was lost. With such an approach it is nearly impossible to route in dynamic networks.

After this short overview of the protocols that inspired us we now start with the description of our algorithm.

Chapter 3

Algorithm

3.1 Basics of the Algorithm

In this section we present the basic concepts of our algorithm. We started with the following reactive approach. If a node wants to establish a connection to another node, it sends out a forward ant (FAnt), that walks randomly in the network. What do we mean with "randomly"? In our algorithm randomly means that on each node the FAnt choses one of its neighbors with equal probability. While hopping from one node to the next the FAnt leaves a track by putting routing table entries in the nodes which point towards the source. It walks around randomly in the network, until it reaches the target it was sent out for. Once it reaches the target, a backward ant (BAnt) is generated, which follows the routing table entries back to the source. On its way the BAnt itself leaves entries in the routing table again for its source (the target of the connection).

3.1.1 Random Walk

Before elaborating the algorithm in detail we focussed on what it actually means to walk randomly in a network. The first and most important thing is certainly that the random walk on networks can be modeled as a Markov chain. This chain is ergodic and has a steady state. In [2] the authors show a way to compute the expected number of steps before absorption (hitting time). If one builds up the transition matrix in the following way:

$$P_{ij} = \begin{cases} 0 & \text{no connection between node i and j} \\ \frac{1}{deg(i)} & \text{if i and j are neighbours} \end{cases}$$

where deg(i) is the number of adjacent edges.

For the absorbing state (the target of the connection) the rule is different:

$$P_{ij} = \begin{cases} 1 & i = j \\ 0 & else \end{cases}$$

When indexing the nodes such that the target node gets index 0, we get a transistion matrix of the following type. Note that the vector r and the matrix Q are determined by the network according to the first rule for the transition matrix.

$$P = \begin{pmatrix} 1 & 0 \\ r & Q \end{pmatrix}$$

Then one can determine the hitting time for each starting state by

$$t \quad = \quad (I-Q)^{-1} \cdot \begin{pmatrix} 1 \\ 1 \\ . \\ . \\ 1 \end{pmatrix}$$

Another approach, which is explained also in [2] and in [1], is that a random walk on a network can be modelled by into an electrical network. The theory to understand that approach is quite broad and we think, that it is not relevant to know the theory exactly to understand the behavior of our algorithm.

Therefore, we only provide a few comments. As the algoritm relies on the number of edges respectively the connectivity of the network it is clear that bad cases with long hitting time can be generated. Such a case would be, if there is a certain edge, which has to be taken. For example: The source node lies in one section of the network and the target node in an other one. The only connection between this two section is one edge. If the FAnt has to reach the target it has to take this edge. This is a quite unfavorable situation for the FAnt. After this short discussion about hitting times we like to have a closer look at the behavior of the FAnts and BAnts and to elaborate the whole algorithm. Let us start with the FAnts.

3.1.2 FAnts

In the beginning of this section we were talking about one FAnt moving around. This was just to understand the basic concept. In real, we are sending out more than one FAnt. Let us first try to understand why and then have a look at how we are sending them out in our implementation. Just imagine we just sent out one FAnt. The quality of the route we are going to find will not be very well. If the FAnt has not taken any loops (we will discuss the prevention of loops in section 3.1.3) the BAnt going back will take the same route the FAnt took. This is quite probably a rather bad one. That is the reason, we send out more than one FAnt and with that most nodes near the source know a quite a short route to the source and the route the BAnt takes is not that bad. A comparision of the found route with the optimal will be done in section 4.1.4. We defined several parameters, the number of FAnts μ, that are sent out at the beginning. After some time we send out a new sequence of FAnts. Each time we sent out a new sequence of FAnts we first multiply the number of FAnt with a given factor α. The number of FAnts sent out is therefore

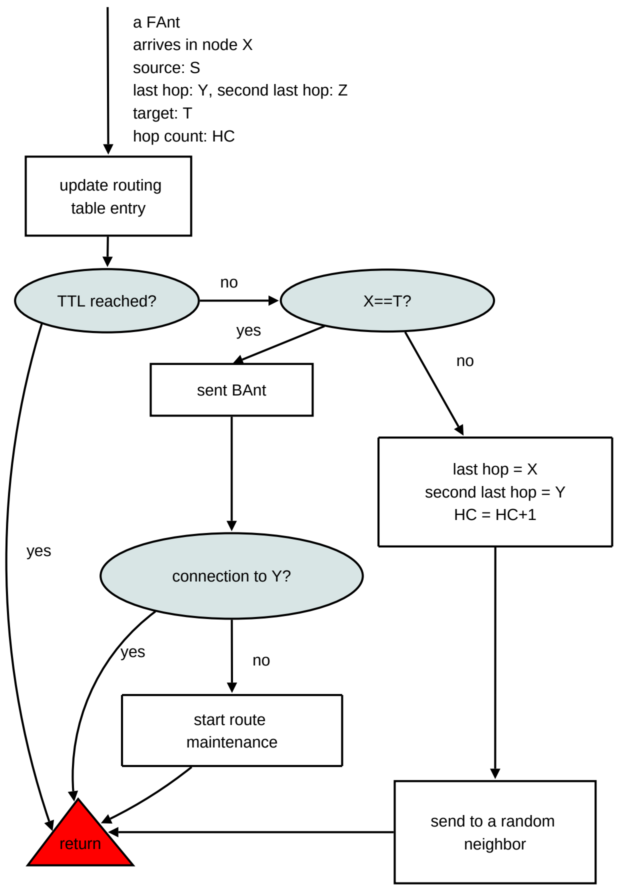

Figure 3.1: flow chart of the "FAnt-handling" in a node

exponentialy increasing. Another parameter is the TTL of the FAnts. We will discuss how we fixed this parameters in section 4.1.2.

Now we describe what a node has to do if it receives a FAnt. A flow chart of this can be found in figure 3.1 on page 8. Reaching a node X a FAnt first updates the routing table. Details how this is done will soon be discussed. Afterwards it is tested, if the FAnt reached the end of its lifetime in this case the ant will be destroyed. If node X is not yet the target node the FAnt updates the last hop, the second last hop and the hop count in its body. If the target is reached, a BAnt is sent back towards the source.

In static and "nice cases" the algorithm works without any problem. If the network is dynamic we must face the fact that the neighbor Y could have moved out of the connection radius of the node. We will introduce the concept (route maintenance) we use in our algorithm to solve this problem in section 3.2.2. But first we have a look at a few other things. As next some detalis about how we make routing table entries.

3.1.3 Updating Routing Information

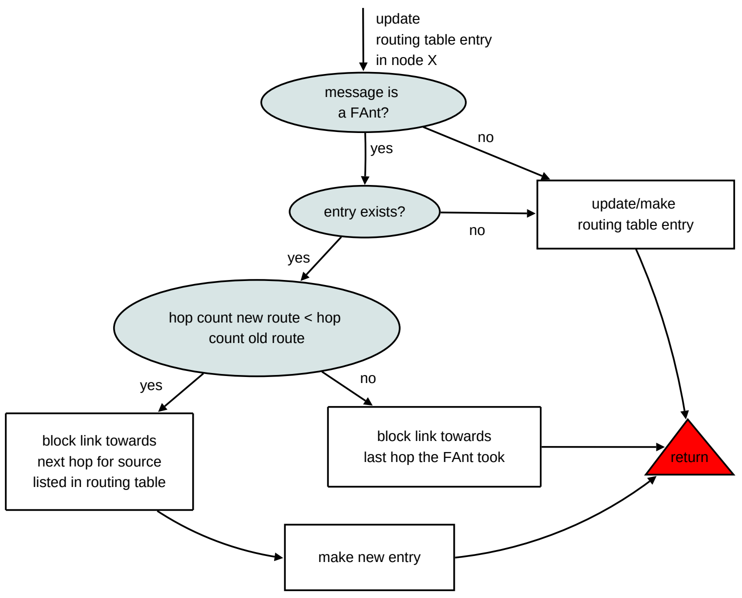
update
routing table entry
in node X
message is a FAnt?
yes
no
entry exists?
no
update/make routing table entry
yes
hop count new route < hop count old route
yes
no
block link towards next hop for source listed in routing table
block link towards last hop the FAnt took
return
make new entry

Figure 3.2: making routing table entries

In the design procedure we decided that each message arriving in a node X updates or makes the routing table entry for the source it came from. It does not put anything in the routing table about where it is going to. We based this decision on the following facts:

- First the message is coming from a source. Therefore it is carrying the newest information about the path to the source the node X is able to get. That is the main reason why we use this information.
- You may wonder why we are not updating the routing information for the target. We will see, that as you can imagine also our routes are not valid forever, if we have to deal with mobility. Therefore we locally have to take a path towards a target but this information may be out of date and not valid anymore.

So these two points illustrate the reason why a message makes only entries towards the source, if it enters a node X.

