Faculty of Information Technologies and Systems

Ant Net for Mobile Ad-Hoc Networks

Popa Mirela

SCIENTIFIC COORDINATORS
Drs.Dr.L.J.M ROTHKRANTZ, Delft University of Technology

Table of contents

1. Introduction

This paper tackles the problem of dynamic routing for packets in a network. We built a simulation environment with a routing system that guides the packets through the network using the fastest way, taking into account the load on the links.

Recently, a new kind of routing protocols for fixed, wired communication networks is developed inspired by the behaviour of the ant colonies. One of these ant-based routing protocols is *AntNet*. AntNet is an *adaptive distributed* routing protocol for packet-switched communication network (the Internet). In *distributed* routing systems there is no control from a central point, but the information is shared among the network nodes. AntNet is also *adaptive* (*dynamic*), which means that the routing tables are created by automated construction and by updating. In the dynamic systems, the routing policy adapts to the changes in the traffic conditions and changes in the network topology. The changes in the network topology can be caused, for example, by break down of the links or the nodes in the network.

AntNet uses “artificial ants” that would repeatedly travel through the network and collect the information about the current traffic conditions in the network. This information would be used to direct the data packets towards their destination. AntNet showed very promising results and turned out to be highly adaptive in dynamic network environments . The capability of AntNet to adapt to dynamic environments seems to make AntNet-like protocol well suited for the routing in MANET.

In this thesis, we address the problem of implementing AntNet to a network topology where the nodes can move. AntNet is made for fixed network and could not work with the mobile nodes. In order to apply AntNet in this environment, we adjusted the activities of the artificial ants in such a way that they can function in a network with mobile nodes. The dynamic network topology required also changes in the model of nodes. This resulted in a dynamic node model and the introduction of a new buffer. Finally, AntNet and the collection of our adjustment resulted into *Mobile AntNet*.

Mobile AntNet is tested on a software tool *AntNet for Ad-hoc network*, which is the Java version of *Ant Simulator* developed by Bogdan Tatomir. The purpose of this conversion to Java is to make the software work on PDAs.

2. Ants Behavior and Communicating Networks

2.1 The Principle of Stigmergy

Small animals like ants have only local knowledge. When a group of such animals interact with each other, they can exhibit a higher level of behavior. This behavior is most often called emergent behavior. A group of insects like ants doesn't have a central controlling authority that steers the behavior. All ants behave according to a few simple rules and interact with their environment. The resulting behavior of the group of ants can be very complex. A few examples that have been observed in several species of ants are:

- regulating nest temperature within limits of 1ºC;
- forming bridges;
- raiding particular areas for food;
- building and protecting their nest;
- sorting brood and food items;
- cooperating in carrying large items;
- emigration of a colony;
- complex patterns of egg and brood care;
- finding the shortest routes from the nest to a food source;
- preferentially exploiting the richest available food source.

These behaviors emerge from the interactions between large numbers of individual ants and their environment. In many cases, the principle of stigmergy is used. Stigmergy is a form of indirect communication through the environment. Ants only react to local stimuli from the environment. Ants can change one or more of those local stimuli. This will influence the future actions of ants at that location. The environmental change may take either of two distinct forms. In the first, the physical characteristics may be changed as a result of carrying out some task-related action, such as digging a hole, or adding a ball of mud to a growing structure. The subsequent perception of the changed

environment may cause the next ant to enlarge the hole, or deposit its ball of mud on top of the previous ball. In this type of stigmergy, the cumulative effects of these local taskrelated changes can guide the growth of a complex structure. This type of influence has been called sematectonic. In the second form, the environment is changed by depositing something that makes no direct contribution to the task, but it is used to influence subsequent behavior. This sign-based stigmergy has been highly developed by ants and other exclusively social insects, which use a variety of highly specific volatile hormones, or pheromones, to provide a sophisticated signaling system. A type of sign-based stigmergy is used to investigate a new approach for congestion avoidance in telecommunications networks. It is based on the way ants find short routes from their nest to a food source, and also on the way they select between food sources of different value.

Depending on the species, ants may lay pheromone trails when traveling from the nest to food, or from food to the nest, or when traveling in either direction (Fig 1) .

Fig. 1

They also follow these trails with a fidelity, which is a function of the trail strength, among other variables. Ants drop pheromones as they walk by stopping briefly and touching their gaster, which carries the pheromone secreting gland, on the ground. The strength of the trail they lay is a function of the rate at which they make deposits, and the amount per deposit. Since pheromones evaporate and diffuse away, the strength of the trail when it is encountered by another ant is a function of the original strength, and the time since the trail was laid. Most trails consist of several superimposed trails from many different ants, which may have been laid at different times; it is the composite trail strength which is sensed by the ants. The principles applied by ants in their search for food are best explained in the following example.

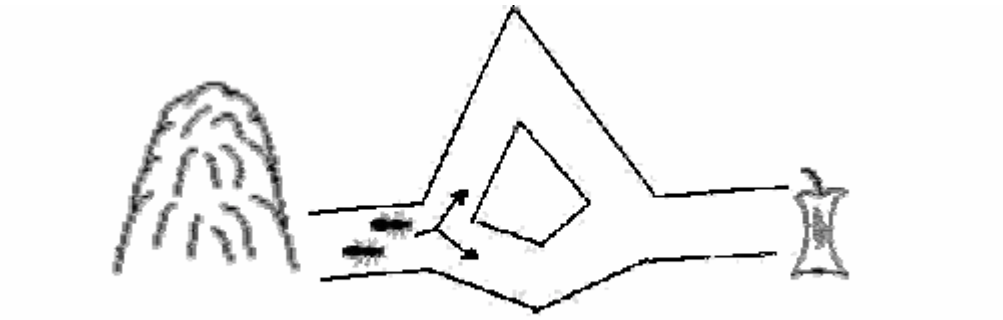


Fig. 2

Figure 2 illustrates two possible routes between nest and food-source. Initially, an ant arriving at a T-crossing (choice point), makes a random decision with a probability of 0.5 of turning left or right. Now let suppose there are two ants leaving the nest to look for food, and let the ants be of a type such as *Lasius Niger* which deposits pheromones when traveling both to and from the nest. If one ant from each pair turns left, and the other turns right, after a while a situation occurs like that in Figure 3. The lines on the paths represent the pheromone trails. The ants that chose the shorter branch have arrived at their destination, while the ones that chose the longer branch are still on their way. Ants initially select their way with a 0.5 probability for both branches, as there is no pheromone on the paths yet. If there is pheromone present, there is a higher probability of

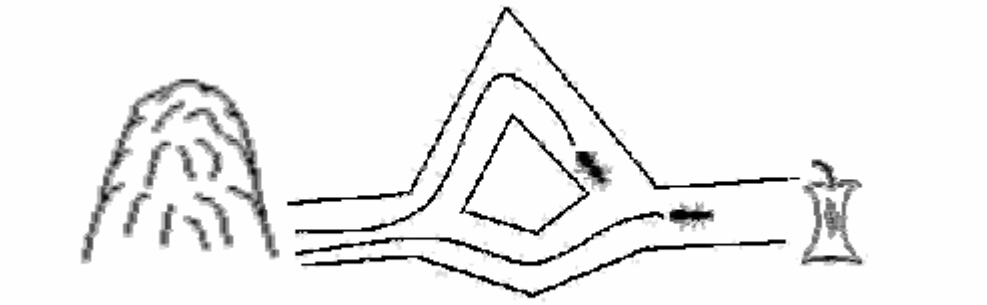


Fig. 3

an ant choosing the path with the higher pheromone concentration, i.e. the path where more ants have traveled recently. If at the moment of the situation in Figure 4 other ants arrive and have to choose between the two paths, they are more likely to choose the shorter path, because that is where the concentration of pheromone is higher. This means that the amount of pheromone on the shorter path is more likely to be reinforced again.

In this way, a stronger pheromone trail will arise on the shorter path, and so the path will be

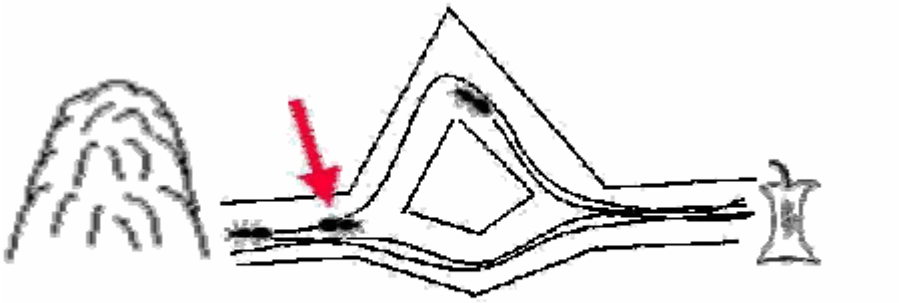


Fig. 4

selected by an increasing proportion of ants. So the ants will bias towards the shortest path because of the following three reasons:

- Shorter routes will be completed earlier than longer routes and thus attract other ants to their source nodes first.
- Shorter routes involve fewer branches, so the number of ants will be larger on longer routes with more branches.
- Ants traveling shorter routes will be younger when they arrive. This cause the pheromone trails to be stronger, because less of the pheromone trail has diffused away.

This strengthening process continues until pheromones on the longer paths will disappear altogether.

2.2 The artificial ants in AntNet

The ant-based routing algorithms are using the same principle as in the example above but they are using *artificial ants* or *agents. S*ince we want agents to solve more complicated problems, we will make them more complex than the real ants. In contrast to real ants, the agents will have same additional possibilities:

- They will have a memory. Memory will allow them, for example, to remember their path and to go back using the same path;
- They will be able to deposit their pheromone proportional to the quality of the produced solution. In our case, for example, we will look at the quality of the ant's path. The quality depends on duration of this trip.
- They are not completely 'blind'. They will be able to 'see' more than just a pheromone and their decisions will be influenced by some local information (in our case by current traffic conditions). This will help ants to solve the *shortcut problem.*

In order to illustrate a possible behaviour of the artificial ants, we will shortly describe a roll of the agents in (Mobile) AntNet.

In (Mobile) AntNet, we will use agents to solve the routing problem in a telecommunication network. We will consider a case with connected network, and that is if all nodes in the network could be reached from any other node in the network.

An ant has a task to find the minimum cost path joining its source *s* and destination *d*. It moves from node *s* towards node *d* hopping from one node to the next until node *d* is reached. During this trip, the ant memorises its path. At each intermediate node, the ant is choosing the next node to move to according to a probabilistic rule. This probabilistic rule depends on a local pheromone trail and current traffic conditions in that part of the network. As it travels towards its destination, the ant is collecting information about traffic load in the network. When *d* is reached, the ant is going back using the same path. On its way back, the ant is laying pheromone and it will distribute the collected information.

The agents have two tasks: *collecting* and *distributing the information* about the traffic load in the network. To make a clear distinction between these two tasks, we will consider two types of ants:

- *Forward ants* $F_{s \to d}$ that are moving from their source *s* to their destination *d*. Their tasks are to explore the network and collect the information about traffic load;
- *Backward ants* $B_{d \to s}$ that are moving along the same path as $F_{s \to d}$ but in the opposite direction, thus from *d* to *s*. Their task is to distribute the information that $F_{s \to d}$ collected.

2.3 Definition of Network Routing System

Routing in distributed networks can be characterized as follows.

A network is *modelled as a graph G*= (*N, E*), consisting of *N* nodes connected by *E* edges. Each node is functioning as a communication end-point (a host) and as a forwarding unit. As a host, each node generates *data and routing packets* with a randomly chosen destination. After that, a packet is sent towards its destination. If a node is not the destination of a packet, the packet is queued in *the buffer space in the node* and will be forwarded towards its target node. To be able to forward a packet towards its destination node, the node is using information from *the routing table*.

The main task of a routing algorithm is to direct data flow from source to destination nodes maximizing network performance. In the problems we are interested in the data flow is not statistically assigned and it follows a stochastic profile that is very hard to model.

In the specific case of communications networks, the routing algorithm has to manage a set of basic functionalities and it tightly interacts with the congestion and admission control algorithms, with the links queuing policy and with the user-generated traffic.

The core of the routing functions is:

- the acquisition, organization and distribution of information about user-generated traffic and network states,
- the use of this information to generate feasible routes maximizing the performance objectives,
- the forwarding of user traffic along the selected routes.

A common feature of all the routing algorithms is the presence in every network node of data structure, called *routing table*, holding all the information used by the algorithm to make the local forwarding decisions. The routing table is both a local database and a local model of the global network status. The type of information it

contains and the way this information is used and updated strongly depends on the algorithm's characteristics.

A board classification of routing algorithms is the following:

- centralized *versus* distributed;
- static *versus* adaptive.

In *centralized* algorithms, a main controller is responsible for updating all the node's routing tables and/or to make every routing decision. Centralized algorithms can be used only in particular cases and for small networks. In general, the delays necessary to gather information about the network status and to broadcast the decisions/updates make them infeasible in practice. Moreover, centralized systems are not fault-tolerant. In this work, we will consider exclusively distributed routing.

In *distributed* routing systems, the computation of routes is shared among the network nodes, which exchange the necessary information. The distributed paradigm is currently used in the majority of network systems.

In *static* (or *obvious*) routing systems, the path taken by a packet is determined only on the basis of its source and destination, without regard to the current network state. This path is usually chosen as the shortest one according to some cost criterion, and it can be changed only to account for faulty links or nodes.

Adaptive routers are, in principle, more attractive, because they can adapt the routing policy to time and spatially varying traffic conditions. As a drawback, they can cause oscillations in selected paths. This fact can cause circular paths, as well as large fluctuations in measured performance. In addition, adaptive routing can lead more easily to inconsistent situations, associated with node or link failures or local topological changes. These stability and inconsistency problems are evident for connection-less than for connection-oriented networks.

2.4 Ad-hoc network model

The nodes and the links in our model could be explained as follows.

- ***The nodes*** in the graph are representing the computing devices. Each node has a node identifier (unique number) and it is functioning as both a *host* and a *router*. As a host, the node can be a communication end-point. This means that it can be a source and a possible destination of a data message. Each node holds a buffer space where the incoming and the outgoing packets are stored. This space is limited by a node capacity [bits].

  The nodes are also the routers. They are able to forward data messages, and to send and receive routing packets.

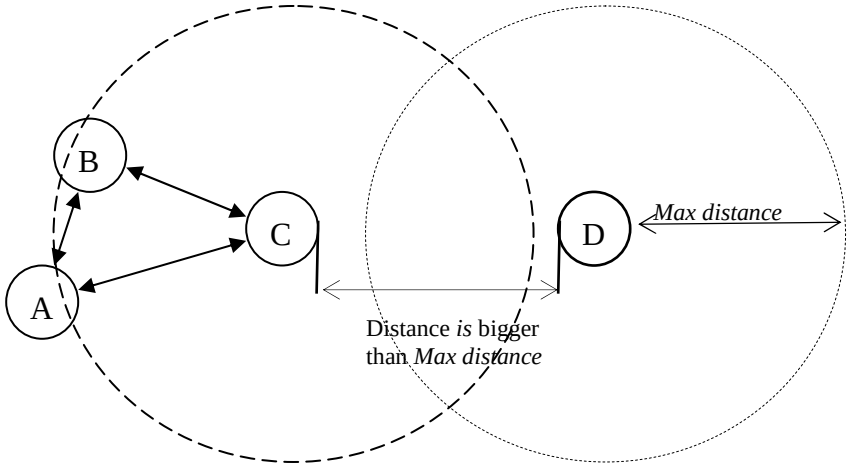


Figure 1. Schematic representation of making connection between nodes The circles around *C* and *D* are giving the transceiver range of these nodes The nodes *A* and *B* are in the range of the node *C* and they are connected to *C* while *D* is out of range and not connected to C

- In MANET, the nodes communicate via wireless ***links***. Each node has a limited transceiver range. We assume that two nodes are connected with each other if the distance between them is smaller than a given *maximum distance.* All the links in

the network are bi-directional. They are characterised with a *bandwidth* [bit/sec] and a *transmission delay* [sec]. Although, wireless communication deals with the effects of radio communication, such as noise, fading, and interference, they will not be taken in consideration. We assume that the links are reliable.

In MANET, network topology is dynamic, because the connectivity among the nodes is changing as they are moving. While moving, the nodes can stay connected to other nodes but they can also be completely without neighbours. In our approach, the nodes are spread over previously defined area. They can move everywhere within this area but they are unable to go out of this area. The network is also defined with a known number of nodes. This means that no new nodes can be introduced in the network. A node in the network will always know who its neighbours are. So, while moving, a node will immediately know its new neighbours, and in every moment during the simulation it knows how many neighbours it has.

In *Ant Simulator* the nodes are moving by making one step per second. To be able to simulate different speeds, we used different step sizes. The step size is proportional to *maximum distance* between the connected nodes. If a node has a higher speed, it will make a bigger step then a node with the lower speed. We call these speeds *moving modes.* We investigated four different moving modes of a node. The moving modes and the speed that a node would have in the real world (between the brackets is given a step size) are:

- *Fixed mode* 0 km/h - the network is connected and the nodes are not moving.
- *Walk mode* 5 km/h (step size is 0.0175 of the maximum distance) ;
- *Bike mode* 15 km/h (step size is 0.0525 of the maximum distance);
- *Car mode* 72 km/h (step size is 0.25 of the maximum distance).

Each node, every second is randomly making its step in one of the four possible directions (up, down, left or right) or staying at the same position. All nodes are making their steps in the same time, and after that they will automatically update their list of neighbours.

2.5 Buffers at a node

Every node in MANET functions, both as a host and a router and it has also the possibility to move. These properties have influence on the structure of the nodes' model. In this paragraph we will discuss buffers at the nodes' model that we use in Mobile AntNet.

Each connected node has three kinds of buffers:

1. *Input buffer*;
2. *Output buffer* with high- and low-priority queues for every neighbour;
3. *Ad-hoc buffer* with one queue for each destination in the network.

The originally AntNet uses nodes with two buffers: the input and the output buffer. In the Mobile AntNet we implemented an extra buffer - *Ad-hoc buffer*.

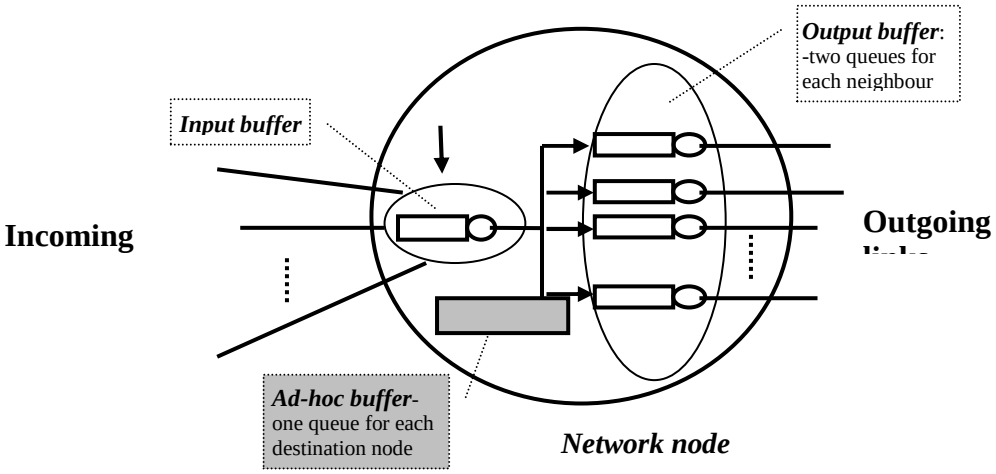


Figure 2. A representation of the node model when the node has neighbours. Compared to AntNet, Mobile AntNet has one extra buffer, called *Ad-hoc buffer*.

The topology of the Ad-hoc network may change rapidly due to node's ability to move. The node's model strongly depends on the number of node's neighbours. This means that as it is moving, the node's structure will change as a result of getting or losing the neighbour(-s). Obviously, the dynamic properties of the network demand also a dynamic structure of the nodes.

To describe the functioning of the buffers in a node, we will analyse the following situations that may occur:

1. connected network with no topology changes;
2. topology of the network is changing
3. a node becomes isolated.

2.5.1 Connected network with no topology changes

In this situation the network is connected. The nodes are not moving and their position is known (we have this situation in the connected fixed network). Every node in the network has one or more neighbours, and with each neighbour, a node has an incoming and an outgoing link. When a node receives a packet from a neighbour, or if a packet is generated in the node itself, then the packet goes first to the *input buffer*. All packets in the input buffer are processed with FIFO policy and the maximum number of the packets that one node can store is limited by *the buffer capacity* of a node. If the buffer is full, that is when the sum of all *packet sizes* [bits] in the node is equal to the *buffer capacity* [bits] of that node, than the next incoming and/or generated packet will be lost.