Preventing Loops

The problem we encountered is that with the the starting procedure we developed so far the algorithm produced loops. To illustrate the problem consider figure 3.3. The FAnt took the route

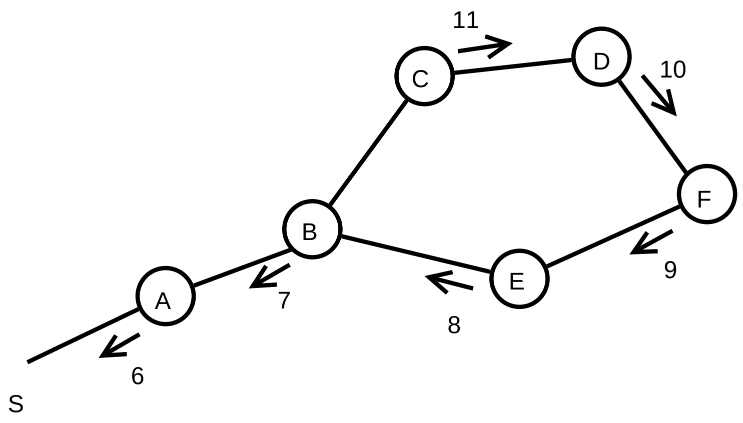

Figure 3.3: Loop Scenario

ABEFDC and writes in each node the routing table entries as drawn in the figure. Arriving again in B we have the problem that when we just overwrite the existing entry we produce a loop. If a returning BAnt reaches one of the nodes and follows the entries made, it ends up in a loop and will never reach the source. The idea behind the concept is the following: When a FAnt arrives

from a certain neighbor (in figure 3.3 C), the particular FAnt is not allowed to make a routing table entry. Now how do we detect if FAnts coming over a particular link are not allowed to make entries? And what is the profit of this procedure?
When a FAnt arrives in node B the node checks, if already a routing table entry exists. If the old one is better the node designates the link between C and B as blocked and it does not allow further FAnts arriving over this link to make routing table entries.
In the case of figure 3.3 node B would not allow FAnts arriving from C to make routing table entries anymore.
But when is a route better than another one? Comparing hop counts does not give you necessarily the correct answer. Nevertheless we decided to base our decision on the hop counts. But additionally we introduce a lifetime for the routing table entries and for the blocked links. The algorithm may over a small period block the wrong edge, but due to the lifetime after a while when the entries expire the procedure is starting again from the beginning. To conclude the section about making routing table entries consider the following routing table entry in node F (figure 3.3). We introduced all of this fields in this section.

target	next hop	after next hop	hops to target	lifetime
S	E	B	9	8

3.1.4 BAnts

As said, if a FAnt reached the target, we send a BAnt back to the source. The flowchart in figure 3.4 shows how a node handles a BAnt.
The procedure is very similar to the one for the FAnt. We first create a routing table entry. We then test, if the BAnt reached the end of its lifetime. This is just a security check that the BAnt is not walking around forever. This could for example happen, if a BAnt ends up in a loop. Loops are discussed in section 3.2.2
Reaching a node X, a BAnt first updates the routing table. A new routing table entry is made. Afterwards it is tested, if the BAnt reached the end of its lifetime. In this case the ant will be destroyed. If the target is reached, the connection is established. If node X is not yet the target node, the BAnt updates the last hop, the second last hop and the hop count in its header. Then the node looks up in his routing table for the next hop towards the source. The node is then trying to send the message to this node. If the node is due to mobility not available anymore the

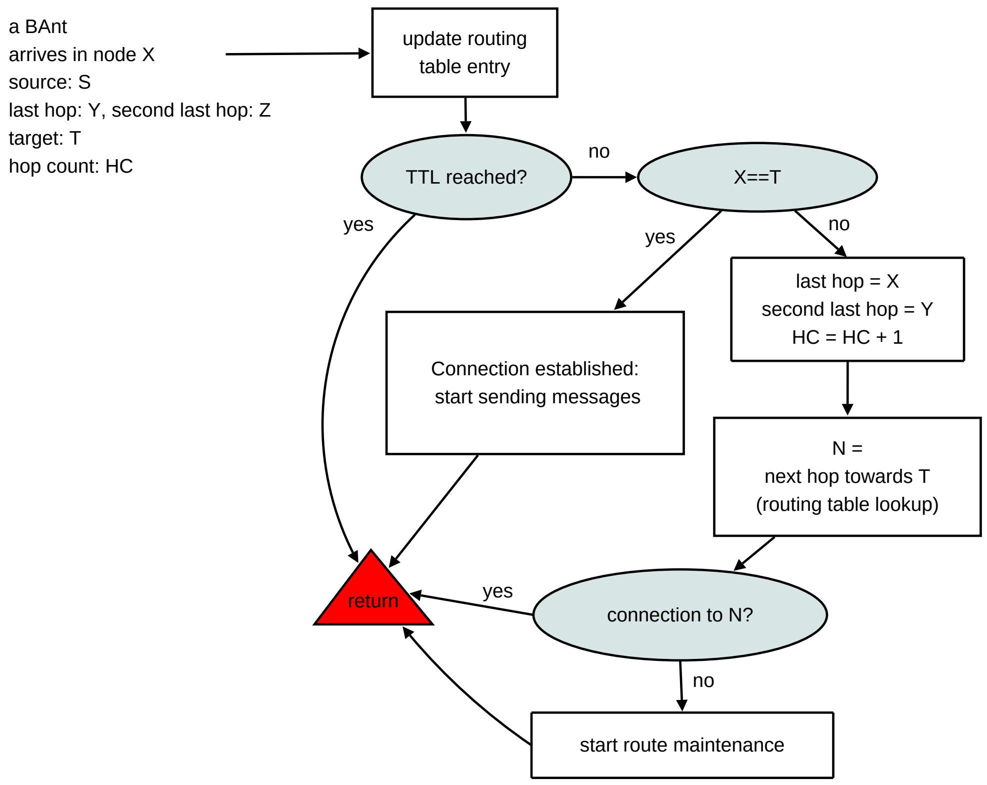

Figure 3.4: flow chart of the BAnt reaching a node

node starts the procedure route maintenance which will be discussed in section 3.2.2 on page 14.

3.1.5 Summary

At this time it is useful to shortly summarize the functionality introduce till now. The algorithm sends out FAnts in predefined exponentially increasing periods. These FAnts randomly walk around the network. Each time they reach a node, they write a routing table entry. A routing table entry contains a triple for a target: the target, the next hop and the after next hop. If a FAnts writes a routing table entry the entry is for the source of the FAnt, the next hop is the last hop the FAnt took and the after next hop is the second last hop the FAnt took. If a FAnt reaches the target a BAnt is sent back over the route the FAnts walked. The BAnt also writes at each node the routing table entry for its source (the target of the connection). When the BAnt returnes back to the source, the connection is fully defined in both directions. The source can start sending data packets. In each node the data packets updates the routing table. To be forwarded it has to read out the next hop in the routing table. If this node is not found anymore, because the node has moved away, a mechanism called route maintenance is started which will be discussed in section 3.2.2.

When we started simulating the algorithm, we encountered, that the route close to the target is (because of the random walk character of the FAnts) not really ingenious. Therefore we decided to build up a new generation of ants: the broadcast ants (BCAnts). We will start section 3.2 by describing the function of this type of ants.
Further on, there are still a few aspects to be discussed about the classes of ants we introduced until now. The following questions are essential: "How many FAnts have to be sent out to find the target", "How long should we wait until we send out the new ones?" or "How long should they walk around the network?" We will discuss such questions in chapter 4

3.2 Extensions of the Algorithm

In this section we present the extensions we made to improve the algorithm. We will start with the BCAnts and then talk about the concept for route maintance we used.

3.2.1 Improving the route

At the end of the previous section we introduced the problem that random walk does not really produce optimal routes. We encountered, that close to the target the route often circles the target node and does not really take a direct way. To work against this issue we introduced the BCAnts. They produce a small flooding around the target (when a FAnt reaches the target) to inform the nodes in the neighborhood that the target node is near. The flow chart in figure 3.5 gives the necessary details: We broadcast over 4 hops a BCAnt from the target of the connection (This will be the source from which the BCAnt comes) and make a routing table entry in each node the BCAnts visit. An other point the BCAnt could be good for, is that they could be used to stop the FAnts. What do we mean with that? As you certainly remember, we are sending out more than one FAnt and which arises the problem that FAnts are still walking around the network not knowing if the target is already found or not. An extension to the algorithm could, be that the FAnts use already existing routing table entries and with the help of the BCAnt the FAnts in the

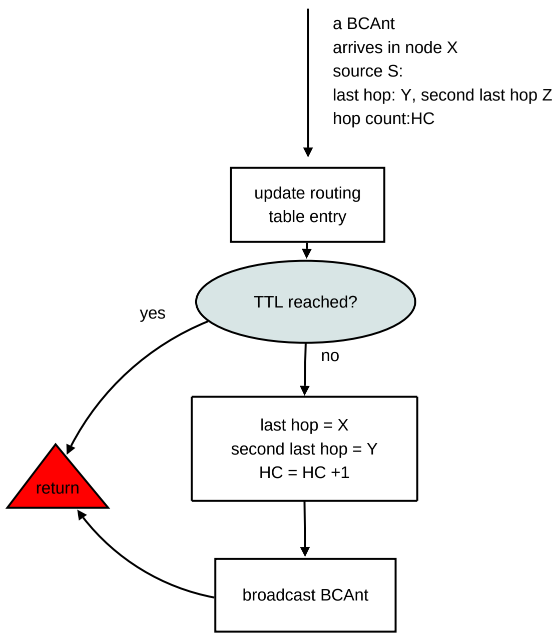

Figure 3.5: flow chart of the BCAnt reaching a node

neighbourhood of the target would end up there quicker. This would be an interesting point to focuss on we have not really done yet.