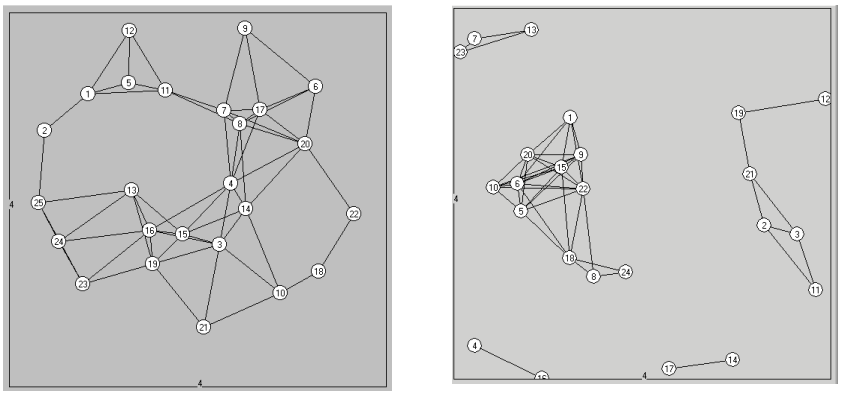
If a node is the destination of a data packet, then the packet is treated as a transfer-completed packet (the node will not send acknowledgment that the packet is successfully received). Otherwise, if a node is not the destination of the packet, the node will decide, by looking at the routing table and concerning the destination of the packet, which neighbour should be the packet's next node. After this decision is taken, the packet goes to the *output buffer* and it is put to the queue that corresponds with the packet's next node. The output buffer consists of two kinds of queues for every neighbour. These queues are:

low-priority buffer and *high-priority buffer queue*. We have two kinds of queues because there are two classes of packets in the network. Data packets will be queued in the low-priority buffer queue and routing packets in the high-priority buffer queue. A packet is waiting in a queue until the link resources are available, then these resources are reserved and the packet is transmitted to the next node. While travelling to the next node, each packet is experiencing a *link delay* [sec]. Obviously, as it is travelling to its destination, the packet is experiencing a delay due to the waiting time in the queues and the transmitting time. If its lifetime is expired during its waiting in a queue, the packet is destroyed.

2.5.2 Topology of the network is changing

In the *second situation*, the nodes are able to move. As the nodes move, the topology of the network is changing enormously, and most of the time we will have the situation where the network is not connected. In figure 4 part **a**. we see a (connected) network at the beginning of the simulation. As a result of nodes' mobility the network is becoming disconnected. The originally connected network became a set of the single nodes and smaller networks that are disconnected from each other.

As said before, the nodes communicate directly with each other if the distance between the nodes is smaller than the given *maximum distance*. As they are moving, the distances between the nodes will vary, and in this way the neighbourhood of a node is changing. A node will lose its neighbour, for example, if the distance between the node and its former neighbour becomes bigger then the maximum distance. Thus, while moving, a node can get a new neighbour, lose a neighbour and current neighbours remain the same. The changes in the neighbourhood of the node will influence its structure.



a. **b**.

Figure 3. Example of impact that the node's mobility has on a network topology. In **a**, a network is given at the beginning of the simulation, and in **b**. the 'same network' during the simulation.

If a node gets a new neighbour or loses a neighbour, its structure is changing. The reason for this is that each node has in its outgoing buffer, the (low- and high-priority) buffer queues for every neighbour. Therefore, when a node gets a new neighbour, the node will create new queues that are corresponding with the link to the new neighbour. Otherwise, if a node loses its neighbour, then the node will destroy the queues that are leading to that neighbour. The data packets that were in the destroyed buffer queue are put back in the input buffer, and they will be re-routed. The routing packets (ants) from the high-buffer queue are immediately destroyed. They are destroyed, because if they have to wait again in the input buffer, the information that they carry with them would be too old to be taken into the consideration. Therefore some ants will not get back to their source. If a node doesn't get any response for a while, and that is if there are no ants that are coming back from a certain node, than is that node seen as *unreachable.* If such a situation occurs, than a data packet that has this unreachable node as its destination is put from the input buffer to the *Ad-hoc buffer*. In this buffer the packet will stay until its destination node become reachable again or until its lifetime expires.

Before we put a new packet in the Ad-hoc buffer, its contents will be checked to see if there are packets whose lifetime has expired. These packets will be destroyed. To be able to control this buffer regularly, each node has a queue for every possible destination in the Ad-hoc buffer. As soon as some destination is reachable again, the Ad-

hoc buffer that responds with this node is emptied and the packets are re-routed to that destination. These packets are not going back to the input buffer, but they are put directly to the queue in the output buffer that corresponds to their next hop. As we have at least two nodes that were disconnected from each other, both of them will send their packets towards each other. The final result of this process is that at the certain moment (when the nodes become reachable again for each other) a large amount of packets will be sent and received in short period of time. We will call this the ***Ad-hoc buffer effect***. This effect may have impact on the data flow in the network.

2.5.3 A node becomes isolated

If a node has no neighbours, then the node will have only the input and the Ad-hoc buffer. As soon as this situation occurs, all the data packets that were in the node's output buffer are put in the corresponding queues in the Ad-hoc buffer. At the same time, all routing packets in that node are destroyed. The node keeps generating both classes of packets. As long as the node is *disconnected* from the rest of the network (has no neighbours), the routing packets are destroyed immediately, and the data packets are put in the Ad-hoc buffer.

The situation when a node has no neighbours is slightly different from the situation when a node loses a neighbour. Without neighbours, the node is unable to send any packets, because there are no links, and the node has to put all its data packets directly to the Ad-hoc buffer. Otherwise, when a node loses a neighbour (assuming that there are more neighbours), it can still send packets.

We see that, in the situations that are typical for the Ad-hoc network (second and third situation), the Ad-hoc buffer will be used, while in the first situation it will not be used (this explains the name of the buffer). If a network is connected, thus every possible destination is reachable, and the nodes are not moving, than the Ad-hoc buffer is empty and not used, but it still exists. The Ad-hoc buffer queues are created as a fixed structure of a node. They will not be destroyed if the nodes are not moving, or created as the node begins to move. The number of Ad-hoc buffers depends on the number of nodes in the

network and this number is constant. On the other hand, the number of neighbours of a node is not constant. The outgoing buffers in a node will be destroyed or created, depending on the number of neighbours that the node has.

The Ad-hoc buffer has a very important roll in the optimal working of the network:

- It helps to improve the control of the network's resources and to decrease the number of lost packets. Without this buffer, the packets that have unreachable destinations would be either destroyed at once, or they would stay circulating in the network. If they are destroyed immediately, this means that we are expecting that their destination will be unreachable during their lifetime. On the other hand, if they stay circulating in the network, they would be queued and transmitted until their destination becomes reachable or until their lifetime is expired. In both cases it would be unnecessary use of network resources. These packets would cause higher delay of other packets due to the queuing of these packets.
- This buffer is also the only storage space in a node. This is obvious if we think of a node without neighbours. The node is not able to store the packets somewhere else in the node and not in the way that is done in the Ad-hoc buffer- by looking at their destinations. Without this storage space all packets in a node without neighbours would be lost.

2.6 Data structure of a node

The nodes in the network are functioning also as routers. To accomplish this task, they are using ants to gather information about the *traffic load* in the network. The traffic load depends on the amount of packets in the network (congestion level of the network). The congestion level is proportional to the delay that a data packet experiences during the trip from its source to its destination. If the congestion level is high the data packets will need more time to get to their destination. To collect the information about the traffic load on their path towards destination, the ants are calculating the time that a data packet would experience using the same path. This information is used to update the two data structures in a node. These two data structures are: *the routing table* and *the local traffic statistics*. The ants are able to read and write in these two structures, while the data packets are only reading information from the routing table to get to their destination.

- ***Routing table*** is a local data-base that helps router to decide where to forward data packets. It contains the information which specifies the next (neighbour) node that should be taken by a data packet to get to any possible destination in the network. Each routing table is organized as a set of
    - all the possible destinations (all the nodes in the network),
    - the probabilities to reach these destinations through each of the neighbours of the node (next hops).

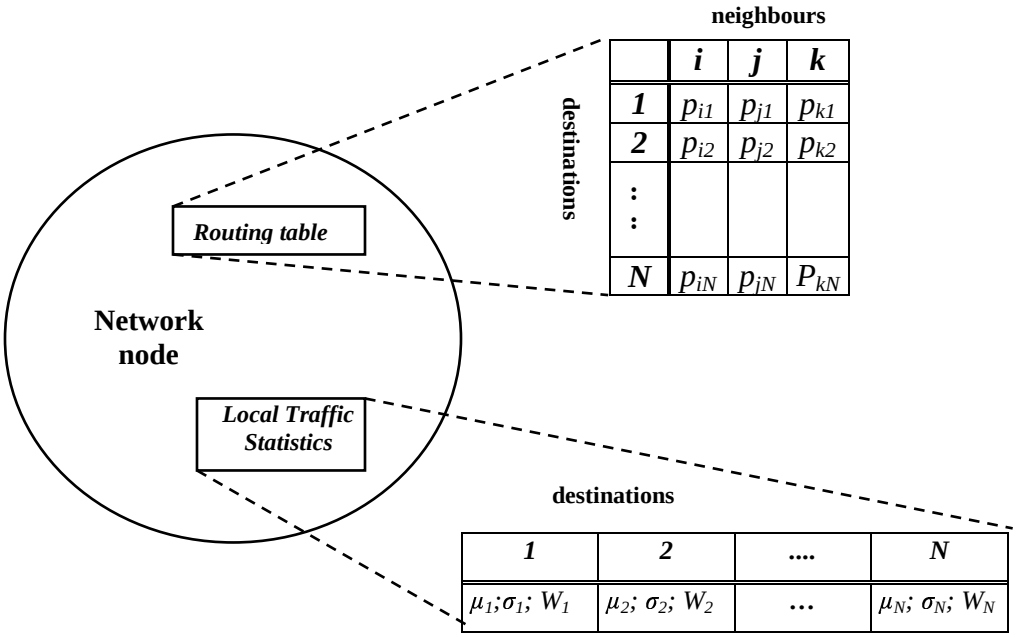


Figure 4. The data structures in a node*: routing table* and *local traffic statistics*. The presented node has *i*, *j* and *k* as its neighbours, and the destinations are all nodes in the network. The network has *N* nodes.

A routing table in a node *i* is given as $P_i = \{p_{jd}\}$ with p_{jd} a probability value that expresses the goodness of choosing node *j* as its next node from the current node *i* if the packet has to go to the destination node *d*. For any connected node *i* it holds that

$\sum_j P_{jd} = 1$; where *j* is a neighbour of node *i* and *d* is an arbitrary destination from all nodes in the network

This means that the sum of all probabilities to get to a randomly chosen destination *d* from the node *i* is equal to 1. If we take the routing table in figure 4 for example, then the sum $p_{i1} + p_{j1} + p_{k1} = 1$. This corresponds with a situation that we have in the fixed networks, but if the nodes are moving as in MANET, then some destinations will not be reachable. For these destinations, the ants will be unable to update the routing tables. Therefore there will be no probabilities corresponding to these destinations. In the case when a node has no neighbours, the node is unable to

send any ants. Obviously, all destinations are unreachable, and the routing table will be empty.

From the ant colony point of view, the probabilities in the routing tables can be seen as amount of pheromone. The probabilities are the product of continual exploration process of the ants. They are updated by the ants that *previously* used a path that is leading to the same destination.

- ***Local traffic statistics*** is a second data-structure that each node has. The main task of this structure is to follow the traffic fluctuations in the network.. It is given by an array M_i (μ_d, σ_d^2, W_d) that represents a sample means μ_d and variance σ_d^2 computed over the packet's delay from node *i* to all the nodes *d* in the network, and value W_d where the packet's best trip time towards destination *d* is stored. The array M_i contains statistics about the traffic from node *i* towards each possible destination *d*. The mean μ_d and variance σ_d^2 are giving an expected trip time and its stability:

$$\mu_d \leftarrow \mu_d + \eta(o_{k\to d} - \mu_d)$$
$$\sigma_d^2 \leftarrow \sigma_d^2 + \eta((o_{k\to d} - \mu_d)^2 - \sigma_d^2)$$

where $o_{k->d}$ is the new observed agents. trip time from node *k* to destination d.