3.2.2 Dealing with mobility

The problem about mobility is, that the information received may not be correct anymore in the next round. We define mobility as the fact that all nodes taking part in the connection can move and change their position at any time. For the simulation in the next section we used a Random Direction mobility model with a given waiting time and a Gaussian distributed speed.

What are the possibilties of our algorithm, if the route described in the routing table entry is out of date? The FAnts routed around do not have any problem with out of date routing table because as described above they do not rely on them and just move randomly.

What about for a BAnt or a payload message? If the node to which the routing table points to is not available any more the message cannot take any further step. We decided to fix this problem locally and try locally to find a new route.

At first sight there may be two solutions solving this problem: One would be to search for a node which has a routing table entry (towards the target) whose hop count is smaller than the

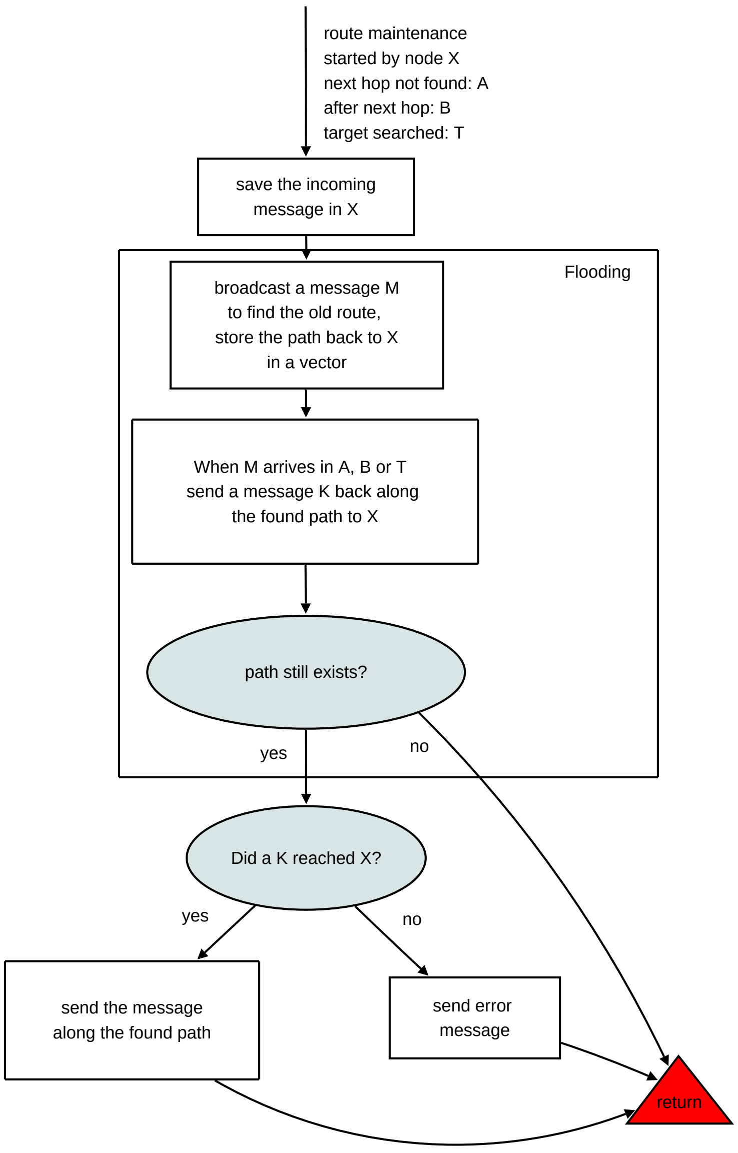
route maintenance
started by node X
next hop not found: A
after next hop: B
target searched: T
save the incoming message in X
Flooding
broadcast a message M to find the old route, store the path back to X in a vector
When M arrives in A, B or T send a message K back along the found path to X
path still exists?
yes
no
Did a K reached X?
yes
no
send the message along the found path
send error message
return

Figure 3.6: flowchart of the route maintenance

hop count of the current node. But the problem with such an approach is, that we actually do not know if a node is nearer or further form the target: With mobility, a characteristics like hop count, as already discussed, does not give reliable information, because the node, the entry is in, could have just moved across the whole plane.

We decided, that we cannot ensure if a found node is nearer or farther of the target with that approach. Another one is more promising:

We search for the next hop, the after next hop (towards the target) listed in the routing table and directly for the target. According to figure 3.6, we initialize a flooding in X. Sending out the message M, which is noting the taken route in its body, we found with high probability one of the searched nodes. If a path to one of these is found, we send a message K back to the node which initialized the route maintenance. We forward the message over the new routing path and make new entries in the routing table. This method is working quite well with a not too high speed of the nodes.

The only problem that may rise is at the point when K returns towards X (see figure 3.6). If the route is broken there, we drop this instance of K. But there is a good chance, that another way was found and there exists still a K and the route maintenance still can work. We will discuss the performance with a few statistics in section 4.2.

Loops

It is important to evaluate the problems we generate with this approach. A problem that always arises with dynamic networks is, that loops may be generated. To be able to discuss our counter steps against loops we first describe how loops are generated. Imagine the following scenario of the ideal case. Here ideal case means a route breaks up and we find a new way to the previous route and so a new route without any loop is established. (graphical details in figures 3.7 to 3.9).

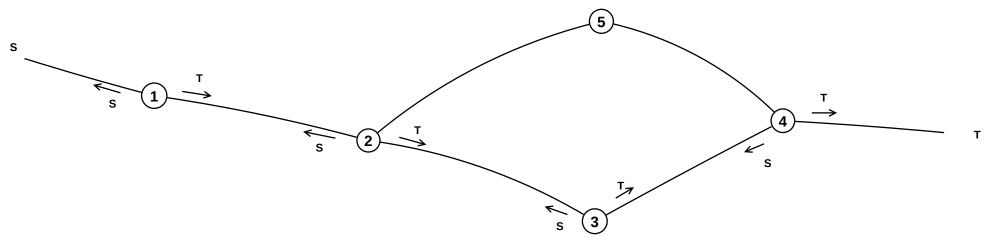

Figure 3.7: ideal case of finding a new route

A message is coming along the path from the source (S) to the target (T). Figure 3.7 describes the situation when everything is ok and the connection is already established. The black arrows are the routing entries already existing. Everything that is made by the route maintenance is red.

The broken link is marked with a cross (figure 3.8) and finally figure 3.9 the entries overwritten by the route maintenance are also marked with a cross.

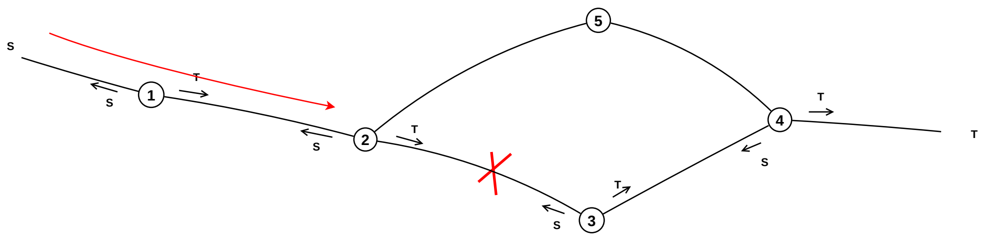

Figure 3.8: ideal case of finding a new route

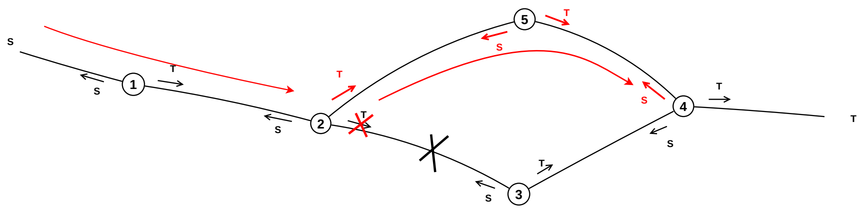

Figure 3.9: ideal case of finding a new route

Time proceeds when going form figure 3.7 to 3.9. First we have the starting point, then a link is broken and at the end a new route is found. The link between node 2 and 3 was broken and the algorithm found a new routing path over node 5 to node 4. The route is not longer, no loop is created so this is a quite good example.