W_d is obtained by applying a moving observation window of size *x* to store the agents' trip time to destination *d*. This window is used to compute the best agents' trip time towards destination *d* W_{best} as observed in the last *x* samples. W_{best} represents a short term memory and it should follow the fluctuations of the traffic in the network. Obviously, this can not be *the best* trip time, but only a moving (temporary) lower bound of the time needed to travel from the current node to some destination node *d.* . The factor η weights the number of most recent samples that will really affect the average. The weight of the t_i sample used to estimate the value of μ_d after j sampling, with j>i, is: $\eta(1-\eta)^{j-i}$. In this way, for example, if η=0.1, approximately only the latest 50 observation will really influence the estimate, for η=0.05, the latest 100, and so on. Therefore the number of effective observations is ≈5(1/ η).

The values from the local traffic statistics are used for estimation of the quality of the ant's path. These values are used to update the routing tables .

3. The Algorithms

3.1. The AntNet Algorithm

AntNet is a novel approach to the adaptive learning of routing tables in communication networks. AntNet is a distributed, mobile agent based Monte Carlo system that was inspired by recent work on the ant colony metaphor for solving optimization problems. It was developed by Marco Dorigo and Gianni Di Caro from IRIDIA, Universite Libre de Bruxelles.

The AntNet algorithm is described as follows.

1. At regular intervals Δt from every node *s*, a mobile agent (forward ant) $F_{s->d}$ is launched toward a destination node *d* to discover a feasible, low-cost path to that node and to investigate the load status of the network. The forward ants don't share the same queues as data packets, but virtual delays will be assigned to them to simulate the experience of the same traffic loads with the data packets.

Destinations are locally selected according to the data traffic patterns generated by the local workload: if *fsd* is a measure (in bits or in number of packets- I used the number of packets) of the data flow $s \rightarrow d$, then the probability of creating at node *s* a forward ant with node *d* as destination is

$$p_d = \frac{f_{sd}}{\sum_{d'=1}^{N} f_{sd'}}$$

In this way, ants adapt their exploration activity to the varying data traffic distribution.

2. While traveling toward their destination nodes, the agents keep memory of their paths and of the traffic conditions found. The identifier of every visited node *k* and the time elapsed since the launching time to arrive at this *k*-th node are pushed onto a memory stack $S_{s->d}(k)$.

3. At each node *k*, each traveling agent headed towards its destination *d* selects the node *n* to move to choosing among the neighbors it did not already visit, or over all

the neighbors in case all of them had been previously visited. The neighbor *n* is selected with probability (goodness) P'_{dn} computed as normalized sum of the probabilistic entry P_{dn} of the routing table with a heuristic correction factor I_n taking into account the state (the length) of the *n*-th link queue of the current node *k*:

$$P'_{dn} = \frac{P_{dn} + \alpha l_n}{1 + \alpha(|N_k| - 1)}$$

The heuristic correction l_n is a [0, 1] normalized value proportional to the length q_n (in bits waiting to be sent) of the queue of the link connecting the node *k* with its neighbor *n*:

$$l_n = 1 - \frac{q_n}{\sum_{n'=1}^{N_k} q_{n'}}$$

The value of α weights the importance of the heuristic correction with respect to the probability values stored in the routing table. l_n reflects the instantaneous state of the node's queues, and assuming that the queue's consuming process is almost stationary or slowly varying, l_n gives a quantitative measure associated with the queue waiting time. The routing tables values, on the other hand, are the outcome of a continual learning process and capture both the current and the past status of the whole network as seen by the local node. Correcting these values with the values of *l* allows the system to be more reactive., at the same time avoiding following all network fluctuations. Agent's decisions are taken on the basis of combination of a long-term learning process and an instantaneous heuristic prediction. Depending on the characteristics of the problem, the best value to assign to the weight α is around 0.4.

Fs →d also computes the virtual delay with the following equation:

$$T(k \rightarrow n) = \frac{q_n + Size(F_{s \rightarrow d})}{Bandwidth_{k \rightarrow n}} + D_{k \rightarrow n}$$

where $Bandwidth_{k->n}$ [bit/s] is the bandwidth of the link $k \rightarrow n$, $D_{k->n}$ [s] is the propagation delay of link $k \rightarrow n$, $Size(F_{s->d})$ [bit] is the size of $F_{s->d}$. Then $F_{s->d}$ pushes $T(k \rightarrow n)$ into his stack $S_{s->d}$ and is put into the high-priority link queue of the link *k*

$\rightarrow n$. Consequently, he is able to move on the link with less delay as if he really moved. The heavy traffic situations will be detected much faster in this way.

4. If a cycle is detected, that is, if an ant is forced to return to an already visited node, the cycle's nodes are popped from the ant's stack and all the memory about them is destroyed. If the cycle lasted longer than the lifetime of the ant before entering the cycle, (that is, if the cycle is greater than half the ant's age) the ant is destroyed. In fact, in this case the agent wasted a lot of time probably because of a wrong sequence of decisions and not because of congestion states. Therefore, the agent is carrying an old and misleading memory of the network state and it is counterproductive to use it to update the routing tables.

5. When the destination node *d* is reached, the agent $F_{s->d}$ generates another agent (backward ant) $B_{d->s}$, transfers to it all of its memory, and dies.

6. The backward ant takes the same path as that of its corresponding forward and, but in the opposite direction. At each node *k* along the path it pops its stack $S_{s->d}(k)$ to know the next hop node. Backward ants also don't share the same link queues as data packets; they use higher priority queues, because their task is to quickly propagate to the routing tables the information accumulated by the forward ants.

7. Arriving at a node *k* coming from a neighbor node *f*, the backward ant updates the two main data structures of the node, the local model of the traffic M_k and the routing table T_k, for all the entries corresponding to the (forward ant) destination node *d*. With some precautions, updates are performed also on the entries corresponding to every node $k' \in S_{k->d}$, $k' \neq d$ on the "sub-paths" followed by ant $F_{s->d}$ after visiting the current node *k*. In fact, if the elapsed trip time of a sub-path is statistically "good" (i.e., it is less than $\mu+I(\mu,\sigma)$, where *I* is an estimate of a confidence interval for μ), then the time value is used to update the corresponding statistics and the routing table. On the contrary, trip times of subpaths not deemed good, in the same statistical sense as defined above, are not used because they don't give a correct idea of the time to go toward the subdestination node. In fact, all the forward ant routing decisions were made only as a function of the destination node. In this perspective, sub-paths are side effects, and they are intrinsically sub-optimal because of le local variations in the traffic load (we can't reason with the same perspective as in dynamic programming,

because of the non-stationarity of the problem representation). Obviously, in case of a good sub-path we can use it: the ant discovered, at zero cost, an additional good route.

In the following two items the way *M* and *T* are updated is described with respect to a generic "destination" node $d' \in S_{k->d}$.

M_k is updated with the values stored in the stack memory $S_{s->d}(k)$. The time elapsed to arrive (for the forward ant) to the destination node *d.* starting from the current node is used to update the mean and variance estimates, $\mu_{d'}$ and $\sigma_{d'}^2$; and the best value over the observation window $W_{d'}$. In this way, a parametric model of the traveling time to destination *d'* is maintained. The mean value of this time and its dispersion can vary strongly, depending on the traffic conditions: a poor time (path) under low traffic load can be a very good one under heavy traffic load. The statistical models had to be able to capture this variability and to follow in a robust way the fluctuations of the traffic. This model plays a critical role in the routing table updating process.

The routing table T_k changed by incrementing the probability $P_{d'f}$ (i.e., the probability of choosing neighbor *f* when destination is *d.*) and decrementing, by normalization, the other probabilities $P_{d'f}$. The amount of the variation in the probabilities depends on a measure of goodness we associate with the trip time $T_{k->d}$, experienced by the forward ant, and is given below. This time represents the only available explicit feedback signal to score paths. It gives a clear indication about the goodness *r* of the followed route because it is proportional to its length from a physical point of view (number of hops, transmission capacity of the used links, processing speed of the crossed nodes) and from a traffic congestion point of view (the forward ants share the same queues as data packets).

The time measure *T*, composed by all the sub-paths elapsed times, cannot be associated with an exact error measure, given that we don't know the "optimal" trip times, which depend on the whole network load status. Therefore, *T* can only be used as a reinforcement-learning field. The reinforcement $r \equiv r(T,M_k)$ is defined to be a function of the goodness of the observed trip time as estimated on the basis of the local traffic model. *r* is a dimensionless value, $r \in (0,1]$, used by the current node *k* as a positive reinforcement for the node *f* the backward ant $B_{d->s}$ comes from. *r* takes into account some average of the so far observed values and of their dispersion to score the goodness

of the trip time *T*, such that the smaller *T* is, the higher *r* is (the exact definition of *r* is discussed in the next subsection). The probability $P_{d'f}$ is increased by the reinforcement value as follows:

$$P_{d'f} \leftarrow P_{d'f} + r(1 - P_{d'f})$$

In this way, the probability $P_{d'f}$ will be increased by a value proportional to the reinforcement received and to the previous value of the node probability (that is, given a same reinforcement, small probability values are increased proportionally more than big probability values, favoring in this way a quick exploitation of new and good, discovered paths).

Probabilities $P_{d'n}$ for destination *d'* of the other neighbor nodes *n* implicitly receive a negative reinforcement by normalization. That is, their values are reduced so that the sum of probabilities will still be 1:

$$P_{d'n} \leftarrow P_{d'n} - rP_{d'n}, n \in N_k, n \neq f$$

It is important to remark that every discovered path receives a positive reinforcement in its selection probability, and the reinforcement is (in general) a non-linear function of goodness of the path, as estimated using the associated trip time. In this way, not only the (explicit) assigned value *r* plays a role, but also the (implicit) ant's arrival rate. This strategy is based on trusting paths that receive either higher reinforcements, independent of their frequency, or low and frequent reinforcements. In fact, for any traffic load condition, a path receives one or more high reinforcements only if it is much better than previously explored paths. On the other hand, during a transient phase after a sudden increase in network load all paths will likely have high traversing times with respect to those learned by the model *M* in the preceding low congestion, situation. Therefore, in this case good paths can be only be differentiated by the frequency of ants. arrivals. Assigning always a positive, but low, reinforcement value in the case of paths with high traversal time allows the implementation of the above mechanism based on the frequency of the reinforcements, while, at the same time, avoids giving excessive credit to paths with high traversal time due to their poor quality.

Probabilities can approach to zero if other probabilities are increased much more often. This is not desirable, because new, better routes will not be discovered quickly. To

solve this problem an exploration probability is used as a minimum value of each probability. An example is 0.1 divided by the number of next nodes. After applying this minimum, the row of probabilities is normalized to 1. this ensures that none of the entries in the probability table will approach zero.