Now we are ready to discuss how loops can be generated:
We detect two possibilities. A loop always arises if the new routing path crosses the old route at some point. This can either happen in the region of the route the message still has to take towards its target ("forward situation") or in the region of the route the message already passed ("backward situation"). The reason why loops are produced is, that the algorithm overwrites existing entries because it does not realize that this produces a loop. Therefore, the approach to avoid looping problems is, to teach the algorithm which entries he is not allowed to overwrite. We elaborate this procedure in detail: Let us first have a look at the first case:
The loops are generated when doing the route maintenance. When we - after finding the route - just move blindly along the routing table entries we end up in a loop (see figure 3.10). The concept to avoid such a problem is in the following: The message that is sent out after the route maintenance, remembers the nodes it met during route maintenance together with overwritten entries (if the entries existed). If on its way later on, the message visits a node that was visited

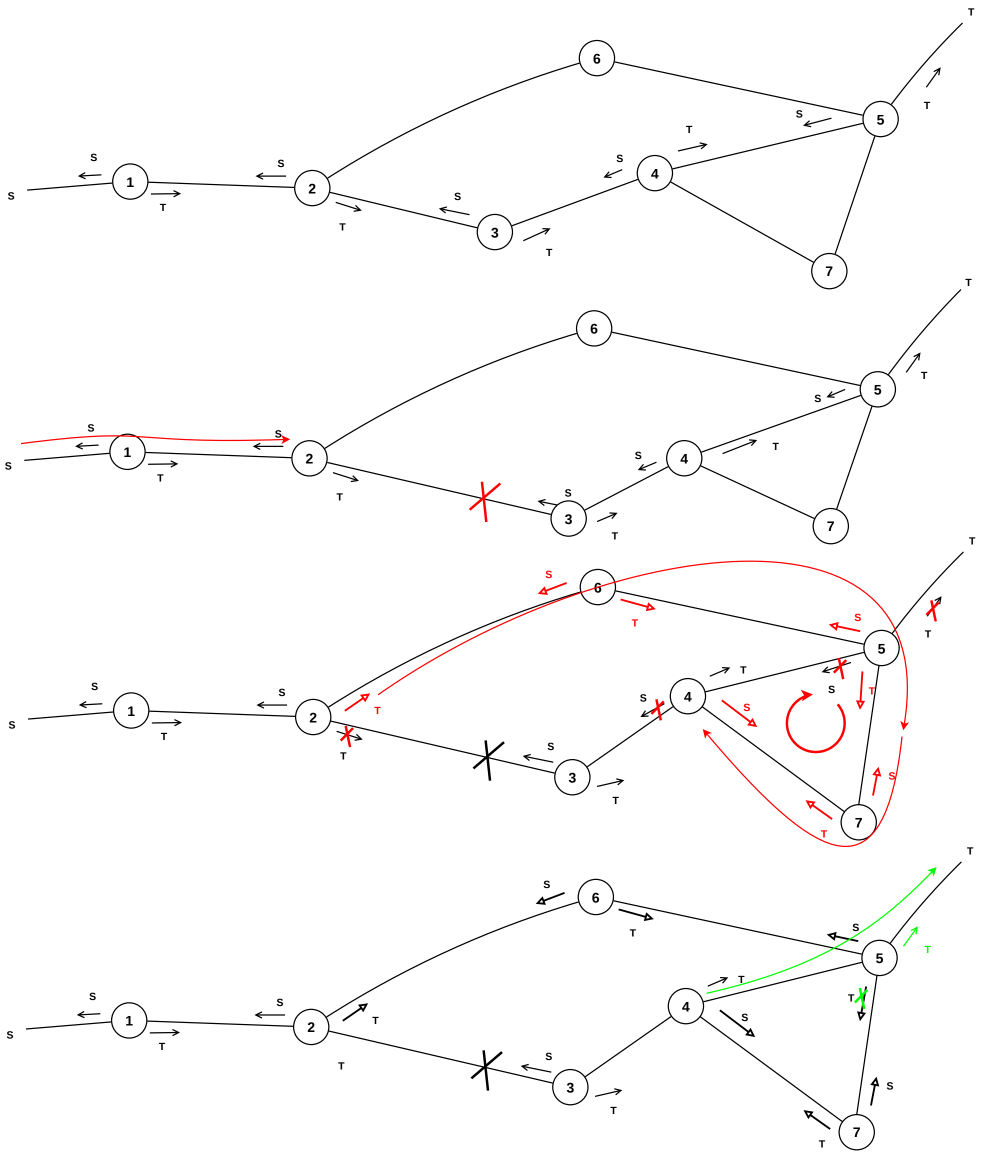

Figure 3.10: the found piece crosses the route in the "future"

during the route maintenance (this would be node 5 in figure 3.10) it puts back the old routing table entry for the target and then follows that entry towards the target. This way, all loops categorized as "forward situation" can be prevented. What about the "backward situation"?

If the algorithm would blindly applying the route maintenance described in the beginning of this section, it would generate a loop between the nodes 1, 2 and 5 in figure 3.11 because the routing table entry in node 1 had been overwritten. A message will end up in this loop, if it is routed towards S.

However, were are not able to fix this problem at the moment of the route maintenance. In this

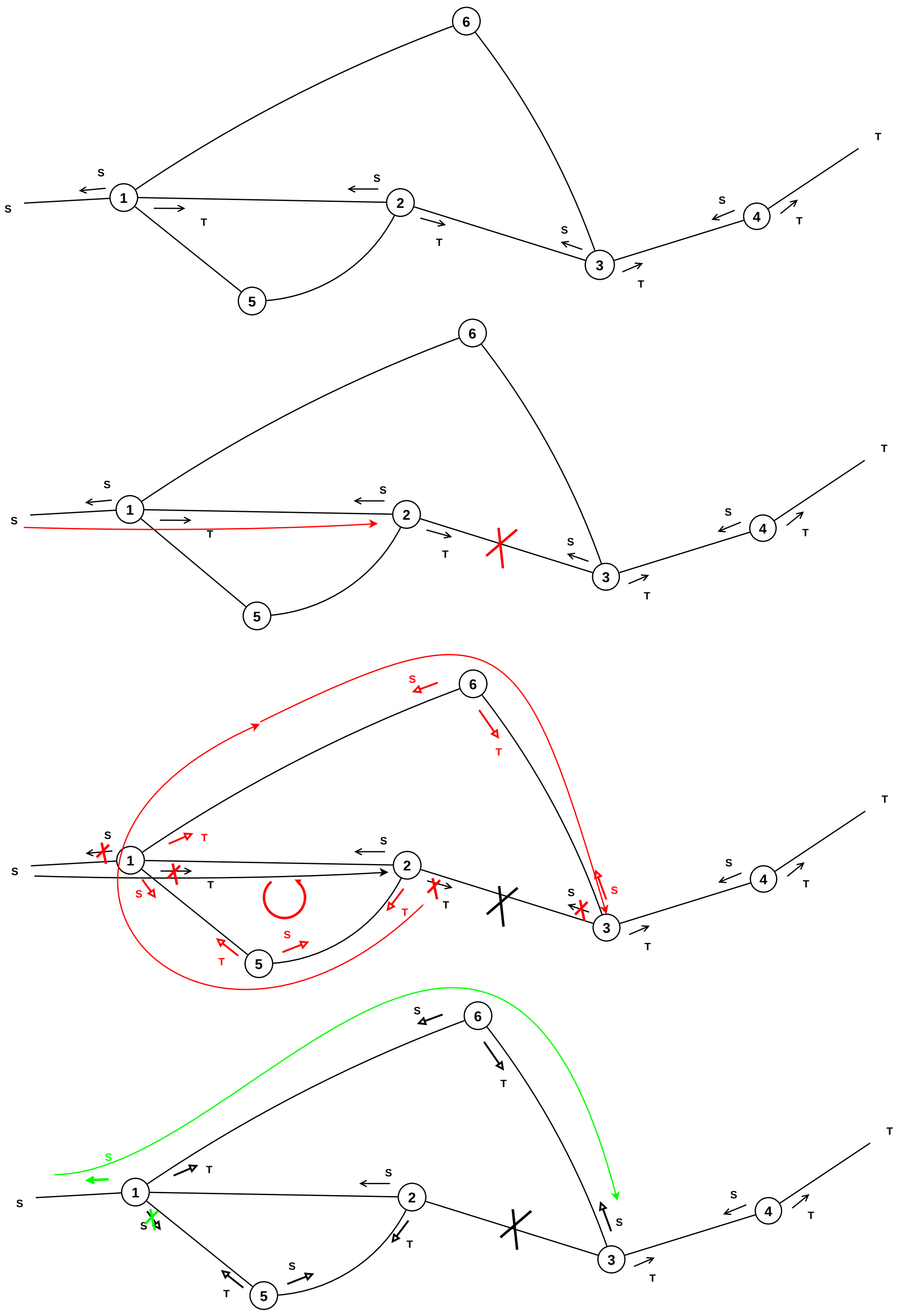

Figure 3.11: The found piece crosses the route in the "past"

case, the algorithm is able to fix this problem when it compares in each node having an entry for the target the next hop and after next hop (towards the target) with the origin of the route maintenance (node 2 in figure 3.11). In the picture node 1 would detect that the existing entry for the target T points to node 2, which is the origin of the route maintenance and then would

not make a new entry towards the source and just move on to node 6. (That is the version we implemented for our simulations)
But imagine that between node 1 and 2 lie two other nodes (named 7 and 8). Now the routing table entry towards the target would point to node 7 (next hop) and 8 (after next hop) and the algorithm would not recognize that by overwriting the existing entry it produces a loop.
This kind of generating loops can only be detected if a message is taking the way back to the source and controles, if it again passes one of the nodes met during the route maintenance procedure.