Routing tables are used in a probabilistic way not only by the ants but also by the data packets. This has been observed to improve AntNet performance, which means that the way the routing tables are built in AntNet is well matched with a probabilistic distribution of the data packets over all the good paths. Data packets are prevented from choosing links with very low probability by remapping the *T*'s entries by means of a power function $f(p)=p^{\alpha}$, $\alpha>1$ which emphasizes high probability values and reduces lower ones.

```
l:=Current lime;
t_end:=Time length of the simulation;
Δt:=Time interval between ants generation;
foreach (Node) /* Concurrent activity over the network */
  M:= Local traffic model;
  T:= Node routing table;
  while (t< t_end )
    in_parallel /* Concurrent activity on each node */
      if (t mod Δt = 0)
        Destination_node:=SelectDestinationNode (data_traffic_distribution);
        LaunchForwardAnt(destination_node, source_node);
      end if
      foreach (ActiveForwardAnt[source_node, current_node, destination_node])
        while (current_node≠destination_node)
          Next_hop_node:=SelectLink(current_node,destination_node,T,link_queues);
        PutAntOnLinkQueue(current_node,next_hop_node);
        WaitOnDataLinkQueue(current_node,next_hop_node);
        CrossTheLink(current_node,next_hop_node);
        PushOnTheStack(next_hop_node,elapsed_time);
```

```
      Current_node:=next_hop_node;
    end while
    LaunchBackwardAnt(destination_node, source_node, stack_data);
    Die();
  end foreach
   foreach(ActiveBackwardAnt[source_node current_node, destination_node])
    while (current_node≠destination_node)
      next_hop_node:=PopTheStack();
      WaitOnHighPriorityLinkQueue(current_node,next_hop_node);
      CrossTheLink(current_node,next_hop_node);
      UpdateLocalTrafficModel(M,current_node,source_node,stack_data);
      Reinforcement:=GetReinforcement(current_node,source_node,reinforcement);
    end while
   end foreach
  end in_parallel
 end while
end foreach
```

Above we have a high-level description of the algorithm in pseudo-code. All the described actions take place in a completely distributed and concurrent way over the network nodes (while, in the text, AntNet has been described from an individual ant's perspective). All the constructs at the same level of indentation inside the context of the statement in_parallel are executed concurrently. The processes of data generation and forwarding are not described, but they can be thought as acting concurrently with the ants.

Figure 5 illustrates a simple example of the algorithm behavior. The forward ant, $F_{1->4}$, moves along the path 1 →2 →3 →4 and, arrived at node 4, launches the backard ant $B_{4->1}$that will travel in the opposite direction. At each node *k*, *k* = 3,..,1, the backward ant will use the stack contents $S_{1->4}(k)$ to update the values for M_k (μ_1 , σ_1^2, W_1) and, in case of good sub-paths, to update also the values for M_k (μ_i , σ_i^2, W_i),i=k+1,…,3. At the same time the routing table will be updated by incrementing the

goodness P_{1j}, j=k+1, of the last node *k*+1 the ant $B_{4->1}$ came from for the case of node *i* = *k* + 1,…,4 as destination node, and decrementing the values of *P* for the other neighbors (here not shown). The increment will be a function of the trip time experienced by the forward ant going from node *k* to destination node *i*. As for *M*, the routing table is always updated for the case of node 4 as destination, while the other nodes I'=k+1,…,3 on the sub-paths are taken in consideration as destination nodes only if the trip time associated to the corresponding sub-path of the forward ant is statistically good.

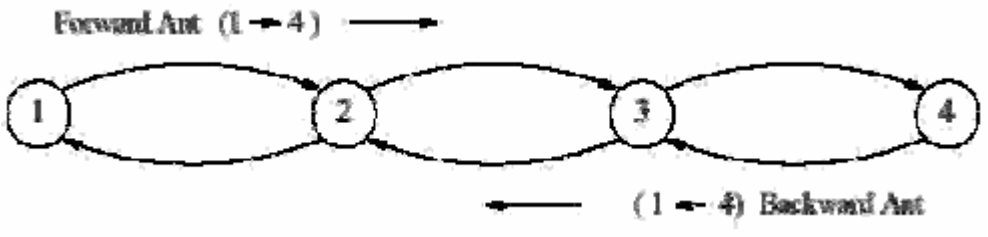


Figure 5

How to score the goodness of the ant's trip time is described as follows.
The reinforcement *r* is a critical quantity that has to be assigned by considering three main aspects:

- paths should receive an increment in their selection probability proportional to their goodness;
- the goodness is a relative measure, which depends on the traffic conditions, that can be estimated by means of the model *M*;
- it is important not to follow all the traffic fluctuations.

This last aspect is particularly important. Uncontrolled oscillations in the routing tables are one of the main problems in shortest paths routing. It is very important to be able to set the best trade-off between stability and adaptation.

r is defined as a function of the ant.s trip time *T*, and of the parameters of the local statistical model *M*:

$$r = c_1\left(\frac{W_{best}}{T}\right) + c_2\left(\frac{I_{\sup} - I_{\inf}}{(I_{\sup} - I_{\inf}) + (T - I_{\inf})}\right)$$

In this equation, W_{best} is the best trip time experienced by the ants traveling toward the destination *d*, over the last observation window *W*. The maximum size of the

window (the maximum number of considered samples before resetting the W_{best} value) is assigned on the basis of the coefficient η of the equation. η weights the number of samples effectively giving a contribution to the value of the μ estimate, defining a sort of moving exponential window. Following the expression for the number of effective samples, was set $|W|_{best}$ =5(c/η)with c<1 (I set *c*=0.8). In this way, the long-term exponential mean and the short-term windowing are referring to a comparable set of observations, with the short-term mean evaluated over a fraction *c* of the samples used for the long-term one. I_{sup} and I_{inf} are convenient estimates of the limits of an approximate confidence interval for $\mu.I_{inf}$ is set to W_{best} , while $I_{sup}= \mu + z \cdot \frac{\sigma}{\sqrt{|W|}}$, with $z=\frac{1}{\sqrt{(1-\gamma)}}$

where γ gives the selected confidence level. The first term in the equation simply evaluates the ratio between the current trip time and the best trip time observed over the current observation window. This term is corrected by the second one, which evaluates how far the value *T* is from *Iinf* in relation to the extension of the confidence interval, that is, considering the stability in the latest trip times. The coefficients *c1* and *c2* weight the importance of each term. The first term is the most important one, while the second term plays the role of a correction. In the current implementation of the algorithm I set *c1* = 0.7 and *c2* = 0.3 the confidence level γ=0.8.

The value *r* obtained from the equation is finally transformed by means of a squash function *s(x)*:

$$s(x)=\left(1+\exp\left(\frac{a}{x|N_k|}\right)\right)^{1}, x \in (0,1], a \in R^{+}$$

$$r \leftarrow \frac{s(r)}{s(1)}$$

Squashing the *r* values allows the system to be more sensitive in rewarding good (high) values of *r*, while having the tendency to saturate the rewards for bad (near to zero) *r* values: the scale is compressed for lower values and expanded in the upper part.

In such a way an emphasis is put on good results, while bad results play a minor role. The coefficient $\frac{a}{|N_K|}$ determines a parametric dependence of the squashed reinforcement

value on the number $|N_k|$ of neighbors of the reinforced node k: the greater the number of neighbors, the higher the reinforcement. The reason to do this is that we want to have similar, strong, effect of good results on the probabilistic routing tables, independent of the number of neighbor nodes.

3.2. ABC-AdHoc

Lacking the backward ants, the ABC algorithm, might be more suitable for routing in an Ad-Hoc wireless network. In dynamic environment, because of the nodes mobility, the path discovered by the forward ant can happen not to be available for the backward ant. We compared a forward-backward version of the ABC, with the AntNet, in a static network. AntNet proved to be better and more adaptive. This is not only because of the extra local traffic statistics, which a node is maintaining, but also because of the more complex formulae used for maintaining the system.

For a wireless ad-hoc network we create a new algorithm (ABC-AdHoc) adding to the ABC some features from the AntNet. Moreover, a new structure was added to the local data system of a node. It is an array U_d, where for each possible destination *d* in the network, is stored the last time when the actual node received traffic from the node *d*. By the concept traffic we mean not only agents but also packets. If the time U_d becomes higher than a certain value (ex. twice the ant generation period multiplied by the number of nodes in a network), the destination *d* is considered unreachable and the actual node stops generating packets for it. Also the buffer structure is changed. Besides the two buffers for each neighbor (a high priority one for ants and a low priority for packets), a new waiting buffer is added. Here are stored for a while the packets for which the destination is seen as unreachable by the node.

The ABC-AdHoc algorithm works as follows:

1. The mobile agents $F_{s\to d}$ are launched at regular time intervals from every network node *s*.

2. Each ant keeps a memory about its path (visited nodes). When an ant arrives in a node *i* coming from a node *j*, it memorizes the identifier of the visited node (*i*) and the trip time necessary for ant to travel from node *i* to *j*. These data are pushed onto the memory stack $S_{s\to d}(i)$ where:

$$T_{i\to j}[s] = \frac{q_j + Size(F_{s\to d})}{Bandwidth_{i\to j}} + D_{i\to j}$$

- $T_{i\to j}[s]$ is the time (virtual delay) that a packet of its size would have to wait to move from node *i* to get to the previous node *j*;
- q_j [bits] is the length of the packets buffer queue towards the link connecting node *i* and its neighbor *j*;
- $Bandwidth_{i\to j}$ is the bandwidth of the link between *i* and *j* in [bit/s];
- $Size(F_{s\to d})$ [bits] is the size of the forward ant $F_{s\to d}$;
- $D_{i\to j}$ [s] is the propagation delay of the link $i\to j$.

2. When an ant comes in the node *i*, it has to select a next node *n* to move to. The selection is done according with the probabilities P_d and the traffic load in the node *i*.

$$P'_{dn} = \frac{P_{dn} + \alpha l_n}{1 + \alpha(|N_i| - 1)}$$

$l_n \in [0,1]$ is a normalized value proportional to the amount x_n (in bits waiting to be sent) of the incoming traffic from the link connecting the node *i* with its neighbor *n*:

$$l_n = 1 - \frac{x_n}{\sum_{j=1}^{|N_i|} x_j}$$

This happens because we can have completely different traffic load between one way the ant is traveling and the other way where the probabilities are updated. We set $\alpha = 0.4 \in [0,1]$.

3. If the ant reaches a node where its destination *d* is seen unreachable, it is killed.
4. In every node *i*, the forward ant $F_{s\to d}$ updates the data structures of the node: the local traffic statistics and the routing table for the source node *s*, and also for the other nodes the $F_{s\to d}$ visited.

First the value U_s is refreshed. If the trip time $T_{i\to s} < I_{\sup}(s)$ where:

$$I_{\sup}(s) = \mu_s + z\frac{\sigma_s}{|W_s|}$$

$z = \frac{1}{\sqrt{1-\gamma}}, \gamma = 0.8 \in [0,1)$ gives a selected confidence level.

The updating process goes on, following the steps:

a. the local data structure is changed:

$$\mu_S = \mu_S + \eta(T_{i \to s} - \mu_s)$$

where $\eta = 0.1$ is a value that makes the old measurements to defuse in time (like the pheromone).

b. a reinforcement value *r* is computed as follows:

$$r = c_1 \cdot \frac{I_{\inf}(s)}{T_{i \to s}} + c_2 \cdot \frac{I_{\sup}(s) - I_{\inf}(s)}{\left(I_{\sup}(s) - I_{\inf}(s)\right) + \left(T_{i \to s} - I_{\inf}(s)\right)}$$

$$r = \frac{1 + e^{\frac{a}{|N_i|}}}{1 + e^{\frac{a}{r \cdot |N_i|}}}$$

where for an entry s, I_{inf} (s) = min{W_s}, the best value in the observation window W_s. c1 = 0:7; c2 = 0:3; a = 5 are the values we used for these parameters.

c. The probabilities are updated with the reinforcement value, $P'_{sk} = P_{sk} + r(1 - P_{sk})$; k = j the previous node chosen by the ant $P'_{sk} = P_{sk} - r \cdot P_{sk}$, for j≠n.

5. If a cycle is detected, that is, if an ant is forced to return to an already visited node the ant is killed. But this is not enough. In the new algorithm killing the ant just when the cycle is found is too late. The changes in the tables are already done. So a cycle should be avoided. This can be done also comparing, in every node i, on the way of $F_{s \to d}$, for all the items k on the ant stack, if $T_{i \to k} > I_{\sup}(k)$. In this case the path the ant followed from k to i, is very possible not to be the best one, according with the local information in node i. This means that the ant started to get closer to the already visited node k, and might end in a cycle after some steps. We decided to kill the ant in this situation, unless the ant is very young (it visited less than 3 other nodes).

6. When the destination node d is reached, the agent $F_{s \to d}$ is making the updates and dies.

The packets have a different behavior than the ants. In the nodes they don't share the same buffers with the ants, having a lower priority. They are routed to destination according with the probabilities. When a packet P $K_{s \to d}$ arrivesin a node i, the value U_s in this node is refreshed.

3.3 Simulation parameters

As it could be noticed, there is quite a big number of parameters, which should be set for the simulation.

The general parameters and their default values are:

- the duration of the simulation: Test time = 100 s.
- the duration of the routing table training: Training time = 10 s.
- the mean of the Poisson distribution for the arriving interval of a new packet:

Packet generation period = 5 ms.

- the period of time between two generation of ants, at one node:

Agent generation period = 300 m.

- the maximum living time for a packet or ant: Traffic max age = 15 s.
- the size of an ant at the birth time: Mean agent size = 192 bits. This size is increased with 64 bits for every visited node.
- the mean of the exponential distribution for the interval between two

generation of packets, at one node: Mean packet size = 4 Kbits.

- the bandwidth of the network links: Link bandwidth = 1.5 Mbits/s.
- the buffer capacity of one node: Node buffer capacity = 1 Gbits.
- the transmission delay on the network links: Link delay = 10 ms.
- the exploration probability used for computing the lower bound for an entry in the probability table: Exploration probability = 0.1.

The parameters specific to AntNet are:

- the parameter αused by ants for computing probabilities before choosing the next node to go: Alfa = 0.4.
- another parameter denoted with α used for mapping the probabilities when a packet has to select the next node to go: Mapping alfa = 3.
- the η parameter used for computing the values for μ and σ^2 of M_k

structures: Eta = 0.1.

- the confidence level γ used for computing the reinforcement r:

Confidence level=0.8

• the parameter *a* used by the squashing function for squashing *r*:

Squash parameter = 5.

• the parameter *c* used in the formula for |*W*|max: Const c = 0.8.

• *c1* used for computing r: Const C1 = 0.7.

• *c2* used for computing r: Const C2 = 0.3.

The parameters specific to Forward and Forward-Backward algorithm are:

• constant *a* used in the formula of ΔP : const a = 2.

• constant *b* used in the formula of ΔP : const b = 0.005.

4. The Simulator

For converting the simulator from Delphi to Java we used Netbeans 3.5.1. The application has two windows:

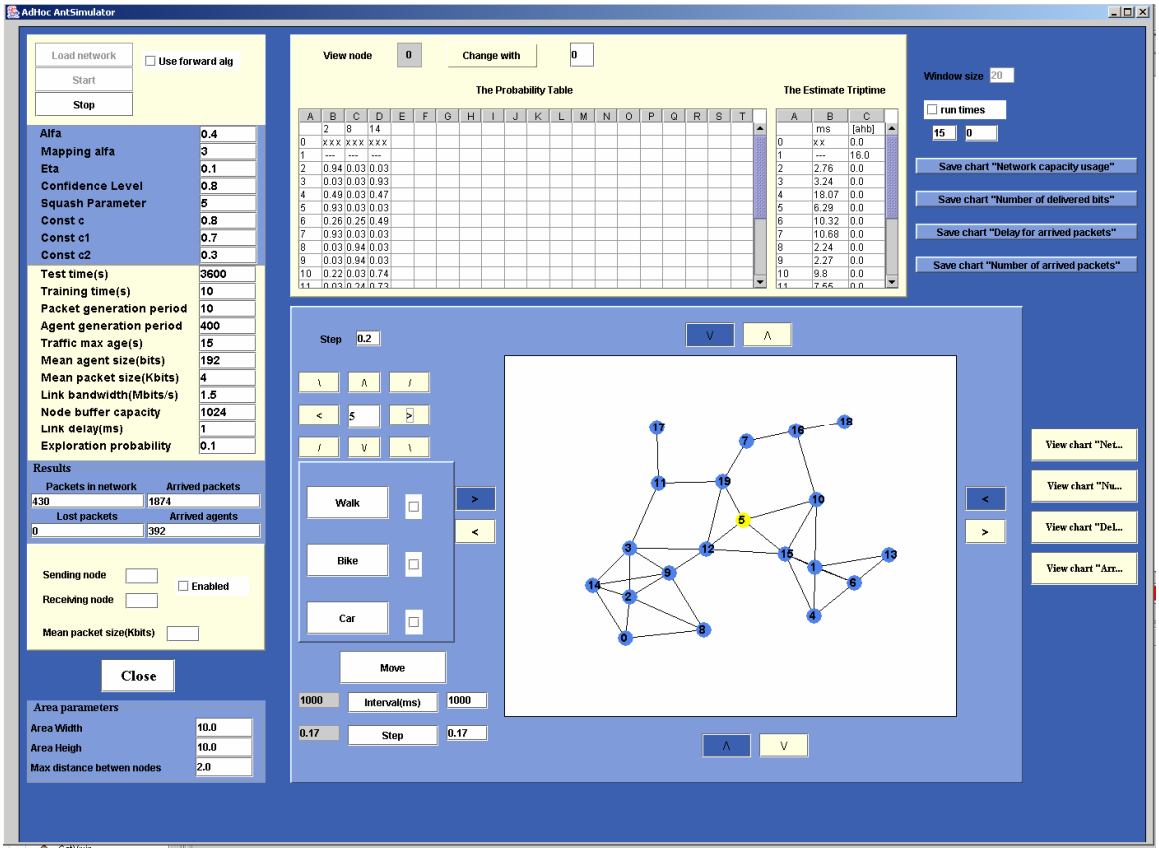


Figure 1

In the left up corner there are three buttons (Figure 1). The first one is for loading the network. This is done from a text file where on the first line is the number of nodes. The following lines in these text files contain pairs of numbers corresponding to every node of the network. Each pair contains the coordinates of that node. Only after a network file is loaded, the Start button can be pressed. The last button is for stopping the simulation.

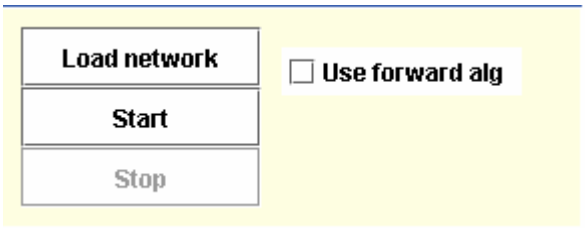


Figure 2

Near these buttons there is a CheckBox. If it is checked then we will use in the simulation the Forward algorithm otherwise we will use the Forward-Backward algorithm.

Under this there is a Panel(see Figure 3) where can be set the values for the parameters specific to AntNet.

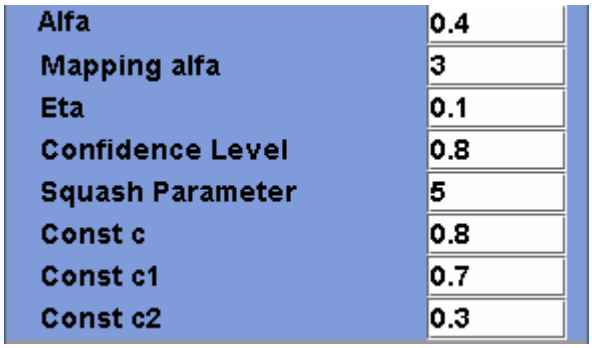


Figure 3

Below in the application window (Figure 4) is the Panel with the network parameters and the parameters common for all the algorithms. All the parameters have the default values mentioned in the previous chapter. But of course they can be changed. I suggest that only the general parameters to be changed for making different tests. The default values for the parameters shown in figure 8, represent the optimal values
(or something close to it) in each algorithm. Changing significantly these values will decrease the performance of the algorithm, which uses the parameters.

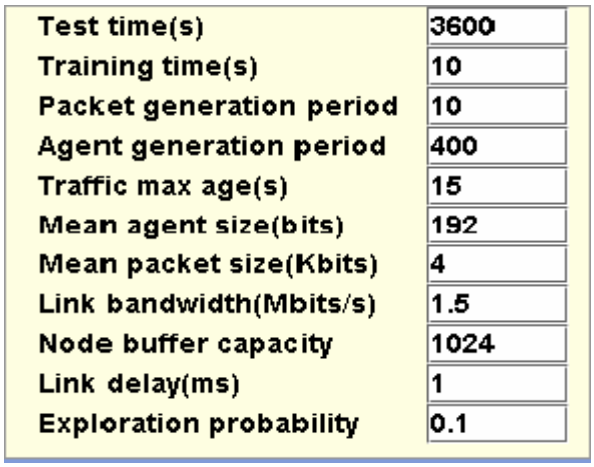


Figure 4

Then follows a Panel (Figure 5) where are displayed four values which represents:

- the total number of packets, which are waiting in the buffers of the Network's nodes.
- the number of the packets, which have arrived at their destination since the beginning of the simulation.
- the number of packets lost since the start of the simulation.
- the number of backward agents, which reached their destination since the beginning of the simulation.

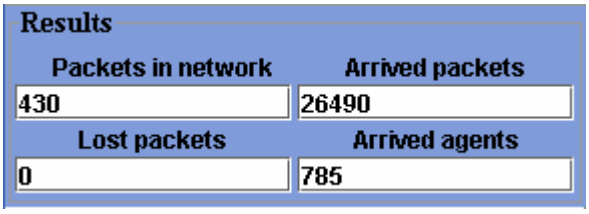


Figure 5

Also these values are an evaluation criterion for the implemented algorithms.