There are two possible implementations to address this problem: In the first, the algorithm sends another message back to the source, in the second, the algorithm waits until a message is sent back from the target to the source. In both cases we would replace the existing routing table entries with the old ones, if we visit a node already met in the new piece of the route found in the route maintenance. Both implementation would solve the problem but with both of them new questions or problems have to be considered.

- The first method provides a way of correcting loops. As a consequence this message could produce a lot of overhead. Because the algoritm does not know at what distance the loop occurs. (There could be even more nodes between node 1 and 2 than just 7 and 8. This is unlikely, but it is possible.) Therefore the message has to walk back to the source and on its way certainly has to take a route maintenance because the route will not be up to date anymore. When route maintenance has taken place the whole procedure has to be repeated and such a message has to be sent in the other direction and so on. This could produce a lot of traffic, specially if we have a long route to travel.

- The other approach is, that we remember the way taken during route maintenance until the target is reached and give this information to the message going back towards the source. Here we have the problem that some time may pass until the message reaches the region again and the route may note have anymore the same "node constellation".

For both approaches there is a weak point, which is more probable in the second approach. Imagine the following situation: A new route maintenance (from another connection towards the source or by the message itself) is started at node 6 and the algorithm directly finds the target. This would suggest that this specific route has no loop. But in fact the loop still exists in the network and if another connection D passes one of the nodes which was involved in the loop the messages from the connection D will loop forever.

3.2.3 Conclusion

As discussed in section 3.2.2 with mobility there is always a possibility that a loop cannot be prevented. Therefore, further investigations could and should be taken in this field. Questions like: "How can a message recognise that it is making a loop all the time?" and "Is there a possibility to break out of this loop and get back on the old route?" are key questions. Unfortunately we did not have time to investigate more closely in this area.
In the next section we will discuss the performance of this system. Question like "How many messages have to be sent out for a route maintenance?", "How many times do we have to start a route maintenance?" or "How effective is this system?" will be in the center of interest.

Chapter 4

Discussion

In this section we discuss how effective our algorithm is. We present our statistics, discuss them and make proposal how the algorithm could be improved. We start with a few words about the general aspects of the simulations. We decided to simulate with 500 nodes. The field the nodes are distributed has side length 560 units and two nodes are within commmunication range if their mutual distance is below 50 units. Initially, the nodes are randomly distributed in the field. These parameters holds for both cases the static and the dynamic network. When starting a simulation we randomly choose a source and a destination. And one simulation end in static case if the BAnt or the flooding echo returnes at the source. In dynamic networks each time a payload messages reaches the target an ACK messages is sent back. The source waits until it gets the ACK back and then sends the next payload message. We stop our simulation if 100 ACKs arrived at the source or if a route maintenance failed. According to what one simulation is, we simulate in static network 500 times and in dynamic 400 times.
In our simulations with mobility, the nodes move according to a random direction mobility model. In this model, each nodes independently determines its path and alternates between an waiting state and a moving state. When starting the waiting state, a node waits for random time, which is Poisson distributed. When starting to move, the node picks a random direction and moves with the node speed, which is Gaussian distributed, until the move time (Uniform distributed) elapsed.

4.1 Comparison with Flooding

First we compare our ant search algorithm with the basic flooding algorithm. Because the flooding algorithm has no mechanism if a connection between two nodes fails due to mobility, we decided to simulate the comparison with the flooding in a static network. First of all we present a short overview of the two algorithms we are going to compare. And after that we demonstrate the statistics out of the simulation.

4.1.1 The Flooding Algorithm

Let us first talk about the flooding algorithm that we compared with our algorithm. If a node wants to send a message to any other node, it starts a flooding request. For these flooding messages a time to live (TTL) is set. The source waits a time second the TTL until it sends out new flooding messages. The TTL of the new sent out flooding message is increasing exponentially starting with TTL = 4. It can be shown that an algorithm with an exponentially increasing TTL needs less messages to find the target than with a linearly increasing TTL in the following way: We assume that the nodes are placed in a line, so that each node has a right and a left neigbor. The source is at the end of the line and the target is k hops away from the source. Now we compare the exponential and the linear case relating to the number of messages needed to find the target. In the general case, if the nodes are not placed in a line, there are much more messages in the network but the differences between the total number of messages are quite the same. And because of that, the following proof holds also for the general case. With an exponentially increasing TTL the algorithm reaches the target at step i, if i is the smallest number s.t. $2^i \geq k$. The total number of messages sent until step i is

$$\sum_{j=0}^{i} 2^j = \sum_{j=0}^{log(k)} 2^j = 2^{log(k)+1} - 1 = O(k)$$

In comparison to that if the TTL increases linearly each step by L, the algorithm reaches the target at step i, if $i \cdot L \geq k$. Now the total number of messages is

$$\sum_{j=0}^{i} L \cdot j = \sum_{j=0}^{k/2} L \cdot j = L \left(\frac{\frac{k}{2}\left(\frac{k}{2}+1\right)}{2} \right) = O(k^2)$$

We can see, the algorithm with an exponentially increasing TTL finds the target with less costs. The time between two subsequent flooding messages are sent out, is two times the TTL. Therefore, it is guranteed, if the flooding reaches the target the echo message returns to the source before the next sequence of flooding messages is sent out. Otherwise it is possible that useless flooding messages are sent out.
The node which receives a flooding message stores them and forwards it if and only if it has not already received the message before. That is a mechanism which prevents that not to much copies of the message are sent all over the network.

4.1.2 The Ant Algorithm

Now the short overview of our ant algorithm. The source which want to send data packets sends out FAnts in exponentially increasing periods. The FAnts have a TTL and die until this time elapsed. The source waites two times the TTL until it sends out a new sequences of FAnts. The number of FAnts sent out each time increases also exponentially. The FAnts sent out walk randomly around the network and leave their track in writing routing table entries corresponding where they come from. If a FAnt reaches the target a BAnt is sent back the route the FAnt took. This is done by reading out the routing table entry. The BAnt writes at each node the routing table entries for the target. When it returnes at the source, the connection is established and data packets can be sent.
If, due to mobility, a node which a message should be sent to, is not any more in communication range, a mechanism called route maintencane is started. It initates a small flooding over 3 hops to refind this node.

Several Parameter Sets

We defined several parameters for the ant algorithm. One is μ the number of FAnts sent out at the beginning. Each time we sent out a new sequence of FAnts we first multiply the number of FAnt with a given factor α, which is also a parameter we can modify. The number of FAnts sent out is exponentially increasing. Another parameter is the TTL of the FAnts, we decided to start with the same TTL as in the flooding (TTL=4). Also for the time between two sequences of FAnts are sent out we choose the same as for the flooding, twice the TTL. The reason is, that the BAnt has time to return to the source before the next set of FAnts are sent out.

To get a feeling for the influence of μ and α on the algorithm performance, we run two simulations: One with $\mu = 30$, $\alpha = 1.1$, the other with $\mu = 10$, $\alpha = 1.05$. As shown in figure 4.1 the second paramter set needs significally less messages to find the target. One disadvantage

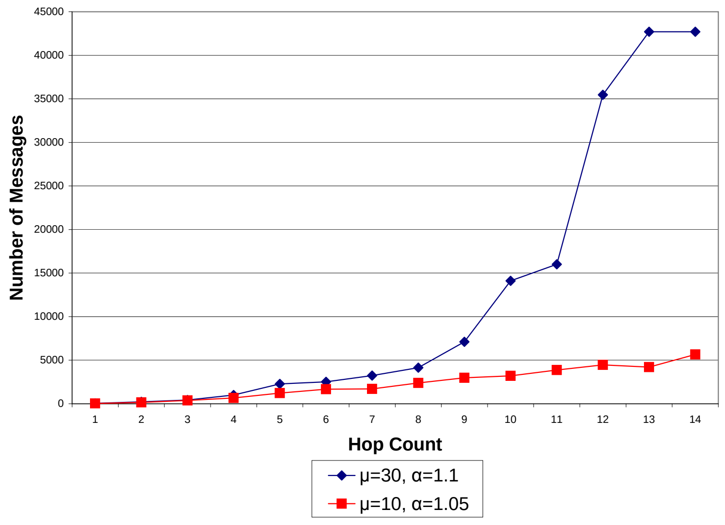

Figure 4.1: Difference of number of messages between the two parameter sets

of the second parameter set is, as one can see in figure 4.2 that it lasts longer until the target is found. We are quite sure that we haven't found the optimal parameters for our starting procedure yet. More investigations should be made to improve them. For the remainings simulations we decided to use the second parameter set with $\mu = 10$ and $\alpha = 1.05$. Because with the first parameter set the total number of messages enormously increases at higher hop counts.