The next Panel (Figure 6) is used for defining a hot connection. This means that the sending node mentioned here will generate packets only with the receiving node as destination. Also the mean of these packets size can be changed. I used this connection to generate congestion only in several nodes of the network and to test the capacity of the algorithms for finding alternative routes.

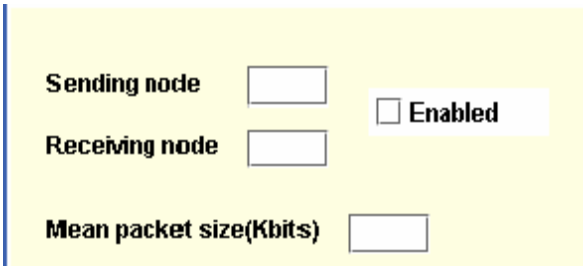
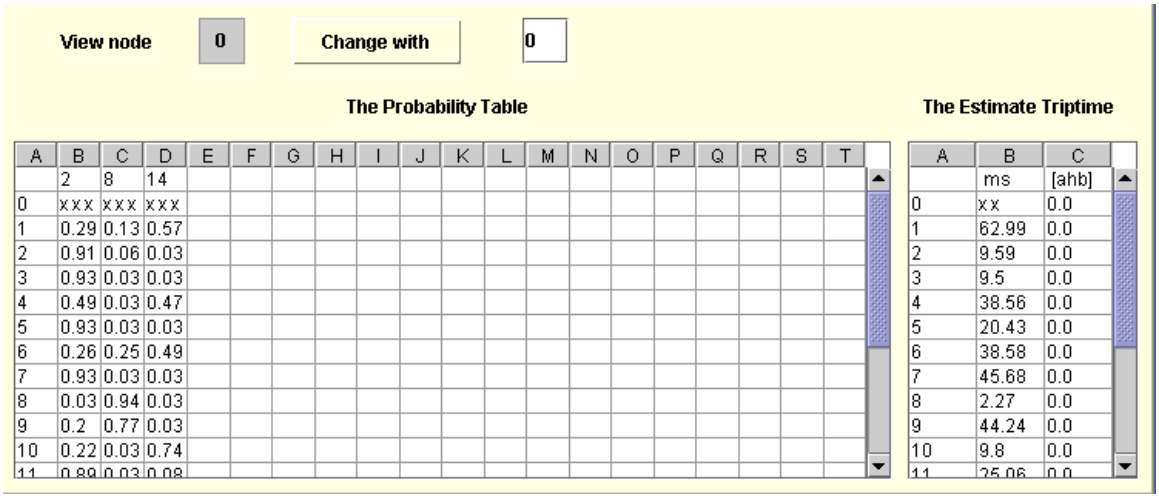


Figure 6

The last Panel is presented in Figure 7. It is used for displaying the routing table of a selected node (View node). This node can be changed pressing the button. The current node will become the one which number was inserted in the TextField situated on the right side of the button. See that all the entries of it as destination are equal. You can notice that not all the sums on a line are 1. This is because the values are truncated when they are displayed. If AntNet is used also the estimated trip times (in milliseconds) from the current node to every destination are shown. These values are initialized at the beginning of the simulation with a value equal with the number of links multiplied with the delay of the links. A value bigger than the live time of a packet shows that on that the packets traveling to that destination will probably be lost.



A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T
	2	8	14																
0	xxx	xxx	xxx																
1	0.29	0.13	0.57																
2	0.91	0.06	0.03																
3	0.93	0.03	0.03																
4	0.49	0.03	0.47																
5	0.93	0.03	0.03																
6	0.26	0.25	0.49																
7	0.93	0.03	0.03																
8	0.03	0.94	0.03																
9	0.2	0.77	0.03																
10	0.22	0.03	0.74																
11	0.89	0.03	0.08																

A	B	C
	ms	[ahb]
0	xx	0.0
1	62.99	0.0
2	9.59	0.0
3	9.5	0.0
4	38.56	0.0
5	20.43	0.0
6	38.58	0.0
7	45.68	0.0
8	2.27	0.0
9	44.24	0.0
10	9.8	0.0
11	25.06	0.0

Figure 7

For each of the three evaluation criterions described previous, a graph as in Figure 8 is drawn. The graphs can be saved or just only displayed during the simulation.

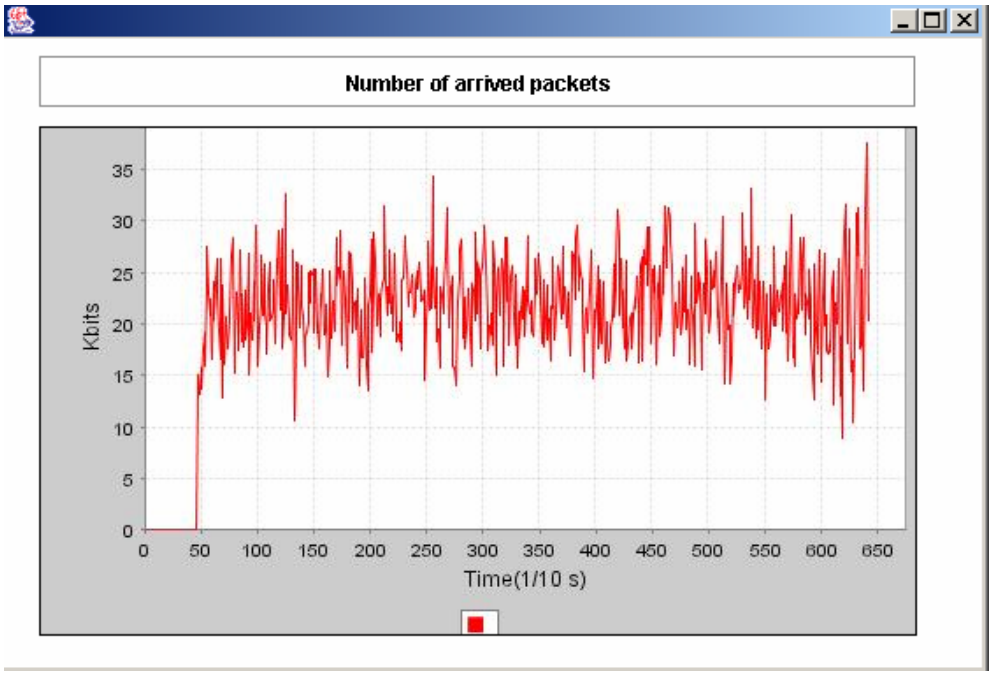


Figure 8

If you want a certain graph to be saved, you have to click on the corresponding button.

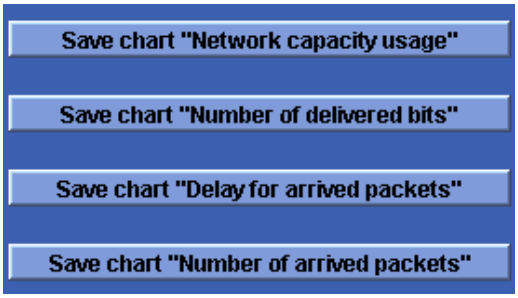


Figure 9

But if you want to see a certain graph, you have to click on one of the buttons below.The first one will display the chart for the “Network capacity usage”, the second one the “Number of delivered bits”, the third one the “Delay for arrived packets”, and the fourth : the “Number of arrived packets”.

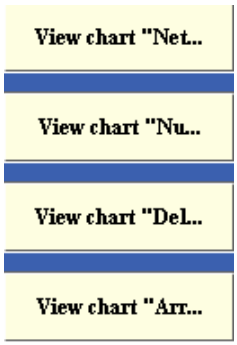


Figure 10

We also have drawn the network's topology, consisting in the nodes and the links between them.

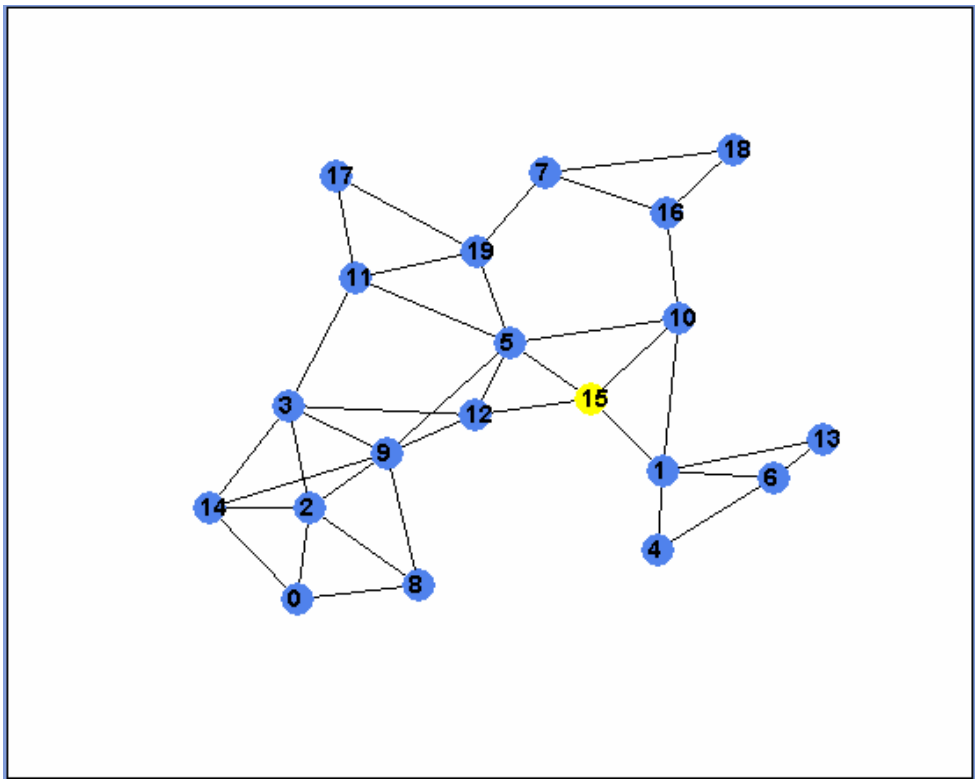


Figure 11

The nodes can be moved in different ways:

- a certain node can be moved in any direction, by pressing the eight buttons showed in the Panel below:

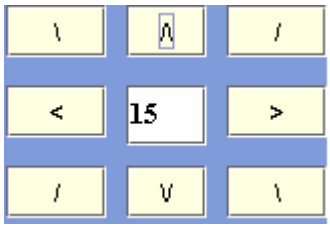


Figure 12

- the type of movement of the network's nodes can be set by pressing one of the buttons below:

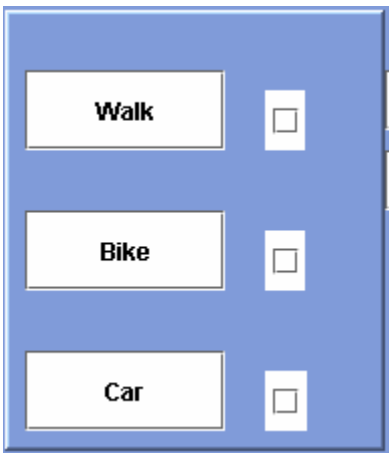


Figure 13

All the nodes will start moving in a random direction with the corresponding speed: if the walk button is pressed this speed will be 5 km/h, if the car button is pressed it will be 15 km/h and if the car button is pressed the speed will be 72 km/h.