4.1.3 Finding the Target

There are three different parameters characterising the procedure of finding the target. It is the total number of messages needed, the delay time from the point the source sends out the first FAnts until the target is found by a FAnt and the number of nodes receiving a FAnt or a flooding message respectively.

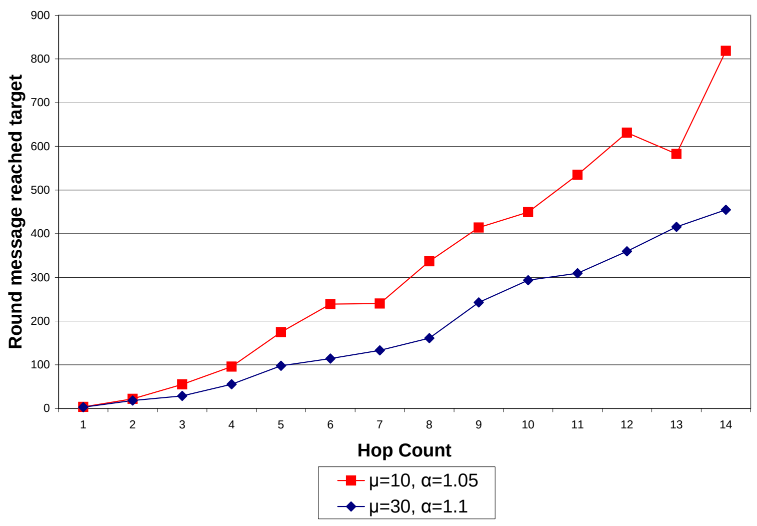

Figure 4.2: Difference of the round the messages reached the target

Number of Messages

Comparing the number of messages, one can see in figure 4.3 that we need less messages with the ant algorithm than the flooding needs. (Hop Count means the number of hops between the source and the target.)

Delay Time

If we compare the delay time one can see in figure 4.4 that the flooding needs much less time to find the target than the ant algorithm needs. But one of our goal was to lower the number of messages in the network.

Number of Nodes Receiving a Message

The last point to discuss in this section is the number of nodes, which receive a flooding message or a FAnt. Of course it is better, if as few as possible nodes receive a message. Because if a node receives a message it has to store a routing table entry and in most cases this information is never needed. We can see in figure 4.5 that the number of nodes which receive a FAnt is smaller than the one which receive a flooding message.

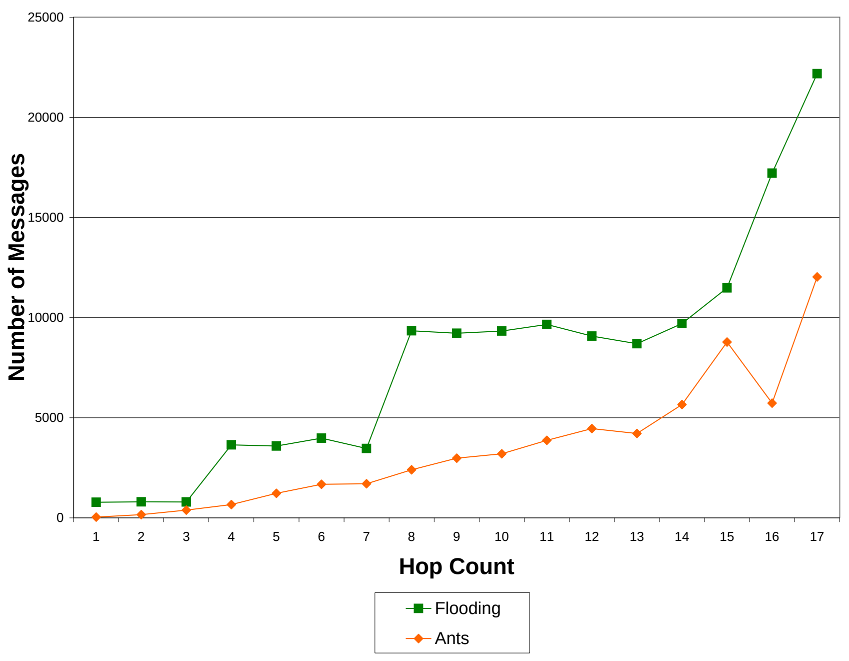

Figure 4.3: Difference of number of messages between ants and floodding

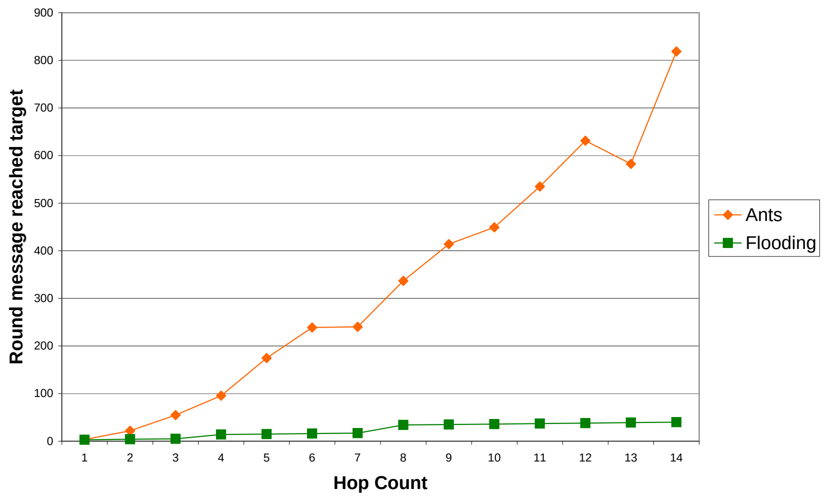

Figure 4.4: Difference of the delay time between ants and flooding

If the source and the target are more than eight hops away from each other, for the flooding algorithm all nodes receive a messages. If there are a lot of connections in one network and

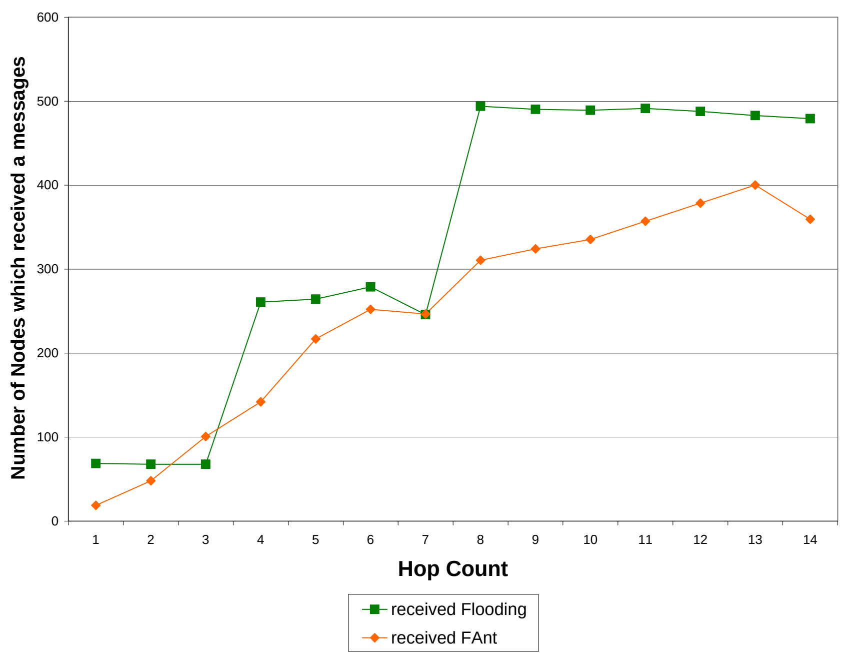

Figure 4.5: the number of nodes visited by route request

nearly all nodes have to store a routing table entry for each connection, as it is for the flooding, the amount a memory increases a lot. With the ant algorithm we see a linear increasing behavior of the number of nodes which received a FAnt.

4.1.4 Quality of the found route

Last but not least, we compare the length of the chosen route. I.e. the number hops between the sender and the target. Clearly, our Ant based algorithm produces longer routes than the flooding algorithm, as the flooding algorithm always finds the shortest path. Because the FAnts walk randomly around the network, they are very unlikely to find the optimal route. In figure 4.6 you can see the difference between the hop count in the ant algorithm and the optimal route found with the flooding algorithm.

4.2 Route Maintenance

As seen in section 3.2.2 we needed a procedure to deal with mobility. In this section we evaluate the performance of the kind of route maintenance we introduced. To find either the next hop we have not connection to anymore, the after next hop or the target we make a small local flooding

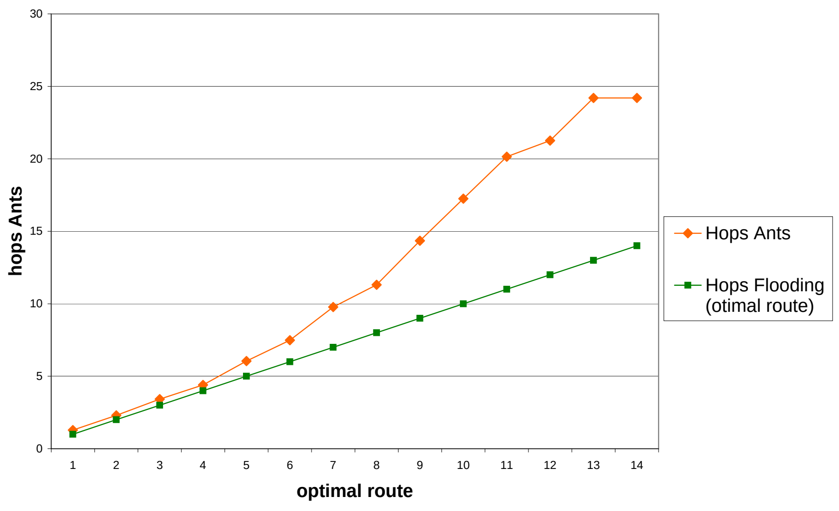

Figure 4.6: Difference to the optimal route

over 3 hops. If the algorithm finds one of these three searched node, the node started the route maintenance sends the message over the path just found. Reaching the found node, the message is send to the next hop for the target (listened in the routing table).

4.2.1 Simulations

We give a short overview which different simulation parameters we were using: In the beginning of this chapter we told about the mobility model. We choosed the mean speed value $\overline{v}$ in the range between 0.01 and 0.2 units and set the variance to $\sigma = 3 \cdot \overline{v}$. We had to choose such small mobility because of two reasons:

- Due to the long delay time we have to find the target our algorithm was not able to establish a connection for higher mobility.
- Secondly we got the samples by sending a message from source to target, replying to this message and afterwards again sent out a new one. Due to this setup, we had the problem that long routes (more than 5 or 6 hops) were not stable with higher mobility. Instead of just implementing a stop-and-wait protocol, a streaming environment would allow stable

routes with higher mobility.

We had to fix a stopping criteria for the simulation. We choosed the following: As message is sending from source to target and vice versa. If 100 messages are sent we stop the connection and start a new one.

Number of Failed Route Maintenance

First we have a look at the percentage of failure. We are interested how many times the route maintenance failed over the whole number of route maintenance done. The graphic 4.7 shows

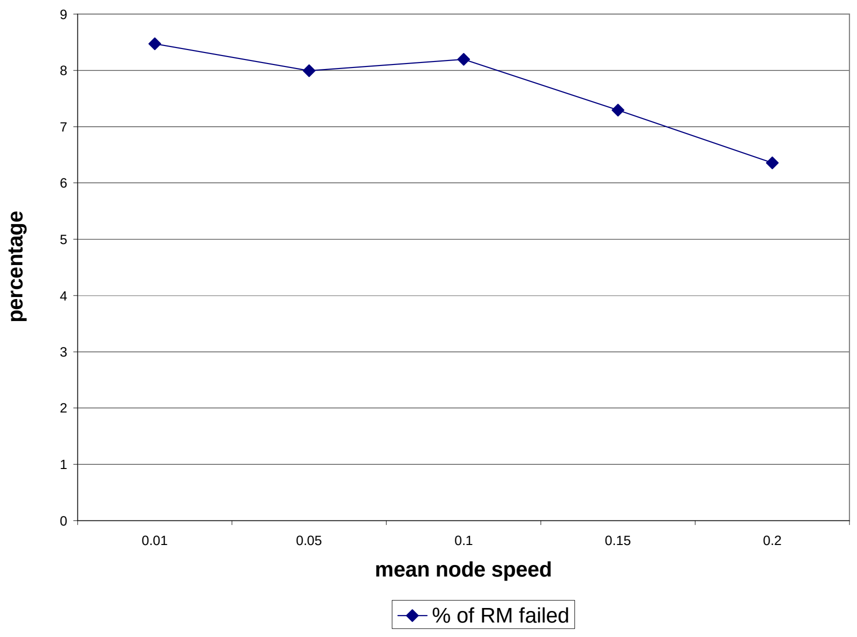

Figure 4.7: percentage of failed route maintenance

that the percentage of the failed route maintenance is, considered over the whole range of node speeds simulated, going down. A reason why this could be is, that because of the mobility the next node moved away, but the after next hop moved nearer. With higher moblitiy this is more probable. Of course with higher mobility when the next hop and the after next hop have moved a larger distance in the same environment as used here the number of route maintenance is increasing again. But in such a case one could think of changing the parameters. For example one could think of increasing the flooding distance. Of course this is not possible without any consequences:

- The number of messages for a route maintenance will increase.
- Secondly there will be a different behavior in the changing of hop counts (statistics for this environment you can find later in this section). Larger difference between two measurements will be a consequence.

Number of Route Maintenance

The graphic 4.7 in section 4.2.1 shows that the procedure of route maintenance has a slightly higher chance to succeed. Nevertheless, embedded in the whole algorithm, this looks a bit different. In the following table you find a overview over a few statistics.

$\bar{v}$	f_{BAnt}	$\bar{r}$	n_{OK}
0.01	19	60.633	41%
0.05	52	33.284	13%
0.1	59	22.622	7%
0.15	84	17.6711	5%
0.2	101	13.704	3%

$\bar{v}$ is the mean node speed, f_{BAnt} is the number of failed BAnt, $\bar{r}$ is the mean number of times a message reaches its target before the connections breaks up. At last we have n_{OK}. The best thing to understand what this number stands for is if you remember the setup for the simulation we have chosen: We said, that we stop our simulation if 100 messages were sent along the path. n_{OK} is therefore percentage of the number of samples in which 100 message where sent.
Just a few comments about the data. As you can see in the table, with higher mobility the number of BAnts that fail goes up, the number of messages that reach the target before the connections breaks up goes down. Imagine a message is sent from its source to its target. This message may take some route maintenance on its way. Figure 4.8 gives the statistics about the number of route maintenace a given message is performing until it reaches its target.

Changing of the Hopcounts

As last we like to discuss how the number of hops the message has to take changes over time. As we had not got that much samples and the mean number of the times a messages reaches its target was not too high we built up two diagrams. Figure 4.9 and 4.10 show how the length of the

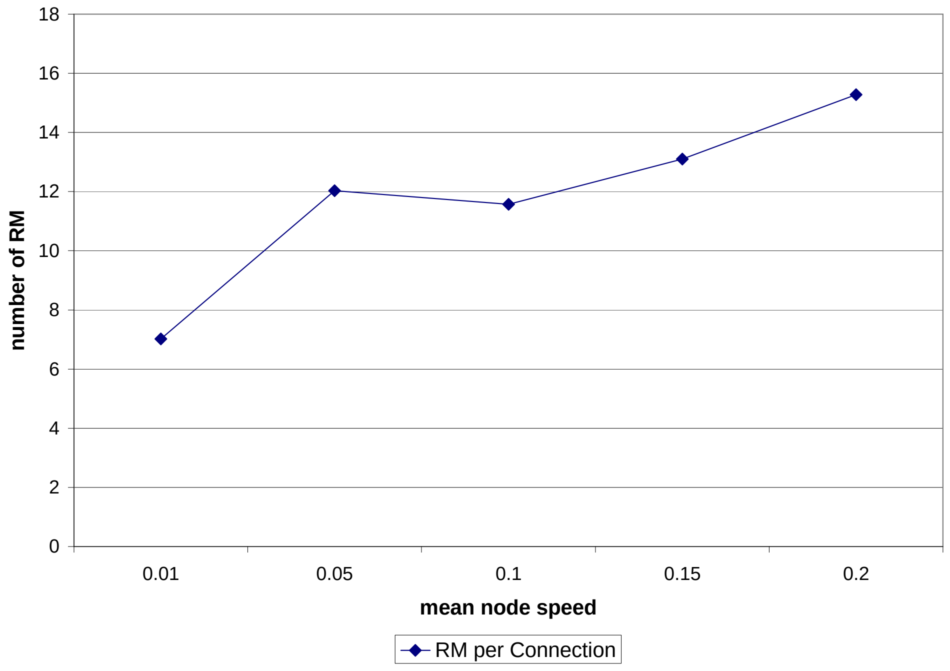

Figure 4.8: numbers of route maintenance needed depending on the underlying node speed

route changes over time. Each time a message reaches its target a sample of the number of node it traversed (hop count) is taken. In figure 4.9 this is done for each node speed until 20 messages are sent over the route. You may wonder why there is a gap between the different node speeds. Again this is because of the chosen setup. In the beginning of section 4.2.1 we discussed the fact that for higher mobility longer routes are less stable. Exactly this fact is illustrated by the gap. For the higher mobilities the longer route were not enough stable to deliver 20 sample to the simulation. The mean number of times a message reaches its target is decreasing with higher mobility (as seen a bit earlier in this section). This fact forced us for longer time periods only draw the statistics for small velocities. The number of hops is increasing over time but as you can see in figure 4.10 it seems that increasing rate becomes smaller over time. This has its background in the chosen procedure. The locality of the route maintenace procedure is responsible for this fact. Because we only search locally for the next hop or the after next hop, when the route length increases the probability of finding the after next hop increases as well. If a after next hop is found in most of the cases the route length is not increased or even decreased (if the after next hop has become neighbor to the node starting the route maintenance).

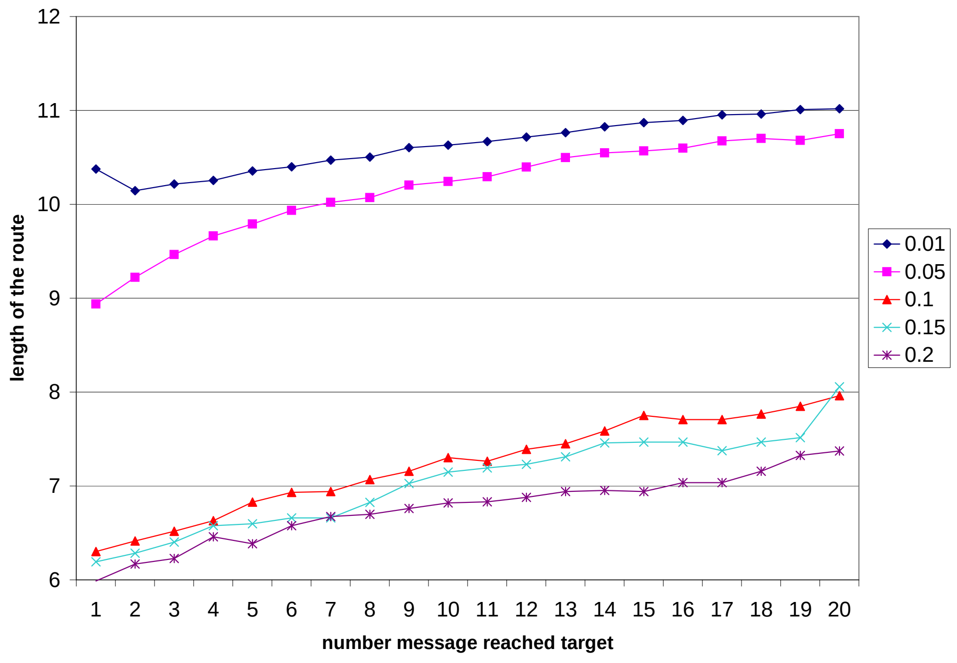

Figure 4.9: number of hops of the route

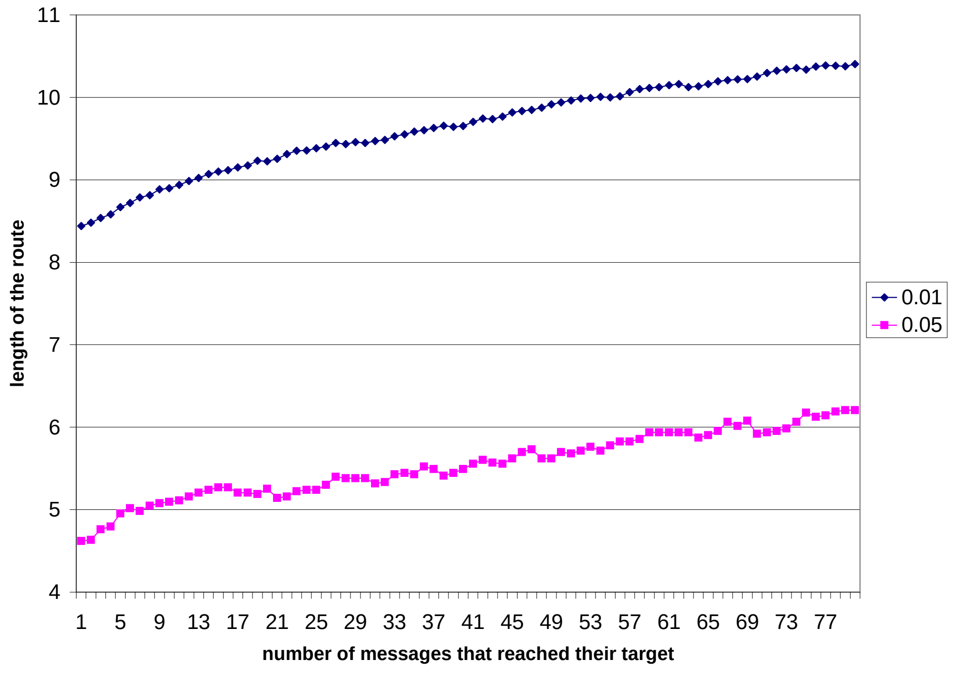

Figure 4.10: number of hops of the route

Chapter 5

Summary and Conclusions

In this section we summerize the Ant Algorithm and some conclusions out of the simulation.

5.1 The ant algorithm

The algorithm sends out FAnts in predefined exponentially increasing periods. These FAnts randomly walk around the network. Each time they reach a node, they write a routing table entry. A routing table entry contains a triple for a target: the target, the next hop and the after next hop. If a FAnts writes a routing table entry the entry is for the source of the FAnt, the next hop is the last hop the FAnt took and the after next hop is the second last hop the FAnt took. If a FAnt reaches the target a BAnt is sent back over the route the FAnts walked. The BAnt also writes at each node the routing table entry for its source (the target of the connection). When the BAnt returnes back to the source, the connection is fully defined in both directions. The source can start sending data packets. In each node the data packets updates the routing table. To be forwarded it has to read out the next hop in the routing table. If this node is not found anymore, because the node has moved away, a mechanism called route maintenance is started. The route maintenance starts a local flooding over 3 hops to find the next hop, the after next hop or the target.

5.2 Simulation results

We made some static simulation to compare the ant algorithm with the basic flooding algorithm. We showed in the simulation that for 500 nodes the total number of messages sent until the target is found is smaller with the ant algorithm. On the other hand one disadvantage is, that the delay time from the source starting searching the target until the target is found is higher than with flooding.
To indentify the qualitiy of the route maintenance we simulated in a dynamic network. We found that the number of failed route maintenace in a sparse graph is decreasing with higher mobility. In a streaming invironment also long route should become stable with the developed and last but not least the route maintenance procedure only slightly increases the hop counts over time.

Chapter 6

Future Works

In this last section we summarize some interesting aspects which should be discussed as well. Due to time constraints we are not able to do that all.

6.1 Ant parameters

One point are the ant parameters, especially μ the number of ants sent out at the beginning and the factor α, μ is multiplied each time a new sequence of FAnts is sent out. As mentioned in the discussion section 4.1.2 we simulate with two different parameter sets. The difference between these two were, related to the total number of messages until the connection is established, quite big. That is why we think there must be better paramters as we found them. More investigations should be made to find an optimum.
Indeed it is also possible to change the start TTL. The FAnts in our simulation had a start TTL of 4 and we increased the TTL exponentially, multiply it each time with 2. It is possible to change this factor or make it variable. If the FAnts in general live longer, the factor is bigger, it is likely that delay time gets smaller, but the total number of messages could be bigger. The same way it hold in the other side, if the factor is smaller, with bigger delay time and smaller total number of messages. The same problem as above, the difficulty supposed to be in finding the optimum.

6.2 Route Maintenance

As discussed in section 3.2.2 with mobility there is always a possibility that a loop cannot be prevented. Therefore further investigations could and should be taken in this field. So questions like: "How can a message recognise that it is making a loop all the time?" and "Is there a possibility to break out of this loop and get back on the old route?" are key questions.
Further on the Route Maintenance procedure should be implemented in a streaming environment and there be tested for its behavior. This scenario should significantly improve the behavior of the developed procedure.

Bibliography

[1] David A. Levin, Yuval Peres, Elezabeth L. Wilmer : Marcov chains and Mixing Times. 2006

[2] Peter G. Doyle, J. Laurie Snell: Random walks and electric networks. 2000

[3] G. Di Carlo and M. Dorigo: AntNet: Distributed stigmergetic control for communications networks. 1998

[4] *http://sarwiki.informatik.hu-berlin.de/Routing_Protocols_overview* 23.03.2007

[5] G. Di Caro, F. Ducatelle, L.M. Gambardella: AntHocNet: An Adaptive Nature-Inspired Algorithm for Routing in Mobile Ad Hoc Networks.

[6] D. Subramanian, P. Druschel, J. Chen: Ants and Reinforcement Learning: A Case Study in Routing in Dynamic Networks.

[7] M. Günes, U. Sorges, I. Bouazizi: ARA - The Ant-Colony-Based Routing Algorithm for MANETs. 2002