- the third mode is by pressing the Move button. In this case all nodes will move in a random direction with the step given by the value introduced in the TextField situated at the Step button's right at every interval of time introduced in the TextField near Interval button.

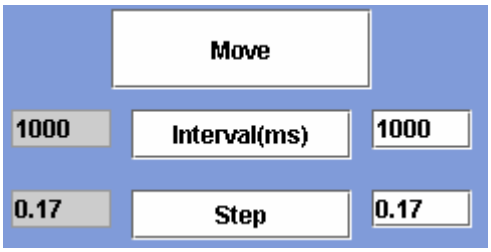


Figure 14

5. Crowd Control Simulation

5.1 Introduction

Many human activities draw huge numbers of people to the same location. It is important to control the flow of people in order to avoid stampedes.

Controlling crowds in airports, train terminals, sporting events, etc., is a complex problem. This particular problem has a great deal interaction between the entities themselves (i.e. among the individual members of the crowd) and the crowd (or individuals) with the environment in which the crowd is placed. These interactions are best observed when the simulation output has a graphical component. Therefore, the combination of a simulation model with a good graphical representation will facilitate the understanding of the behavior of this complex system.

The simulation focuses on the construction of such a model able to reproduce some of the characteristics of crowds.

Crowd control can be applied to many different areas, such as police operations, shopping center design, public park re-organization, rail station re-engineering and epidemic diffusion.

In order to develop a model which can be used in the design of public facilities we have to take into account some of his characteristics.

Efficiency plays a major role in facilities design since moving people into and out of the facility has a direct effect on the usability of the facility (crowds leaving and entering a stadium are examples of this). Finally, quality of service plays an important role in facility layout. The more enjoyable to the user (i.e. the crowd) a facility is, the more used the facility will be.

Before building a model we have to understand the crowd dynamics and the relationship of crowding to facility design and management.

An accepted definition of crowd is that of a large group of individuals in the same physical environment, sharing a common goal (e.g. people going to a rock show or a

football match). The individuals in a crowd may act in a different way than when they are alone or in a small group.

Understanding crowding is important because numerous incidents have been recorded in which uncontrolled crowding has resulted in injuries and, in some instances, death.

Crowds of non-combatants play a large and increasingly recognized role also in modern military operations, and often create substantial difficulties for the combatant forces involved.

The fields in which crowd control can be used are different and shows us the usability of this concept.

5.2 Simulation

I tried to model the crowd control on the base of the AntNet Algorithm.

In this algorithm I use a network of nodes which communicate each other via wireless *links***.** I assume that two nodes are connected if the distance between them is smaller than a given *maximum distance***.**

Every node stores a data structure, called routing table holding all the information used by the algorithm to make the local forwarding decisions.

In my model humans are represented by nodes which have a certain position in space given by the following coordinates (x_i, y_i).

Every node has a leading node, which he must follow. The leading node is obtained from the routing table, representing the node with the highest probability to be chosen as the next node for all the possible destinations.

If the distance between a node and his leading node overflow a certain level, the node has to follow it's leader.

The aim of this simulation is to avoid collision while moving the nodes in a certain direction and also to keep together the nodes which depend on each other.

The collision could appear in two situations:

a) between two nodes
b) between a node and the border of the given surface.

In order to avoid these collisions we have to adopt certain decisions, according to the situation:

-before moving a node, we verify whether is a valid movement, meaning that the node has to remain inside the given area; otherwise the node won't be moved.

-in the first case we move the other node keeping the same direction and the same step as the node which caused the collision; if appears other collision, we repeat the process, while is possible;

The nodes are moved in two ways:

-by the user

-as a consequence of a movement, in order to avoid collision or to follow the leading node

In the first case the node is moved in the desired direction on a distance equal to the step value. In the second one, the movement is made in a different way. If the nodes are in collision, the node will moved on the line which connects the two nodes, but in the opposite direction. Otherwise, if a node has to follow the moving node then it will be moved on the same line but towards the node.

When a node is moved, also the nodes which depend on it, have to be moved.

So, when we move a node, we have to take care of some aspects:

-the node shouldn’t overflow the borders of the area

-it has to avoid collision with other nodes

-the nodes which depend on it, have to follow it.

In order to simulate a crowd the borders of the given surface can be modified and the nodes have to move according to them.

If a border moves towards a node, the node has to move in the same direction .If the border moves in the opposite direction then the nodes which are too close have to follow it.

By moving the borders, the network topology is changing.

The way in which I implemented this model is described below:

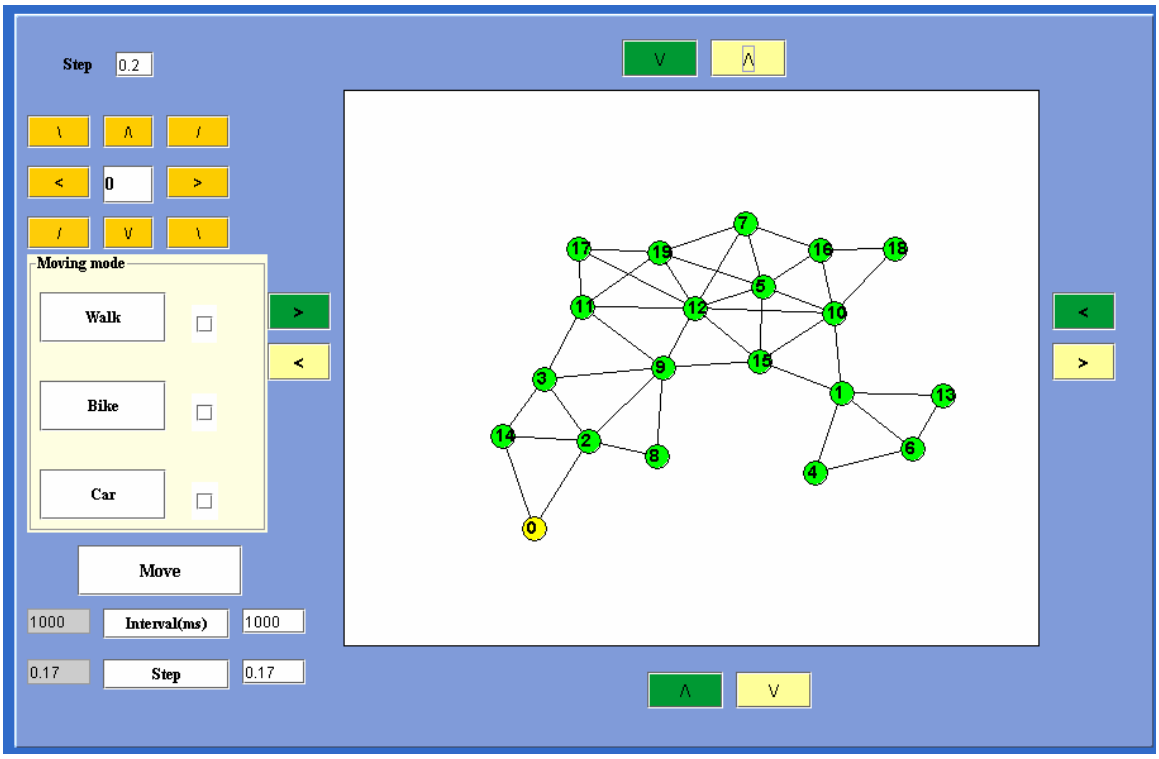


Figure 15

A certain node can be moved in any of the eight directions, by pressing the corresponding orange button. A node can be selected in two ways: by a mouse click or by introducing it's number in the jTextField surrounded by the orange buttons. In both cases the node's color will change into yellow.

The step after which the nodes are moving can be changed, by introducing the desired value in the jTextField from the right corner of the window.

The borders of the given area can be modified by pressing the buttons next to them.

The border will move in the direction indicated on the button(towards the nodes or back to it's initial position), as you can see in the figure below:

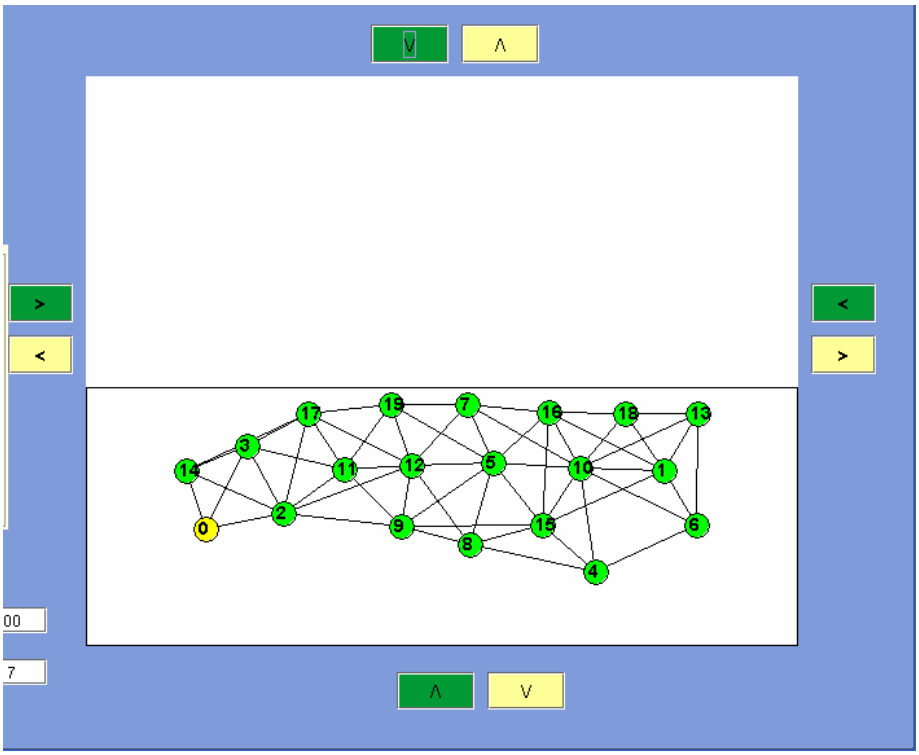


Figure 16

When the nodes become too close and collision can't be avoided, the borders will stop moving. In this situation clicking on the buttons for moving the borders, will produce no action. Still, the borders can be moved in the opposite direction and you can observe the way in which nodes are moving, till there isn't any collision left.

So, by moving the borders you can orientate the crowd in the desired direction.

5.3 Results

5.3.1 DelftNet

In order to test the algorithm, I run the simulator in different situations. First I moved the nodes, till they become crowded and secondly I let them moving in a random way.

The differences can be seen in the graphs below: