Annex I

A1-Project proposal

Project Title:

Can nature help in the routing problem?

Project Aims and Objectives:

Determinate the viability of bio-inspired protocols

Research Area of Study:

Networking ad-hoc protocols

Methodology:

Research the existing network protocols. Advantages and disadvantages.

Make protocol. Design new Protocol. Create model. Write simulation package.

Compare from simulation and get results. Compare existing protocols with new.

Project Outcomes and Deliverables:

Analysis of routing traditional

Protocol bio-inspired capable

Simulation of new protocol

Create results

Metric comparative (traditional/bio-inspired) protocol

Project Timetable:

11 December: Reviews: Studying and analyse the IP Packet structure, algorithms and protocols used in routing,

25 January: Studying and propose a new algorism (and protocols) bio inspired.

8 February: Discuses and determinate parameters to compare:

1 March: Create the bio inspired algorithm

15 March: Compare the algorithm/protocols and get a conclusion:

29 March: Writing Report

26 April: Finish review 0

A2.1- WAntNet.cc

```
#include <stdio.h>
#include "tcl.h"
#include "antnet.h"

#include <vector>
#include <algorithm>

// Debug output macros
#define DEBUG_ANTNET(msg)
/*
#define DEBUG_ANTNET(msg)      if (node_address() == OID) \
        cout << node_address() <<":"<< CURRENT_TIME <<" # "<< __FUNCTION__ <<" # "<< msg
<<"\n"
*/
#define DEBUG_ANTNET_ANT(msg)
/*
#define DEBUG_ANTNET_ANT(msg)  if (node_address() == OID) \
        cout << node_address() <<":"<< CURRENT_TIME <<" # "<< __FUNCTION__ <<" # "<< msg
<<"\n"
*/
#define DEBUG_ANTNET_DATA(msg)
/*
#define DEBUG_ANTNET_DATA(msg) if (node_address() == OID) \
        cout << node_address() <<":"<< CURRENT_TIME <<" # "<< __FUNCTION__ <<" # "<< msg
<<"\n"
*/

RNG ANTNET::randomgen_;

/*************************************************************
 *       T C L   C O M M A N D S   &   I N I T
 *************************************************************/

static class ANTNETClass : public TclClass
{
public:
        ANTNETClass() : TclClass("Agent/ANTNET") {}

        TclObject* create(int argc, const char*const* argv)
        {
                assert(argc == 5);
                return (new ANTNET((nsaddr_t) Address::instance().str2addr(argv[4])));
        }

        void bind()
        {
                TclClass::bind();
                add_method("num-nodes");
                add_method("service-time");
                add_method("seed");
                add_method("FA-genrate");
                add_method("eta");
                add_method("max-window-size");
                add_method("confidence-level");
                add_method("c1-c2");
```

```
                add_method("alfa");
                add_method("remap-prob-factor");
                add_method("squash-factor");
                add_method("choose-nexthop-uniformly-threshold");
                add_method("max-reinforcement");
                add_method("max-packets-prioqueue");
                add_method("max-packets-normalqueue");
                add_method("max-packets-rescued");
                add_method("allowed-hello-loss");
                //add_method("");
        }

} class_ANTNET;

// Constructor
ANTNET::ANTNET(nsaddr_t id) : Agent(PT_ANTNET),
        rtable(this),
        ltm(this),
        ndpc(this),
        SS_timer(this),
        FA_timer(this),
        PQ_timer(this),
        LN_timer(this)
{
        node_address_ = id;
        node_ = (MobileNode*)Node::get_node_by_address(id);

        // Clean up queues and destinations list
        //PrioInQueue_.clear();
        //PrioOutQueue_.clear();
        //NormalInQueue_.clear();
        //NormalOutQueue_.clear();
        //LostNeighborBuffer_.clear();
        destinations_.clear();

        logtarget_ = 0;
        ifqueue_   = 0;

        // Debugging stuff
        sequence_counter_ = 0;
        all_pkt_counter_  = 0;
        data_pkt_counter_ = 0;
        recv_pkt_counter_ = 0;
        forw_pkt_counter_ = 0;
        salv_pkt_counter_ = 0;
        ndp_pkt_counter_  = 0;
        fa_pkt_counter_   = 0;
        ba_pkt_counter_   = 0;
        max_ifq_len       = 0;
}
ANTNET::~ANTNET()
{
        // Clean up
        destinations_.clear();
}

// ### ANTNET parameters (from Di Caro & Dorigo) ###
//
// Default values are listed below. Should be changed,
```

```
// if needed, by each experiment script.
//
float      ANTNET::SERVICE_TIME      = 0.001;
float      ANTNET::FA_INTERVAL       = 0.3;
float      ANTNET::ETA               = 0.1;
u_int32_t ANTNET::MAX_WINDOW_SIZE = 50;
float      ANTNET::CONFIDENCE_LEVEL  = 0.95;
float      ANTNET::ZETA              = ZETA(0.95);
float      ANTNET::C1                = 0.7;
float      ANTNET::C2                = 0.3;
float      ANTNET::ALFA              = 0.3;
float      ANTNET::REMAP_PROB_FACTOR = 1.2;
float      ANTNET::SQUASH_FACTOR_A   = 40.0;
float      ANTNET::CHOOSE_NEXTHOP_UNIFORMLY_THRESHOLD = 0.1;
float      ANTNET::MAX_REINFORCEMENT = 0.9;
u_int32_t ANTNET::MAX_PACKETS_PRIOQUEUE   = 50;
u_int32_t ANTNET::MAX_PACKETS_NORMALQUEUE = 50;
u_int32_t ANTNET::MAX_PACKETS_RESCUED     = 50;
u_int32_t ANTNET::ALLOWED_HELLO_LOSS      = 3;


int ANTNET::command(int argc, char const *const *argv)
        // Process commands fired at TCL-level
{
        // Access to TCL layer
        Tcl& tcl = Tcl::instance();

        if (argc == 2) {

                if (strcmp(argv[1], "start") == 0)
                {
                        // Start simulation timers
                        SS_timer.handle((Event*) 0);
                        return TCL_OK;
                }

        } else if (argc == 3) {

                if (strcmp(argv[1], "seed") == 0) {
                        // Seed random generator
                        unsigned long s;
                        if (sscanf(argv[2],"%ld", &s) == 1) {
                                randomgen_.set_seed(RNG::RAW_SEED_SOURCE, s);
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Seeding failed.");
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "num-nodes") == 0)
                {
                        int num_nodes;
                        if (sscanf(argv[2],"%d", &num_nodes) == 1)
                        {
                                // Add all nodes to destinations list
                                for (nsaddr_t d=0; d<num_nodes; d++)
                                        addDestination(d);
                                return TCL_OK;
                        }
```

```
                        tcl.add_errorf("ANTNET::command: Invalid number of nodes %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "service-time") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::SERVICE_TIME) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid service time %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
          if (strcmp(argv[1], "FA-genrate") == 0)
                {
                        float FA_genrate = 0.0;
        if (sscanf(argv[2], "%f", &FA_genrate) == 1)
                        {
                                if (FA_genrate > 0.0)
                                        ANTNET::FA_INTERVAL = 1 / FA_genrate;
                                else //stop sending FA (put 900s as interval)
                                        ANTNET::FA_INTERVAL = 900.0;
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Invalid packet generation rate %s.\n",
argv[2]);
                        return TCL_ERROR;
      }
                else
                if (strcmp(argv[1], "eta") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::ETA) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid ETA coefficient %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "max-window-size") == 0)
                {
                        if (sscanf(argv[2],"%d", &ANTNET::MAX_WINDOW_SIZE) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum window size %s.\n",
argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "confidence-level") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::CONFIDENCE_LEVEL) == 1)
                        {
                                ANTNET::ZETA = ZETA(ANTNET::CONFIDENCE_LEVEL);
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Invalid confidence level %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "alfa") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::ALFA) == 1)
```

```
                        {
                                assert(ANTNET::ALFA <= 1.0);
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Invalid ALFA factor %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "remap-prob-factor") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::REMAP_PROB_FACTOR) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid remapping probabilities factor
%s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "squash-factor") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::SQUASH_FACTOR_A) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid squash factor A %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
          if (strcmp(argv[1], "choose-nexthop-uniformly-threshold") == 0)
                {
        if (sscanf(argv[2], "%f", &ANTNET::CHOOSE_NEXTHOP_UNIFORMLY_THRESHOLD) == 1)
                        {
                                assert(ANTNET::CHOOSE_NEXTHOP_UNIFORMLY_THRESHOLD
<= 1.0);
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Invalid threshold to choose nexthop
uniformly %s.\n", argv[2]);
                        return TCL_ERROR;
      }
                else
                if (strcmp(argv[1], "max-reinforcement") == 0)
                {
                        if (sscanf(argv[2],"%f", &ANTNET::MAX_REINFORCEMENT) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum reinforcement %s.\n",
argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "max-packets-prioqueue") == 0)
                {
                        if (sscanf(argv[2],"%d", &ANTNET::MAX_PACKETS_PRIOQUEUE) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum DATA and FA queue
length %s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "max-packets-normalqueue") == 0)
                {
```

```
                        if (sscanf(argv[2],"%d", &ANTNET::MAX_PACKETS_NORMALQUEUE) ==
1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum BA queue length %s.\n",
argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "max-packets-rescued") == 0)
                {
                        if (sscanf(argv[2],"%d", &ANTNET::MAX_PACKETS_RESCUED) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum packets to rescue %s.\n",
argv[2]);
                        return TCL_ERROR;
                }
                else
                if (strcmp(argv[1], "allowed-hello-loss") == 0)
                {
                        if (sscanf(argv[2],"%d", &ANTNET::ALLOWED_HELLO_LOSS) == 1)
                                return TCL_OK;
                        tcl.add_errorf("ANTNET::command: Invalid maximum HELLO packets to lose
%s.\n", argv[2]);
                        return TCL_ERROR;
                }
                else
      if (strcmp(argv[1], "if-queue") == 0)
                {
        ifqueue_ = (PriQueue *) TclObject::lookup(argv[2]);
                        if (ifqueue_ == 0) {
          tcl.resultf("ANTNET::command: Invalid network interface queue %s.\n", argv[2]);
          return (TCL_ERROR);
        }
        return (TCL_OK);
      }
                else
                if (strcmp(argv[1], "port-dmux") == 0)
                {
                        dmux_ = (PortClassifier *)TclObject::lookup(argv[2]);
                        if (dmux_ == 0) {
                                tcl.resultf("ANTNET::command: %s lookup of %s failed\n", argv[1],
argv[2]);
                                return TCL_ERROR;
                        }
                        return TCL_OK;
                }
                else
                if (strcmp(argv[1], "log-target") == 0 || strcmp(argv[1], "tracetarget") == 0)
                {
                        logtarget_ = (Trace*) TclObject::lookup(argv[2]);
                        if (logtarget_ == 0)
                                return TCL_ERROR;
                        return TCL_OK;
                }

        } else if (argc == 4) {

                if (strcmp(argv[1], "c1-c2") == 0)
                {
```

```
                        float temp_c1, temp_c2;
                        if ((sscanf(argv[2],"%f", &temp_c1) == 1) && (sscanf(argv[3],"%f", &temp_c2))
== 1)
                        {
                                assert((temp_c1 + temp_c2) == 1.0);
                                ANTNET::C1 = temp_c1;
                                ANTNET::C2 = temp_c2;
                                return TCL_OK;
                        }
                        tcl.add_errorf("ANTNET::command: Invalid weighting coeficient C1 %s and C2
%s.\n", argv[2], argv[3]);
                        return TCL_ERROR;
                }

        }

        return (Agent::command(argc, argv));
}


/*************************************************************
 *              T I M E R S
 *************************************************************/

void StartSimulationTimer::handle(Event*)
        /** Start timers randomly to avoid collisions. Otherwise,
         *  _every_ node will start sending broadcast hello messages
         *  at the same time.
         */
{
        switch (number_of_times_called)
        {
                case 1: // t=0
                        number_of_times_called++;
                Scheduler::instance().schedule(this, &intr, JITTER_BEGINNING_SIMULATION *
Random::uniform());
                break;
                case 2:
                        // Start sending forward ants
                        agent_->FA_timer.handle((Event*) 0);
                        // Polling queues
                        agent_->PQ_timer.handle((Event*) 0);
                        // Send hello messages and check neighbors
                agent_->ndpc.ntimer.handle((Event*) 0);
                agent_->ndpc.htimer.handle((Event*) 0);
                        break;
                default:
                        fprintf(stderr, "%s: Invalid number_of_times_called (%d)\n", __FUNCTION__,
number_of_times_called);
                        exit(1);
        }
}


/*************************************************************
 *      F O R W A R D   A N T   G E N E R A T I O N
 *************************************************************/

void ForwardAntGenerationTimer::handle(Event*)
```

```
	/** Create and send forward ants at the given rate
	 */
{
	// Start sending forward ants only after
	// some neighbors have been discovered
	if (!running_)
	{
		running_ = true;
		Scheduler::instance().schedule(this, &intr, NEIGHBOR_DISCOVERY_PERIOD);
		return;
	}
	agent_->generateForwardAnt();
	assert(ANTNET::FA_INTERVAL >= 0.0);
	Scheduler::instance().schedule(this, &intr, ANTNET::FA_INTERVAL);
}

void ANTNET::generateForwardAnt()
	/** Generate a new ForwardAnt packet
	 *  and send it to a chosen destination
	 */
{
	// Create packet and access headers
	Packet* p = allocpkt();
	hdr_cmn* ch = HDR_CMN(p);
	hdr_ip*  ih = HDR_IP(p);
	hdr_antnet* ah = HDR_ANTNET(p);

	DEBUG_ANTNET_ANT(">>> *** START ANT GENERATION *** >>>");

	// Choose destination address and prepare Ant data
	nsaddr_t dest = destinationForForwardAnt();
	AntNet_Data* ant = new AntNet_Data(dest);

	// Add current node as a visited one (source node)
	// and look for a neighbor to send the ant towards
	ant->visited_nodes().push_back(node_address());
	nsaddr_t nh_addr = nextHopForForwardAnt(ant);

	// If no neighbor is found, abort FA generation
	if (nh_addr == (int32_t)IP_BROADCAST)
	{
		DEBUG_ANTNET_ANT("No neighbor found. Aborting FA generation.");
		delete ant;
		Packet::free(p);
		return;
	}
	// Otherwise, add nexthop jump to list
	ant->insertJump(nh_addr);
	p->setdata(ant);

	// AntNet header
	ah->ah_type = FORWARD_ANT;
	ah->seq_no = sequence_counter_++;

	DEBUG_ANTNET_ANT("Generating FA["<< ant->visited_nodes().front() <<"->"<< ant->dest()
<<":"<< ah->seq_no << \
		"] [nh_addr="<< nh_addr <<"] [jumps="<< ant->printJumpStack() <<"]");

	// Common header
```

```
	ch->ptype() = PT_ANTNET;
	ch->direction() = hdr_cmn::DOWN;
	ch->size() = IP_HDR_LEN + ant->size() + sizeof(hdr_antnet);
	ch->error() = 0;
	ch->next_hop() = nh_addr;
	ch->prev_hop_ = node_address();

	// IP header
	ih->saddr() = node_address();
	ih->daddr() = nh_addr; // BUG fixed, pkts arrive at daddr, not nexthop, so we put nh_addr as daddr
	ih->sport() = RT_PORT;
	ih->dport() = RT_PORT;
	ih->ttl() = IP_DEF_TTL;

	DEBUG_ANTNET_ANT("<<< *** END ANT GENERATION *** <<<");

	// Put in normal outgoing queue
	qelement e; e.p = p; e.h = NULL;
	NormalOutQueue_.push(e);
}


/*************************************************************
 *
 *	R O U T I N G   M A N A G E M E N T
 *
 *************************************************************/

void ANTNET::recv(Packet* p, Handler* h)
	/** Handle incoming packet
	 */
{
	// For debugging purposes ...

	double now = CURRENT_TIME;
	int nowint = (int) floor(now);
	int frequency = 900;

	if ((nowint % frequency) == 0)
		DEBUG_ANTNET("### Max number pkts in IFQ until now is "<< max_ifq_len <<"
###");
	/*	DEBUG_ANTNET("Packets [ total="<< all_pkt_counter_ \
			<<" [data="<< data_pkt_counter_ <<" [recv="<< recv_pkt_counter_ \
			<<" fw="<< forw_pkt_counter_ <<" salvaged="<< salv_pkt_counter_ \
			<<"]] *** [NDP="<< ndp_pkt_counter_ <<" FA="<< fa_pkt_counter_ \
			<<" BA="<< ba_pkt_counter_ <<"] ]");
	*/
	// End debug

	if (p)
		// Queue packet
		enqueue((qelement){p,h});

	// Process next packet if possible
	//poll_queue();
}

/*************************************************************
 *	Q U E U E   M A N A G E M E N T
```

```
 ************************************************************/

void PollQueueTimer::handle(Event*)
        /** Poll internal queues every SERVICE_TIME seconds
         */
{
        agent_->poll_queue();
        assert(ANTNET::SERVICE_TIME >= 0.0);
        Scheduler::instance().schedule(this, &intr, ANTNET::SERVICE_TIME);
}

void ANTNET::enqueue(qelement e)
{
        struct hdr_cmn *ch = HDR_CMN(e.p);
        if (ch->ptype() == PT_ANTNET) {
                struct hdr_antnet *ah = HDR_ANTNET(e.p);
                // Only BAs have to go on priority queue
                if (ah->ah_type == BACKWARD_ANT)
                {
                        if (PrioInQueue_.size() == ANTNET::MAX_PACKETS_PRIOQUEUE)
                        {
                                DEBUG_ANTNET("PrioInQueue has already
"<<MAX_PACKETS_PRIOQUEUE<<" pkts. Dropping new packet.");
                                drop(e.p, DROP_RTR_QFULL);
                        }
                        else
                                PrioInQueue_.push(e);
                        return;
                }
        }

        // Packet is either a FORWARD_ANT or DATA packet
        if (NormalInQueue_.size() == ANTNET::MAX_PACKETS_NORMALQUEUE)
        {
                DEBUG_ANTNET("NormalInQueue has already
"<<MAX_PACKETS_NORMALQUEUE<<" pkts. Dropping new packet.");
                drop(e.p, DROP_RTR_QFULL);
        }
        else
                NormalInQueue_.push(e);
}

void ANTNET::poll_queue()
        /** Look at all queues and process packets if possible,
         *  either sending them to IFQ or processing locally
         */
{
        /*
        if ((NormalInQueue_.size() > 10) || (NormalOutQueue_.size() > 10) || \
                (PrioInQueue_.size() > 10) || (PrioOutQueue_.size() > 10) || (ifqueue_->length() > 900))
        DEBUG_ANTNET("Queue size [In="<< NormalInQueue_.size() \
                <<" Out="<< NormalOutQueue_.size() <<" PrioIn="<< PrioInQueue_.size() \
                <<" PrioOut="<< PrioOutQueue_.size() <<" IFQ="<< ifqueue_->length() <<"]");
        */
        if (ifqueue_->length() > max_ifq_len)
                max_ifq_len = ifqueue_->length();

        // First, send all prioritary packets, like BAs
        float delay = ARP_DELAY;
```

```
	while (! PrioOutQueue_.empty())
	{
		qelement prio = PrioOutQueue_.front();
		PrioOutQueue_.pop();
		Scheduler::instance().schedule(target_, prio.p, delay);
		delay += ARP_DELAY;
	}
	// Second, send all normal data packets and FAs
	while (! NormalOutQueue_.empty())
	{
		qelement normal = NormalOutQueue_.front();
		NormalOutQueue_.pop();
		Scheduler::instance().schedule(target_, normal.p, delay);
		delay += ARP_DELAY;
	}
	// Third, look if some BA has arrived and process it
	// Note: only one packet at once
	if (! PrioInQueue_.empty())
	{
		// There are some BA, process them all
		qelement next = PrioInQueue_.front();
		PrioInQueue_.pop();
		processPacket(next.p, next.h);
	}
	// Finally, process data packets and FA
	// Note: if we put ELSE then only one packet is processed in SERVICE_TIME secs,
	// otherwise two (2) are processed, one prio and one data
	if (! NormalInQueue_.empty()) {
		qelement next = NormalInQueue_.front();
		NormalInQueue_.pop();
		processPacket(next.p, next.h);
	}
}



/*************************************************************
 *	P R O C E S S I N G  P A C K E T S
 *************************************************************/

void ANTNET::processPacket(Packet* p, Handler* h)
	/** Update list of neigbors and destinations
	 *  and process packet depending on its type
	 */
{
	hdr_cmn *ch = HDR_CMN(p);
	hdr_ip  *ih = HDR_IP(p);

	// Common processing for all packets

	// Debug
	all_pkt_counter_++;

	// Add both src and dest addresses
	// in the destinations list
	// FIXME: should add all possible destinations
	// at the very beginning, so we learn all routes
	// DONE!
	if ((ih->saddr() != (int32_t)IP_BROADCAST) && (ih->saddr() != node_address()))
```

```
                addDestination(ih->saddr());
        if ((ih->daddr() != (int32_t)IP_BROADCAST) && (ih->daddr() != node_address()))
                addDestination(ih->daddr());

        // Debugging info
/*      if (node_address() == OID)
                DEBUG_ANTNET("type=" << ch->ptype() << " [src=" << ih->saddr() << \
                        " dst=" << ih->daddr() << "] [hop prev=" << ch->prev_hop_ << \
                        " next=" << ch->next_hop() << "] *** *** ***");
*/

        // If this is a packet I'm originating, must add IP header
        if (ih->saddr() == node_address() && (ch->num_forwards() == 0))
                ch->size() += IP_HDR_LEN;

        // If it is a Forward/Backward ant, must process it
        if (ch->ptype() == PT_ANTNET)
                processControlPacket(p,h);
        // Otherwise route data packet
        else
                forwardDataPacket(p,h);
}

void ANTNET::forwardDataPacket(Packet* p, Handler* h)
        /** Route data packet towards its destination
         */
{
        hdr_cmn* ch = HDR_CMN(p);
        hdr_ip* ih = HDR_IP(p);

        // Debug
        data_pkt_counter_++;

        // Decrement TTL and drop packet if necessary
        if (--ih->ttl_ < 0)
        {
                drop(p, DROP_RTR_TTL);
                return;
        }

        // Send data packet up to local agent
        if (ch->direction() == hdr_cmn::UP &&
                (ih->daddr() == (int32_t)IP_BROADCAST || ih->daddr() == node_address()))
        {
                recv_pkt_counter_++;
                DEBUG_ANTNET_DATA("Received DATA["<< ih->saddr() <<"->"<< ih->daddr() <<"]
: sending to local agent");
                dmux_->recv(p, NULL);
                return;
        }
        else
        {
                forw_pkt_counter_++;
                ch->direction() = hdr_cmn::DOWN;
                if (ih->daddr() == (int32_t)IP_BROADCAST)
                {
                        ch->next_hop() = IP_BROADCAST;
                        ch->addr_type() = NS_AF_NONE;
                }
```

```
                else
                {
                        nsaddr_t nh_addr = nextHopForDataPacket(ih->daddr());
                        if (nh_addr == (int32_t)IP_BROADCAST) {
                                DEBUG_ANTNET("Forwarding DATA["<< ih->saddr() <<"->"<< ih-
>daddr() <<"] : no route found!");
                                drop(p, DROP_RTR_NO_ROUTE);
                                return;
                        }
                        else
                        {
                                ch->next_hop() = nh_addr;
                                ch->addr_type() = NS_AF_INET;
                        }
                }

                // Set previous hop and forward packet
                ch->prev_hop_   = node_address();
                ch->direction() = hdr_cmn::DOWN;

                DEBUG_ANTNET_DATA("Forwarding DATA["<< ih->saddr() <<"->"<< ih->daddr() <<
\
                        "] to [nh_addr="<< ch->next_hop() <<"]");

                sendNormalPacket(p,h);
        }

        // Process next queued packet (if any)
        //poll_queue();
}

void ANTNET::processControlPacket(Packet* p, Handler* h)
        /** Process ant information to update RT table
         */
{
        // Access common and IP headers, as well as AntNet_Data
        hdr_cmn *ch = HDR_CMN(p);
        hdr_ip  *ih = HDR_IP(p);
        hdr_antnet *ah = HDR_ANTNET(p);
        AntNet_Data* ant = (AntNet_Data*)p->userdata();

        //DEBUG_ANTNET("Received [" << ah->typeDescription() << "] control packet");
        if (ah->ah_type != NDP_HELLO_MSG)
                DEBUG_ANTNET_ANT(">>> *** ENTER >>>");

        switch (ah->ah_type) {

        case NDP_HELLO_MSG:
                ndp_pkt_counter_++;
                // Send to NDP for link layer detection
                ndpc.recv(p);
                // Drop it, as it's only useful to NDP
                Packet::free(p);
                break;

        case FORWARD_ANT:
                fa_pkt_counter_++;
                // New node visited
```

```
                DEBUG_ANTNET_ANT("Received FA["<< ant->visited_nodes().front() <<"->"<< ant-
>dest() <<":"<< ah->seq_no << \
                                "] [jumps="<< ant->printJumpStack() <<"]");
        ant->visited_nodes().push_back(node_address());

                if (node_address() != ant->dest())
                {
                        // Not yet at destination, choose nexthop to visit
                        // and add final address to destinations list
                        addDestination(ant->dest());
                        nsaddr_t nh_addr = nextHopForForwardAnt(ant);
                        if (nh_addr == (int32_t)IP_BROADCAST)
                        {
                                DEBUG_ANTNET("No neighbors found, FA dropped!");
                                drop(p, DROP_RTR_NO_ROUTE);
                                break;
                        }
                        if (ant->longCycleDetected(nh_addr))
                        {
                                DEBUG_ANTNET("Long cycle detected in ant's lifetime, FA
dropped!");
                                drop(p, DROP_RTR_ROUTE_LOOP);
                                break;
                        }

                        // Update ANTNET routing protocol info
                        ant->insertJump(nh_addr);

                        // Update common and IP headers
                        ch->size() = ant->size() + sizeof(hdr_cmn) + sizeof(hdr_antnet);
                        ch->prev_hop_   = node_address();
                        ch->next_hop()  = nh_addr;
                        ch->direction() = hdr_cmn::DOWN; // important!

                        ih->saddr() = node_address();
                        ih->daddr() = nh_addr;

                        DEBUG_ANTNET_ANT("Forwarding FA["<< ant->visited_nodes().front() \
                                <<"->"<< ant->dest() <<":"<< ah->seq_no << \
                                "] [nh_addr="<< nh_addr <<"] [jumps="<< ant->printJumpStack()
<<"]");

                        sendNormalPacket(p,h);
                        break;
                }
                else
                        // Agent is at destination, prepare for backtracking
                {
                        DEBUG_ANTNET_ANT("FA["<< ant->visited_nodes().front() <<"->"<< ant-
>dest() \
                                <<":"<< ah->seq_no <<"] arrived at destination [jumps="<< \
                                ant->printJumpStack() <<"]");

                        // Update ANTNET routing protocol info
                        ah->ah_type = BACKWARD_ANT;
                        ant->arrivaltime() = CURRENT_TIME;
                        // Reset TTL field
                        ih->ttl() = IP_DEF_TTL;
```

```
                        // Continue backtracking ...
                }

        case BACKWARD_ANT:
                ba_pkt_counter_++;
                // Remove current node from visited_nodes
                DEBUG_ANTNET_ANT("Received BA["<< ant->visited_nodes().front() <<"->"<< ant-
>dest() <<":"<< ah->seq_no << \
                        "] [jumps="<< ant->printJumpStack() <<"]");
                ant->visited_nodes().pop_back();

                // Update rtable using travel time spent
                // by ant in all nodes but destination
                if (node_address() != ant->dest())
                {
                        addDestination(ant->dest());
                        rtable.updateOnBackwardAntArrival(ant);
                        ant->removeLastJump();
                }

                // When backtracking is complete, terminate agent
                if (ant->visited_nodes().empty())
                {
                        DEBUG_ANTNET_ANT("BA["<< node_address() <<"->"<< ant->dest()
<<":"<< ah->seq_no << \
                                "] arrived at home [jumps="<< ant->printJumpStack() <<"]");
                        Packet::free(p);
                        break;
                }

                // Move to next node contained in ant stack
                // Must be a neighbor
                for (Neighbor *nb = ndpc.nbhead.lh_first; nb; nb = nb->nb_link.le_next)
                {
                        if (nb->nb_addr == ant->visited_nodes().back()) {
                                // Move (updating common and IP headers)
                                ch->prev_hop_   = node_address();
                                ch->next_hop()  = nb->nb_addr;
                                ch->direction() = hdr_cmn::DOWN; // important!
                                ih->saddr()     = node_address();
                                ih->daddr()     = nb->nb_addr;

                                DEBUG_ANTNET_ANT("Backtracking BA["<< ant-
>visited_nodes().front() <<"->"<< \
                                                ant->dest() <<":"<< ah->seq_no <<"] [nh_addr="<< nb-
>nb_addr \
                                                <<"] [jumps="<< ant->printJumpStack() <<"]");

                                // Put BA in the high priority queue to avoid congestion
                                sendPrioPacket(p,h);

                                // Process next queued packet (if any)
                                //poll_queue();
                                return;
                        }
                }
                // Next node not found
                DEBUG_ANTNET_ANT("Backtracking BA["<< ant->visited_nodes().front() <<"->"<<
ant->dest() << \
```

```
                        ":"<< ah->seq_no <<"] unsuccesful: node " << ant->visited_nodes().back() << "
not found");
                drop(p, DROP_RTR_NO_ROUTE);
                break;

        default:
                printf("ANTNET: Invalid ANTNET type (0x%x) \n", ah->ah_type);
                Packet::free(p);
                break;
        }

        if (ah->ah_type != NDP_HELLO_MSG)
                DEBUG_ANTNET_ANT("<<< EXIT *** <<<");

        // Process next queued packet (if any)
        //poll_queue();
}



/*************************************************************
 *    L O S T   N E I G H B O R S   M A N A G E M E N T
 *************************************************************/

void LostNeighborTimer::handle(Event*)
        /** Purge all packets whose nexthop is still not available
         */
{
        // TODO: when neighbor is lost, entry is removed from rtable
        // we should not erase the info, cause then we lose all the
        // adaptative thing, lost neighbor buffer would not work properly
}

void ANTNET::salvagePacketsFromIFQ(nsaddr_t id)
        /** Some packets were in the ifqueue, but the route to their
         *  destination has changed, so we need to rescue them
         */
{
        Packet *p;
        struct hdr_ip *ih;
        struct hdr_cmn *ch;
        while ((p = ifqueue_->filter(id)))
        {
                ih = HDR_IP(p);
                ch = HDR_CMN(p);

                DEBUG_ANTNET("***");
                DEBUG_ANTNET("*** Salvaging packet from IFQ ["<< ih->saddr() <<"->"<< ih-
>daddr() \
                                <<"] [lost_neighbor="<< ch->next_hop() <<"] ***");
                salv_pkt_counter_++;

                if (ch->ptype() == PT_ANTNET)
                {
                        struct hdr_antnet *ah;
                        ah = HDR_ANTNET(p);
                        // Reroute forward ants
                        if (ah->ah_type == FORWARD_ANT)
                        {
```

```
                                DEBUG_ANTNET("Rerouting forward ant (FA).");
                                rerouteForwardAnt(p, NULL);
                        }
                        else if (ah->ah_type == BACKWARD_ANT)
                        {
                                DEBUG_ANTNET("Dropping backward ant (BA).");
                                drop(p, DROP_RTR_NO_ROUTE);
                        }
                }
                else
                // If the neighbor is lost only for a while, store it
                // in a buffer. If neighbor doesn't reappears, reroute
                {
                        //DEBUG_ANTNET_DATA("Storing data packet due to lost neighbor.");
                        //storePacketLostNeighbor(p);
                        DEBUG_ANTNET("Rerouting data packet.");
                        forwardDataPacket(p, NULL);
                }
                DEBUG_ANTNET("***");
        }
}

void ANTNET::rerouteForwardAnt(Packet* p, Handler* h)
        /** Reroute forward ant (FA) after a neighbor is lost.
         *  Should remove nexthop and departure time from stack
         *  and insert a new one.
         */
{
        // Some verifications ...
        struct hdr_cmn *ch = HDR_CMN(p);
        struct hdr_ip *ih  = HDR_IP(p);
        if (ch->ptype() != PT_ANTNET)
        {
                DEBUG_ANTNET("Packet type is not PT_ANTNET!");
                return;
        }
        struct hdr_antnet *ah = HDR_ANTNET(p);
        if (ah->ah_type != FORWARD_ANT)
        {
                DEBUG_ANTNET("Packet is not a FORWARD_ANT!");
                return;
        }

        // First remove last jump inserted,
        // as nexthop is no longer valid
        AntNet_Data* ant = (AntNet_Data*)p->userdata();
        ant->removeLastJump();

        // Choose another nexthop, provided that
        // there are still some neighbors
        nsaddr_t nh_addr = nextHopForForwardAnt(ant);
        if (nh_addr == (int32_t)IP_BROADCAST)
        {
                DEBUG_ANTNET("No neighbors found to reroute ant, FA dropped!");
                drop(p, DROP_RTR_NO_ROUTE);
                return;
        }
        if (ant->longCycleDetected(nh_addr))
        {
```

```
                DEBUG_ANTNET("Long cycle detected in ant's lifetime, FA dropped!");
                drop(p, DROP_RTR_ROUTE_LOOP);
                return;
        }

        // Update ANTNET routing protocol info
        ant->insertJump(nh_addr);

        // Update common and IP headers
        ch->size() = ant->size() + sizeof(hdr_cmn) + sizeof(hdr_antnet);
        ch->prev_hop_   = node_address();
        ch->next_hop()  = nh_addr;
        ch->direction() = hdr_cmn::DOWN; // important!

        ih->saddr() = node_address();
        ih->daddr() = nh_addr;

        DEBUG_ANTNET_ANT("Rerouting FA["<< ant->visited_nodes().front() <<"->"<< \
                        ant->dest() <<":"<< ah->seq_no <<"] [nh_addr="<< nh_addr \
                        <<"] after lost neighbor [jumps="<< ant->printJumpStack() <<"]");
        sendNormalPacket(p,h);
}

void ANTNET::storePacketLostNeighbor(Packet* p)
        /** Store rescued data packets in a limited buffer. After
         *  LOST_NEIGHBOR_TIMEOUT secs, reroute them
         */
{
        if (LostNeighborBuffer_.size() == ANTNET::MAX_PACKETS_RESCUED)
        {
                DEBUG_ANTNET("Buffer is full. Dropping paquet.");
                drop(p, DROP_RTR_QFULL);
        }
        else
                LostNeighborBuffer_.push((rescued_packet){p,CURRENT_TIME});
}



/*****************************************************************
 *            R O U T I N G   D E C I S I O N S
 *****************************************************************/

nsaddr_t ANTNET::nextHopForDataPacket(nsaddr_t dest)
  /** Choose next hop address for data packets
        *  based on probabilities in routing table
        */
{
        // If destination is a neighbor, send directly (Lookahead)
        Neighbor *nb = ndpc.nbhead.lh_first;
        for (; nb; nb = nb->nb_link.le_next)
                if (nb->nb_addr == dest)
                        return nb->nb_addr;

        NextHopEntry *nh_entry = rtable.getBestNextHopEntryForDataPacket(dest);
        if (nh_entry == NULL) // No route found
                return IP_BROADCAST;
        else
                return nh_entry->addr();
```

```
}

nsaddr_t ANTNET::nextHopForForwardAnt(AntNet_Data* FA)
        /** Choose next hop a forward ant should visit
         */
{
        // Build list of neighbor addresses not already visited
        std::vector<nsaddr_t> unvisited_neighbors;
        for (Neighbor *nb = ndpc.nbhead.lh_first; nb; nb = nb->nb_link.le_next)
        {
                if (nb->nb_addr == FA->dest())
                {
                        // Destination is a neighbor, send directly
                        DEBUG_ANTNET_ANT("Destination is already a neighbor, sending directly");
                        return nb->nb_addr;
                }
                if (find(FA->visited_nodes().begin(), FA->visited_nodes().end(),
                                nb->nb_addr)  ==  FA->visited_nodes().end())
                {
                        // Unvisited (new) neighbor found
                        unvisited_neighbors.push_back(nb->nb_addr);
                }
        }

        // Select which unvisited neighbor to visit next.
        //
        // UNIFORMLY either if:
        //
        // a) All neighbors have already been visited
        //
        bool allow_revisit = false;
        if (unvisited_neighbors.empty())
        {
                for (Neighbor *nb = ndpc.nbhead.lh_first; nb; nb = nb->nb_link.le_next)
                        unvisited_neighbors.push_back(nb->nb_addr);
                allow_revisit = true;
                // If there are still no neighbors, then abort
                // returning IP_BROADCAST (should not happen)
                if (unvisited_neighbors.empty()) {
                        DEBUG_ANTNET("unvisited_neighbors is empty.");
                        return IP_BROADCAST;
                }
        }
        // or
        //
        // b) Uniform variable to discover new paths
        //    is below a threshold
        //
        float explore_new_paths = ANTNET::randomgen_.uniform(0,1);

        DEBUG_ANTNET_ANT("FA["<< FA->visited_nodes().front() <<"->"<< FA->dest() \
                <<"] [nb allow_revisit="<< allow_revisit <<" [unvisited="<< \
                unvisited_neighbors.size() <<" size="<< ndpc.nb_size() \
                <<"]] [explore_new_paths="<< explore_new_paths <<"]");

        if (allow_revisit || (explore_new_paths <
ANTNET::CHOOSE_NEXTHOP_UNIFORMLY_THRESHOLD))
        {
                int   num_nodes     = unvisited_neighbors.size();
```

```
                float uniform_value = ANTNET::randomgen_.uniform(0,1);
                return (nsaddr_t) floor(uniform_value * num_nodes);
        }

        // Otherwise we choose looking at the probabilistic routing table.
        NextHopEntry* nh_entry =
                rtable.getBestNextHopEntryForForwardAnt(FA->dest(), unvisited_neighbors);
        if (nh_entry != NULL)
                return nh_entry->addr();
        else
                return IP_BROADCAST;
}


nsaddr_t ANTNET::destinationForForwardAnt()
        /** Choose destination node address for forward ant.
         *  Can choose uniformly or looking at the node queue
         */
{
        // Abort if destinations vector is empty
        if (destinations_.size() == 0)
        {
                DEBUG_ANTNET_ANT("No destinations available.");
                return IP_BROADCAST;
        }

        // Choose uniformly
        int   num_nodes     = destinations_.size();
        float uniform_value = 0.0;

        nsaddr_t dest = node_address();
        while (dest == node_address()) // Just to avoid sending FA to itself
        {
                uniform_value = ANTNET::randomgen_.uniform(0,1);
                dest = (nsaddr_t) floor(uniform_value * num_nodes);
        }
        DEBUG_ANTNET_ANT("Destination chosen is "<< dest);
        return dest;
}

void ANTNET::addDestination(nsaddr_t dest)
        /** Add new destination to list
         */
{
        // Look for existing destination
        for (std::vector<nsaddr_t>::iterator i = destinations_.begin();
                        i != destinations_.end(); i++)
                if ((*i) == dest)
                        return;
        // Destination not found, add it to list and rtable
        DEBUG_ANTNET("New destination found: "<< dest);
        destinations_.push_back(dest);
        rtable.updateOnNewDestination(dest);
        // Initialize mean and variance for this destination,
        // although first BA will init mean value again
        ltm.mean(dest) = 0.0;
        ltm.var(dest)  = 0.0;
}
```

// EOF

A2.2-WAntNet.h

```
#ifndef ns_antnet_h
#define ns_antnet_h

#include <list>
#include <queue>
#include <map>
#include <sstream>
#include <iostream>
#include <classifier-port.h>
#include <timer-handler.h>
#include <priqueue.h>
#include <random.h>
#include <connector.h>
#include <dynalink.h>
#include <node.h>
#include <cmu-trace.h>
#include "antnet-packet.h"
#include "ndp.h"


/*****************************************************************
 *        C O N S T A N T S  &  P A R A M E T E R S
 *****************************************************************/

// Used to scale events on time
#define TIME_FACTOR               1
#define JITTER_BEGINNING_SIMULATION  1.5 * TIME_FACTOR

// ### Neighbor management ###
#define HELLO_INTERVAL         1.0 * TIME_FACTOR
// Popular choices of HELLO_INTERVAL are between 0.2 and 1s.
// Must be larger than the time difference between a node
// propagates a route request and gets the route reply back.
#define ALLOWED_HELLO_LOSS    ANTNET::ALLOWED_HELLO_LOSS  // packets
#define MaxHelloInterval      (1.25 * HELLO_INTERVAL)
#define MinHelloInterval      (0.75 * HELLO_INTERVAL)
// Wait for the neighbors to be discovered
// and then start sending forward ants
#define NEIGHBOR_DISCOVERY_PERIOD    10 * TIME_FACTOR
// Think it should be 30 ms
#define ARP_DELAY 0.01      // Fixed delay to keep ARP happy


// ### Debugging stuff ###
#define REFRESH_INTERVAL 100  // Interval to refresh debug info
#define OID               5  // Node observed when debugging
#define MAX_FA_JUMPS      10  // When to print jump stack


// Wait until some FAs have arrived at all destinations
// and rtable is build in every node
// Has to be changed in MANET test tools and traffic
// pattern generator cbrgen.tcl
//
// TRAINING_PERIOD   100 * TIME_FACTOR
```

```
// At the beginning, all neighbors are equiprobable,
// but fter the first BA arrived at destination,
// new neighbors will have less probability
#define MINIMUM_PROBABILITY_NH_ENTRY 0.01

// If prob don't sum up 1.0 exactly
// which correction can be applied
#define MAX_ALLOWED_PROB_CORRECTION  0.01

#define ERROR_BAD_PROBABILITY 1


// Packet queue element
typedef struct {
        Packet* p;
        Handler* h;
} qelement;

typedef struct {
        Packet*  p;
        float      expire;
} rescued_packet;

// Link element structure
typedef struct {
        const char* classname;
        Connector* ref;
} lelement;

struct ltstr {
  bool operator()(const char* s1, const char* s2) const {
    return strcmp(s1, s2) < 0;
  }
};


/*************************************************************************
 *          L O C A L   T R A F F I C   M O D E L
 *************************************************************************/

class travel_time
{
protected:
        nsaddr_t nexthop_;    // Address of neighbor used as next hop
        float time_;          // Time experienced towards destination

public:
        travel_time(nsaddr_t nh, float t)
        {
                nexthop_ = nh; time_ = t;
        }
        ~travel_time() {}

        inline nsaddr_t& nexthop() { return nexthop_; };
        inline float& time() { return time_; };
};

typedef list<travel_time*> traveltime_list_t;
```

```
class LocalTrafficModel
{
protected:
        // List of travel times for every destination
        map<nsaddr_t, traveltime_list_t*> timemap_;
        // Mean value of trip times
        map<nsaddr_t, float> mean_;
        // Variance value of trip times
        map<nsaddr_t, float> variance_;

        ANTNET* agent;

public:
        // Constructor
        LocalTrafficModel(ANTNET*);
        ~LocalTrafficModel();

        // Access methods
        inline float& mean(nsaddr_t d) { return mean_[d]; };
        inline float& var(nsaddr_t d) { return variance_[d]; };
        inline int timemap_size() { return timemap_.size(); };
        inline int timelist_size(nsaddr_t d) { return timemap_[d]->size(); };

        // Insert travel time from current node to dest in register
        void registerNewTravelTime(nsaddr_t, nsaddr_t, float);

        // Retrieve information
        travel_time* getBestTravelTime(nsaddr_t dest);
        travel_time* getLastTravelTime(nsaddr_t dest);
        void print(void);

        // Statistical functions
        double squash(double, int);
        void estimateMeanVariance(nsaddr_t dest, float);
};



/*************************************************************************
 *            R O U T I N G   T A B L E
 *************************************************************************/

class ANTNET;
class NextHopEntry;
typedef multimap<nsaddr_t, NextHopEntry*> antnet_rtable_t;

class AntRoutingTable
{
protected:
        bool    initialized_;   // Has some BA already arrived?
        antnet_rtable_t  rt_;   // Map of nexthop addresses and associated
                                // probability for every destination
        ANTNET*       agent;   // To access neighbors and destinations

public:
        // Constructor
        AntRoutingTable(ANTNET *a);
```

```
        ~AntRoutingTable();

        // Update on events
        void updateOnNewDestination(nsaddr_t);
        void updateOnNewNeighbor(nsaddr_t);
        void updateOnLostNeighbor(nsaddr_t);
        void updateOnBackwardAntArrival(AntNet_Data*);

        // Retrieve information
        NextHopEntry* getBestNextHopEntryForDataPacket(nsaddr_t dest);
        NextHopEntry* getBestNextHopEntryForForwardAnt(nsaddr_t dest, std::vector<nsaddr_t> nb_list);
        void print(void);

protected:
        // Scale probabilities after updating
        void normalizeProbabilitiesForDestination(nsaddr_t dest, float, nsaddr_t nh_updated);
        void scaleProbabilitiesForDestination(nsaddr_t dest);

        // Retrieve probabilities
        float getProbUsingNeighbor(Neighbor *nb, nsaddr_t dest);

        // NextHop entries management
        NextHopEntry* getNextHopEntry(nsaddr_t nh_addr, nsaddr_t dest);
        int eraseNextHopEntry(nsaddr_t nh_addr, nsaddr_t dest);
};

class NextHopEntry
{
protected:
        nsaddr_t addr_;
        float probability_;

public:
        NextHopEntry(nsaddr_t);
        ~NextHopEntry() {}

        // Access methods
        inline nsaddr_t& addr() { return addr_; };
        inline float& prob() { return probability_; };

        // Set a correct probability
        int setProbability(float);
};



/*************************************************************************
 *                      T I M E R S
 *************************************************************************/

class ForwardAntGenerationTimer : public Handler {
public:
        ForwardAntGenerationTimer(ANTNET* a) : agent_(a) {running_ = false;}
        void handle(Event*);
private:
        bool running_;
   ANTNET *agent_;
        Event  intr;
};
```

```
class StartSimulationTimer: public Handler {
public:
        StartSimulationTimer(ANTNET* a) : agent_(a) {number_of_times_called = 1;}
        void handle(Event*);
private:
        int number_of_times_called;
        ANTNET          *agent_;
        Event    intr;
};

class PollQueueTimer: public Handler {
public:
        PollQueueTimer(ANTNET* a) : agent_(a) {}
        void handle(Event*);
private:
        ANTNET          *agent_;
        Event    intr;
};

class LostNeighborTimer: public Handler {
public:
        LostNeighborTimer(ANTNET* a) {}
        void handle(Event*);
private:
        ANTNET          *agent_;
        Event    intr;
};



/*************************************************************************
 *              R O U T I N G   A G E N T
 *************************************************************************/

class ANTNET : public Agent
        /** AntNet routing agent
         */
{
        // Make some friends first
        friend class AntRoutingTable;
        friend class LocalTrafficModel;
        friend class ForwardAntGenerationTimer;
        friend class StartSimulationTimer;
        friend class PollQueueTimer;
        friend class LostNeighborTimer;
        friend class NDP;

public:
        // Constructor and TCL stuff
        ANTNET(nsaddr_t);
        ~ANTNET();

        virtual const char* module_name() const { return "ANTNET"; }
        virtual int command(int argc, const char*const* argv);

        void recv(Packet*, Handler*);           // Handle incoming packet

        static RNG randomgen_;              // Random generator
```

```
        // ### ANTNET parameters (from Di Caro & Dorigo) ###
        //
        // Simulated agent processing time (secs)
        static float SERVICE_TIME;
        // FA generation interval in secs (inverse of rate)
        static float FA_INTERVAL;
        // Factor ETA weights the number of most recent samples
        // that will really affect the average
        static float ETA;
        static u_int32_t MAX_WINDOW_SIZE;
        // Factor GAMMA is the confidence level (range between 75% to 95%)
        #define ZETA(gamma)  (1 / sqrt((1 - gamma)))
        static float CONFIDENCE_LEVEL;
        static float ZETA;
        // Coefficients C1 and C2 are used to weight the 2 terms
        // of equation (7) when calculating the reinforcement value
        static float C1;
        static float C2;
        // When choosing nexthop, weight the importance of queue
        // and rtable entries. Values are between 0.2 and 0.5
        static float ALFA;
        // Data packets are prevented from choosing links with
        // very low probability by remapping the rtable entries
        // by means of a power function
        static float REMAP_PROB_FACTOR;
        // Squash factor (a/Nk)
        // Should be double than average number of neighbors
        static float SQUASH_FACTOR_A;
        // If uniform random number is below this threshold
        // we choose uniformly to help discovering new paths
        static float CHOOSE_NEXTHOP_UNIFORMLY_THRESHOLD;
        // Maximum reinforcement that can be given to a route
        static float MAX_REINFORCEMENT;
        // Maximum number of packets waiting in queues
        static u_int32_t MAX_PACKETS_PRIOQUEUE;
        static u_int32_t MAX_PACKETS_NORMALQUEUE;
        static u_int32_t MAX_PACKETS_RESCUED;
        static u_int32_t ALLOWED_HELLO_LOSS;

protected:
        MobileNode* node_;
        inline MobileNode* node() { return node_; };
        nsaddr_t node_address_;         // Address of attached node
        inline nsaddr_t& node_address() { return node_address_; };

        // Routing table linking destination and available nexthops
        AntRoutingTable     rtable;
        // Traffic statistics are stored in an LTM obejct
        LocalTrafficModel   ltm;

        // Queue management
        queue<qelement> PrioInQueue_;     // Queue of backward ants (BA) to be processed
        queue<qelement> NormalInQueue_;   // Queue of forward ants (FA) and data packets
        queue<qelement> PrioOutQueue_;    // Queue of BA to be forwarded
        queue<qelement> NormalOutQueue_;  // Queue of FA and data packets to be forwarded
        void poll_queue();                // Process queued packets if possible
        void enqueue(qelement);           // Put ingoing packet in the right queue
        inline void sendPrioPacket(Packet* p, Handler* h) {
```

```
        PrioOutQueue_.push((qelement){p,h});
}
inline void sendNormalPacket(Packet* p, Handler* h) {
        NormalOutQueue_.push((qelement){p,h});
}

// Routing decisions
nsaddr_t nextHopForDataPacket(nsaddr_t);
nsaddr_t nextHopForForwardAnt(AntNet_Data*);
nsaddr_t destinationForForwardAnt();

// Processing packets
void processPacket(Packet*, Handler*);
void forwardDataPacket(Packet*, Handler*);
void processControlPacket(Packet*, Handler*);

// Neighbor Discovery Protocol controller
NDP ndpc;

// Lost neighbors management
void salvagePacketsFromIFQ(nsaddr_t);     // Rescue packets from IFQ
void storePacketLostNeighbor(Packet*);    // which have lost nexthop
queue<rescued_packet> LostNeighborBuffer_; // and store them in a buffer
void rerouteForwardAnt(Packet*, Handler*);

// Destinations management
std::vector<nsaddr_t> destinations_;
void addDestination(nsaddr_t dest);

// Timers
StartSimulationTimer        SS_timer;
ForwardAntGenerationTimer   FA_timer;
PollQueueTimer                  PQ_timer;
LostNeighborTimer               LN_timer;

// Forward Ant generation
u_int32_t sequence_counter_;
void generateForwardAnt();

// A pointer to the network interface queue that sits
// between "classifier" and " "link layer"
PriQueue* ifqueue_;
Mac* mac_;

// Log routing table info
Trace* logtarget_;

// For passing packets up to agents
PortClassifier* dmux_;

// Debugging stuff
int all_pkt_counter_;
int data_pkt_counter_;
int recv_pkt_counter_;
int forw_pkt_counter_;
int salv_pkt_counter_;
int ndp_pkt_counter_;
int fa_pkt_counter_;
int ba_pkt_counter_;
```

```
        int max_ifq_len;
};

#endif
```

A2.3-WAntNet-packet.cc

```
#include "antnet.h"
#include "antnet-packet.h"

#define DEBUG_ANTNET_PKT(msg)
//#define DEBUG_ANTNET_PKT(msg) cout <<"Ant["<< visited_nodes().front() <<"->"<< dest() <<"] #
"<< __FUNCTION__ <<" # "<< msg <<"\n"

// Make instances of static variables
int hdr_antnet::offset_;

// TCL stuff
class AntNetHeaderClass : public PacketHeaderClass {
public:
        AntNetHeaderClass() : PacketHeaderClass("PacketHeader/AntNet", sizeof(hdr_antnet)) {
                bind_offset(&hdr_antnet::offset_);
        }
} class_antnethdr;


/*************************************************************
 *            N E X T H O P  J U M P S
 *************************************************************/

void AntNet_Data::insertJump(nsaddr_t nh_addr)
        /** Insert jump to given nexthop into the stack
         */
{
        //nexthop_jump *jump = new nexthop_jump(nh_addr);
        nexthop_jump *jump = new nexthop_jump;
        jump->nexthop_addr = nh_addr;
        jump->departure_time = CURRENT_TIME;
        jumps_->push_back(jump);
}

void AntNet_Data::removeLastJump()
        /** Remove last jump from the stack
         */
{
        delete jumps_->back();   // delete object
        jumps_->pop_back();      // remove pointer
}

string AntNet_Data::printJumpStack()
{
        stringstream o;
        o << "[";
        for (list<nexthop_jump*>::iterator i = jumps_->begin();
                        i != jumps_->end(); i++)
                o << (*i)->nexthop_addr << ":" << (*i)->departure_time << " ";
        o << "]";
        return o.str();
}

/*************************************************************
 *            T R A V E L  T I M E
```

```
 ************************************************************/

float AntNet_Data::getTravelTime()
        /** Return time spent by forward ant since it has been
         *  forwarded by current node to destination and until
         *  it has returned to the current node.
         *
         *  The idea is to gather all possible travel times.
         *  We do that storing the departure time at each visited
         *  node in the agent itself. Then we can calculate the
         *  travel time to destination easily.
         *
         *  LET OP!
         *
         *  Please note that after calling this method, one
         *  should remove nexthop_jump from the ant's stack
         */
{
        // Has the ant really travelled?
        assert(jumps_->size() > 0);

        // We assume that desired nexthop_jump is the last one
        // (oneway travel time or round trip?)
        //float travel_time = CURRENT_TIME - jumps_->back()->departure_time();  // round trip
        float travel_time = arrivaltime() - jumps_->back()->departure_time;    // one way

        // Verify that travel time is not negative
        assert(travel_time > 0);
        return travel_time;
}

nsaddr_t AntNet_Data::getNexthop()
        /** Return nexthop used by forward ant to arrive at destination
         */
{
        // We assume that desired nexthop_jump is the last one
        return jumps_->back()->nexthop_addr;
}

float AntNet_Data::getTotalTravelTime()
        /** Get travel time from source to destination
         */
{
        //return CURRENT_TIME - birthtime();   // round trip
        return arrivaltime() - birthtime();  // one way
}

int AntNet_Data::longCycleDetected(nsaddr_t nh_addr)
        /** If a cycle is detected, that is, if an ant
         *  is forced to return to an already visited node,
         *  the cycle's nodes are opped 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, the ant is destroyed.
         */
{
        DEBUG_ANTNET_PKT("nexthop_addr="<< nh_addr);
        for (std::list<nsaddr_t>::iterator i = visited_nodes().begin();
                        i !=    visited_nodes().end(); i++)
```

```
                if ((*i) == nh_addr) // Already visited node found
                {
                        // Cycle lasted longer than ant's lifetime
                        // before entering the cycle?
                        u_int32_t cycle_size = 0;
                        for (; i != visited_nodes().end(); i++)
                                cycle_size++;
                        DEBUG_ANTNET_PKT("cycle.size()="<< cycle_size <<"
visited_nodes.size()="<< visited_nodes().size());

                        if (cycle_size > visited_nodes().size()/2)
                                return 1; // Long cycle found

                        // Pop back cycle's nodes and inserted jumps
                        while (visited_nodes().back() != nh_addr)
                                removeCycleNodeFromStack();
                        // Remove node starting the cycle
                        removeCycleNodeFromStack();

                        return 0; // Non-long cycle repaired
                }

        // Not visited yet, so no cycle detected
        DEBUG_ANTNET_PKT("Ok, continue");
        return 0;
}

void AntNet_Data::removeCycleNodeFromStack()
{
        DEBUG_ANTNET_PKT("Cycle detected! Prev stack=["<< printJumpStack() <<"]");
        DEBUG_ANTNET_PKT("Removing cycle node "<< visited_nodes().back() << \
                " and jump ["<< jumps_->back()->nexthop_addr <<":"<< jumps_->back()->departure_time
<<"]");
        visited_nodes().pop_back();
        removeLastJump();
}
```

A2.4-WAntNet-packet.h

```
#ifndef ns_antnet_packet_h
#define ns_antnet_packet_h

#include <list>
#include <map>
#include <string>

#include "packet.h"
#include "agent.h"
#include "node.h"
#include "tcl.h"

#define HDR_ANTNET(p) ((struct hdr_antnet*)hdr_antnet::access(p))

#define NDP_HELLO_MSG    0x01
#define FORWARD_ANT      0x02
#define BACKWARD_ANT     0x03

#define CURRENT_TIME     Scheduler::instance().clock()


struct hdr_antnet
{
        u_int8_t            ah_type;    // FA, BA or Hello
        u_int32_t           seq_no;     // Sequence number to keep track of control packets

        static int offset_;  // offset for this header
        inline static int& offset() { return offset_; }
        inline static hdr_antnet* access(Packet* p) {
                return (hdr_antnet*) p->access(offset_);
        }
        string typeDescription()
        {
                switch (ah_type)
                {
                        case NDP_HELLO_MSG:
                                return "NDP";
                        case FORWARD_ANT:
                                return "FA_";
                        case BACKWARD_ANT:
                                return "BA_";
                        default:
                                return "___";
                }
        }
};

struct nexthop_jump
{
        nsaddr_t nexthop_addr;
        float departure_time;
};

class AntNet_Data : public AppData
        /** Data stored in AntNet packets
```

```
        */
{
friend class ANTNET;

protected:
        nsaddr_t dest_;             // Destination (need it for backtracking)
        float birthtime_;           // Time of "birth"
        float arrival_time_;        // Arrival at destination
        list<nsaddr_t>* visited_nodes_;   // List of already visited nodes
        list<nexthop_jump*>* jumps_;      // List of departure times to nexthop

public:
        AntNet_Data(nsaddr_t dst) : AppData(ANTNET_DATA)
        {
                dest_ = dst;
                birthtime_ = CURRENT_TIME;
                arrival_time_ = 0;
                visited_nodes_ = new list<nsaddr_t>();
                jumps_ = new list<nexthop_jump*>();
        }

        AntNet_Data(AntNet_Data& d) : AppData(d) {
                dest_ = d.dest_;
                birthtime_ = d.birthtime_;
                arrival_time_ = d.arrival_time_;
                visited_nodes_ = new list<nsaddr_t>(d.visited_nodes());
                jumps_ = new list<nexthop_jump*>(d.jumps());
        }

        virtual ~AntNet_Data() {
                delete visited_nodes_;
                delete jumps_;
        }

        // Access methods
        inline nsaddr_t& dest() { return dest_; };
        inline list<nsaddr_t>& visited_nodes() { return *visited_nodes_; };
        inline list<nexthop_jump*>& jumps() { return *jumps_; };
        inline float& birthtime() { return birthtime_; };
        inline float& arrivaltime() { return arrival_time_; };
        inline int size() {
                int sz = 0;
                sz = sizeof(nsaddr_t)     // Destination
                  + sizeof(float)         // Birthtime
                  + sizeof(float)         // Arrival time
                  + visited_nodes_->size() * sizeof(nsaddr_t)
                  + jumps_->size() * sizeof(nexthop_jump*);
                assert(sz >= 0);
                return sz;
        }

        virtual AppData* copy() { return new AntNet_Data(*this); }

        void insertJump(nsaddr_t);    // Insert jump to nexthop into stack
        void removeLastJump();        // Remove last jump from stack
        string printJumpStack();

        float getTravelTime();        // Get travel time from current node
                                      // to destination (one way)
```

```
        nsaddr_t getNexthop();        // Get nexthop used
        float getTotalTravelTime();   // Get travel time from source to destination

        int longCycleDetected(nsaddr_t); // Checks for cycles in ant travel
        void removeCycleNodeFromStack(); // Removes last visited_node and jump
};

#endif // ns_antnet_packet_h
```

A2.5-WAntNet-rtable.cc

```
#include "antnet.h"

#define DEBUG_RTABLE(msg)
/*
#define DEBUG_RTABLE(msg)  if (agent->node_address() == OID) \
        cout << agent->node_address() <<":"<< CURRENT_TIME <<" # "<< __FUNCTION__ <<" # "<<
msg <<"\n"
*/


// Constructor
AntRoutingTable::AntRoutingTable(ANTNET *a)
        // Prepare all data structures
{
        initialized_ = false;
        rt_.clear();
        agent = a;
}

AntRoutingTable::~AntRoutingTable()
        // Clean all related structures
{
        rt_.clear();
}


/*************************************************************
 *      R O U T I N G   T A B L E   U P D A T E
 *************************************************************/

void AntRoutingTable::updateOnNewDestination(nsaddr_t dest)
        /** When a new destination is discovered
         *  update rtable inserting a new NextHopEntry
         *  for each neighbor discovered so far
         */
{
        // Insert a new entry in rtable for every known neighbor
        NextHopEntry* nh_entry = NULL;
        Neighbor* nb = agent->ndpc.nbhead.lh_first;
        for (; nb; nb = nb->nb_link.le_next)
        {
                nh_entry = getNextHopEntry(nb->nb_addr, dest);
                if (! nh_entry) // Not yet inserted in rtable
                {
                        nh_entry = new NextHopEntry(nb->nb_addr);
                        rt_.insert(make_pair(dest, nh_entry));

                        //DEBUG_RTABLE("Inserted new entry [nh_addr="<< nb->nb_addr <<",
dest="<< dest <<"]");
                }
        }
        scaleProbabilitiesForDestination(dest);
}

void AntRoutingTable::updateOnNewNeighbor(nsaddr_t nb_addr)
```

```
{
	// Look for every destination
	for (std::vector<nsaddr_t>::iterator i = agent->destinations_.begin();
			i != agent->destinations_.end(); i++)
	{
		// Create new entry with minimum probability
		// and insert it into the routing table
		NextHopEntry *nh_entry = new NextHopEntry(nb_addr);
		rt_.insert(make_pair((*i), nh_entry));
		scaleProbabilitiesForDestination(*i);
	}
}

void AntRoutingTable::updateOnLostNeighbor(nsaddr_t nb_addr)
{
	// Look for every destination
	for (std::vector<nsaddr_t>::iterator i = agent->destinations_.begin();
			i != agent->destinations_.end(); i++)
	{
		// Search entry and get pointer
		NextHopEntry *lost_entry = getNextHopEntry(nb_addr, *i);

		// Delete it from routing table
		eraseNextHopEntry(nb_addr, *i);
		scaleProbabilitiesForDestination(*i);

		// Free memory
		delete lost_entry;
	}
}

void AntRoutingTable::updateOnBackwardAntArrival(AntNet_Data* BA)
	/** Register new travel time and update routing table
	 *  with new calculated probabilities
	 */
{
	// Update local traffic model
	//
	// Store time spent in travelling from current node
	// to destination and which nexthop was used
	float oneway_time = BA->getTravelTime();
	nsaddr_t dest     = BA->dest();
	nsaddr_t nh_addr  = BA->getNexthop();
	agent->ltm.registerNewTravelTime(nh_addr, dest, oneway_time);

	// When first BA has arrived, nexthops are not equiprobable
	if (!initialized_)
		initialized_ = true;

	// Retrieve reference to NextHopEntry
	NextHopEntry* nh_entry = getNextHopEntry(nh_addr, dest);
	if (nh_entry == NULL) {
		// If no nh_entry exists yet, that's because we didn't
		// know we have a neighbor. So we should create a nh_entry
		// in RTABLE and register the new neighbor in NDP
		DEBUG_RTABLE("Ooops! BA received before knowing we have neighbors!");
		DEBUG_RTABLE("*** Adding nexthop to rtable and neighbor list ***");
		agent->ndpc.nb_insert(nh_addr);
		updateOnNewNeighbor(nh_addr);
```

```
                nh_entry = getNextHopEntry(nh_addr, dest);
        }

        // Update probability using a reinforcement value we get from
        // local traffic model and trip time from backward ant (BA)
        float old_prob = nh_entry->prob();

        // Equation (7) : calculate reinforcement
        float best_time = agent->ltm.getBestTravelTime(dest)->time();
        float limit_sup = agent->ltm.mean(dest) +
                ANTNET::ZETA * (sqrt(agent->ltm.var(dest) / agent->ltm.timelist_size(dest)));

        assert((ANTNET::C1 + ANTNET::C2) == 1.0);

        double r = ANTNET::C1 * (best_time / oneway_time);
        double div = ((limit_sup - best_time) + (oneway_time - best_time));
        if (div != 0) // check dividebyzero exception
                r += ANTNET::C2 * ((limit_sup - best_time) / div);

        // Equation (8) : squash reinforcement value
        int nb_num = agent->ndpc.nb_size();
        double reinforcement = agent->ltm.squash(r,nb_num);

        // Reinforcement should not be 1,
        // as it will reward neighbor too much
        if (reinforcement == 1)
                reinforcement = ANTNET::MAX_REINFORCEMENT;

        // Increase probability of this entry
        float new_prob = (old_prob + (float)reinforcement * (1.0 - old_prob));
        nh_entry->setProbability(new_prob);

        DEBUG_RTABLE("Updated prob for [nh="<< nh_addr <<" dest="<< dest \
                        <<"] [old="<< old_prob <<" new="<< new_prob <<"]");

        // Decrease rest of probabilities normalizing them
        normalizeProbabilitiesForDestination(dest, reinforcement, nh_entry->addr());

/*
        double now = CURRENT_TIME;
        int nowint = (int) floor(now);
        int frequency = REFRESH_INTERVAL;
        if ((agent->node_address() == OID) && ((nowint % frequency)==0))
                print();
*/
}


void AntRoutingTable::normalizeProbabilitiesForDestination(nsaddr_t dest, float r, nsaddr_t nh_updated)
{
        NextHopEntry *nh_entry = NULL;
        float sum_prob = 0.0, old_prob, new_prob;

        Neighbor* nb = agent->ndpc.nbhead.lh_first;
        for (; nb; nb = nb->nb_link.le_next)
        {
                if (nb->nb_addr != nh_updated)
                {
                        nh_entry = getNextHopEntry(nb->nb_addr, dest);
```

```
                        old_prob = nh_entry->prob();
                        new_prob = old_prob - ( r * old_prob );
                        nh_entry->setProbability(new_prob);
                        sum_prob += new_prob;
                }
        }
        DEBUG_RTABLE("Sum of normalized prob for [dest="<< dest <<"] is [p="<< sum_prob <<",
r="<< r <<"]");
}


void AntRoutingTable::scaleProbabilitiesForDestination(nsaddr_t dest)
        /** Sum of route probabilities for a given destination has to
         *  be equal to 1. Thus, we need to scale all route prob. if
         *  we add or delete a route.
         */
{
        float total = 0.0;      // Sum of all route probabilities
        float new_prob;         // New probabiliy to set
        float factor = 0.0;     // Rescaling factor

        // Sum unscaled probabilities to get rescaling factor
        Neighbor* nb = agent->ndpc.nbhead.lh_first;
        for (; nb; nb = nb->nb_link.le_next)
        {
                if (! initialized_)
                        factor += 1; // All neighbors have equal prob
                else
                        factor += getProbUsingNeighbor(nb, dest);
        }
        if (factor == 1.0 || factor == 0.0) // No need to scale up
                return;

        // Scale up every neighbor route probability
        NextHopEntry* nh_entry = NULL;
        nb = agent->ndpc.nbhead.lh_first;
        for     (; nb; nb = nb->nb_link.le_next)
        {
                nh_entry = getNextHopEntry(nb->nb_addr, dest);
                if (! initialized_) // Equal prob
                        new_prob = 1 / factor;
                else
                        new_prob = nh_entry->prob() / factor;
                nh_entry->setProbability(new_prob);
                total += new_prob;

        //      if (dest == OID)
        //              DEBUG_RTABLE(dest << " using NH " << nb->nb_addr << " is " << new_prob);
        }

        // Obtain error between actual value and theoretical one
        // and correct values if possible
        float diff = 1.0 - total;
        if ((diff > 0.0) && (diff < MAX_ALLOWED_PROB_CORRECTION)) //positiu
        {
        /*      DEBUG_RTABLE("Correcting [nh="<< nh_entry->addr() <<" dest="<< \
                                dest <<" prob="<< new_prob <<"] adding "<< diff); */
                nh_entry->setProbability(new_prob + diff);
                total = 1.0;
```

```
        }
        else if ((diff < 0.0) && (diff > -MAX_ALLOWED_PROB_CORRECTION)) //negatiu
        {
        /*      DEBUG_RTABLE("Correcting [nh="<< nh_entry->addr() <<" dest="<< \
                                dest <<" prob="<< new_prob <<"] adding "<< diff); */
                nh_entry->setProbability(new_prob + diff);
                total = 1.0;
        }

        // All new probabilities should sum up 1.0
        assert(total == 1.0);
}


/*************************************************************
 *      R E T R I E V E   P R O B A B I L I T I E S
 *************************************************************/

float AntRoutingTable::getProbUsingNeighbor(Neighbor *nb, nsaddr_t dest)
{
        NextHopEntry *nh_entry = getNextHopEntry(nb->nb_addr, dest);
        return nh_entry->prob();
}

NextHopEntry* AntRoutingTable::getBestNextHopEntryForDataPacket(
                nsaddr_t dest)
        /** Return the NextHopEntry with best probability for given
         *  destination. Used only when forwarding DATA PACKETS.
         */
{
        NextHopEntry* nh_entry = NULL;
        float norm_factor = 0.0;
        float limit_inf   = 0.0;
        float limit_sup   = 0.0;

        // Set uniform value using normalizing factor
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
        {
                if ((*i).first == dest)
                        norm_factor += pow((*i).second->prob(), ANTNET::REMAP_PROB_FACTOR);
        }
        float uniform_value = ANTNET::randomgen_.uniform(0,norm_factor);

        // Select best probability for given destination
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
                if ((*i).first == dest)
                {
                        limit_inf = limit_sup;
                        limit_sup += pow((*i).second->prob(), ANTNET::REMAP_PROB_FACTOR);
                        if ((limit_inf < uniform_value) && (uniform_value < limit_sup))
                        {
                                nh_entry = (*i).second;
                                break;
                        }
                }
        DEBUG_RTABLE("Best entry for [dest="<< dest <<"] is [nh="<< nh_entry->addr() \
                        <<" prob="<< nh_entry->prob() <<" uniform="<< uniform_value <<"]");
        return nh_entry;
}
```

```
NextHopEntry* AntRoutingTable::getBestNextHopEntryForForwardAnt(
                nsaddr_t dest, std::vector<nsaddr_t> nb_list)
        /** Return the NextHopEntry with best probability.
         *  Used when routing FORWARD ANTS. Only look at
         *  unvisited neighbors in nb_list.
         */
{
        NextHopEntry* nh_entry = NULL;
        float norm_factor = 0.0;
        float limit_inf   = 0.0;
        float limit_sup   = 0.0;

        // Set uniform value using normalizing factor, as we only select
        // probabilities for dest and nexthop from nb_list
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
                if (((*i).first == dest) && (find(nb_list.begin(),
                                nb_list.end(), (*i).second->addr()) != nb_list.end()))
                {
                        norm_factor += (*i).second->prob();
                }
        float uniform_value = ANTNET::randomgen_.uniform(0,norm_factor);

        // Look if current value matches
        // previously generated uniform space
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
                if (((*i).first == dest) && (find(nb_list.begin(),
                                nb_list.end(), (*i).second->addr()) != nb_list.end()))
                {
                        limit_inf = limit_sup;
                        limit_sup += (*i).second->prob();
                        if ((limit_inf <= uniform_value) && (uniform_value <= limit_sup))
                        {
                                nh_entry = (*i).second;
                                break;
                        }
                }
        if (nh_entry == NULL)
                printf("getBestEntryFA: norm=[%f] inf=[%f] sup=[%f] unif=[%f]\n",\
                                norm_factor,limit_inf,limit_sup,uniform_value);
        DEBUG_RTABLE("Best entry for [dest="<< dest <<"] is [nh="<< nh_entry->addr() \
                        <<" prob="<< nh_entry->prob() <<" uniform="<< uniform_value <<"]");
        return nh_entry;
}

void AntRoutingTable::print()
        /** Just for debugging purposes
         */
{
        nsaddr_t dest = -1;
        float prob    = 0.0;
        printf("*********************************\n");
        printf("*      RtTable @ node %2d      *\n", agent->node_address());
        printf("*********************************\n");
        printf(" Dest | [Neighbor: probability] [...]\n");

        for (antnet_rtable_t::iterator i = rt_.begin();  i != rt_.end(); i++)
        {
                if ((*i).first != dest) {
```

```
                dest = (*i).first;
                printf("\n %2d:  | ",dest);
            }
            prob = (*i).second->prob();
            if (prob == 0.0)
                printf("[%d:     ] ", (*i).second->addr());
            else if (prob == 1.0)
                printf("[%d: -1- ] ", (*i).second->addr());
            else
                printf("[%d:%1.3f] ", (*i).second->addr(), prob);
        }
        printf("\n*******************************\n");
}


/*************************************************************
 *    N E X T H O P   E N T R I E S   M A N A G E M E N T
 *************************************************************/

NextHopEntry::NextHopEntry(nsaddr_t nh_addr)
{
        addr_ = nh_addr;
        probability_ = MINIMUM_PROBABILITY_NH_ENTRY;
}

int NextHopEntry::setProbability(float prob)
{
        if (prob < 0.0 || prob > 1.0)
                return ERROR_BAD_PROBABILITY;
        probability_ = prob;
        return 0;
}

NextHopEntry* AntRoutingTable::getNextHopEntry(nsaddr_t nh_addr, nsaddr_t dest)
        /** Retrieve pointer to NextHopEntry object in routing table associated
         *  with destination <dest> and using neighbor <ngh>
         */
{
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
        {
                if ((*i).first == dest)             // select entries for given destination
                        if ((*i).second->addr() == nh_addr)   // select neighbor entry
                                return (*i).second;
        }
        // NextHopEntry not found
        return NULL;
}

int AntRoutingTable::eraseNextHopEntry(nsaddr_t nh_addr, nsaddr_t dest)
        // Erase nexthop from routing table
        // Actual nexthop object destruction is done outside
{
        for (antnet_rtable_t::iterator i = rt_.begin(); i != rt_.end(); i++)
        {
                if ((*i).first == dest)             // select entries for given destination
                        if ((*i).second->addr() == nh_addr)   // select neighbor entry
                        {
                                rt_.erase(i);
                                return true;
```

```
                                }
        }
        return false;
}

// EOF
```

A2.6-WAntnet-ltm.cc

```
#include "antnet.h"
#include <cfloat>

#define DEBUG_LTM(msg)
/*
#define DEBUG_LTM(msg) if (agent->node_address() == OID) \
        cout << agent->node_address() <<":"<< CURRENT_TIME <<" # "<< __FUNCTION__ <<" # "<<
msg <<"\n"
*/


// Constructor
LocalTrafficModel::LocalTrafficModel(ANTNET *a)
{
        timemap_.clear();
        mean_.clear();
        variance_.clear();
        agent = a;
}
LocalTrafficModel::~LocalTrafficModel()
{
        timemap_.clear();
        mean_.clear();
        variance_.clear();
}



/*************************************************************
 *   U P D A T E   L O C A L   T R A F F I C   M O D E L
 *************************************************************/

void LocalTrafficModel::estimateMeanVariance(nsaddr_t dest, float trip_time)
        /** Update mean and variance values using exponential strategy
         */
{
        // Update mean value
        float old_mean = mean(dest);
        mean(dest) = old_mean + ANTNET::ETA * (trip_time - old_mean);
        // Update variance value
        float old_var = var(dest);
        float temp = trip_time - mean(dest);
        var(dest) = old_var + ANTNET::ETA * (temp*temp - old_var);

        // For debugging purposes ...
        //double now = CURRENT_TIME;
        //int nowint = (int) floor(now);
        //int frequency = REFRESH_INTERVAL;

        //if ((nowint % frequency) == 0)
                DEBUG_LTM("Mean[old="<< old_mean <<" new="<< mean(dest) \
                        <<"] Var[old="<< old_var <<" new="<< var(dest) \
                        <<"] [samples="<< timelist_size(dest) <<" max="<<
ANTNET::MAX_WINDOW_SIZE <<"]");
        // End debug
```

```
}

double LocalTrafficModel::squash(double x, int nb_num)
        /** Squashing the reinforcement values (x), 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.
         *
         *  squash(r) = s(r) / s(1) where s(x) = 1 / (1 + exp(a / x*nb_num)
         */
{
        // Coefficient A determines a parametric dependence
        // of the squashed reinforcement value on number of neighbors
        double exp_r = ANTNET::SQUASH_FACTOR_A / (x * nb_num);
        double exp_1 = ANTNET::SQUASH_FACTOR_A / nb_num;

        if (exp_r > log(DBL_MAX)) // to avoid SIGFPE
                return (1.0 + exp(exp_1)) / (1.0 + DBL_MAX);
        else
                return (1.0 + exp(exp_1)) / (1.0 + exp(exp_r));
}

void LocalTrafficModel::registerNewTravelTime(nsaddr_t nh_addr_used,
                nsaddr_t dest, float time)
        /** Store time experienced by backward ant in a register
         *
         *  The TravelTime Register should be transparent to user,
         *  only used by the Routing Table object. The idea is to
         *  store all the travel times experienced by backward ants
         *  into a vector in order to choose the best one and update
         *  the routing table.
         */
{
        // Prepare time entry
        travel_time *time_entry = new travel_time(nh_addr_used, time);

        // Search for destination in register
        traveltime_list_t *ttlist = NULL;
        for (map<nsaddr_t, traveltime_list_t*>::iterator i = timemap_.begin();
                        i != timemap_.end(); i++)
                if ((*i).first == dest)
                        // Destination has already a travel_time list, let's use it
                        ttlist = (*i).second;

        // Destination not found?
        if (ttlist == NULL) {
                // Create a new travel_time list and attach it to register
                ttlist = new list<travel_time*>();
                timemap_.insert(make_pair(dest, ttlist));
        }

        // Insert time_travel in register
        ttlist->push_back(time_entry);

        // If MAX_WINDOW_SIZE reached, remove first entry
        if (ttlist->size() > ANTNET::MAX_WINDOW_SIZE)
        {
                delete ttlist->front();
                ttlist->pop_front();
```

```
	}

	DEBUG_LTM(" Registered " << time << "s using hop " << nh_addr_used << " to dest " << dest);

	// Init mean value if it is first BA
	if (mean(dest) == 0.0) mean(dest) = time;

	// Update statistical data
	estimateMeanVariance(dest, time);
}


/*************************************************************
 *	G E T  S T A T I S T I C S
 *************************************************************/

travel_time* LocalTrafficModel::getBestTravelTime(nsaddr_t dest)
{
	travel_time* best_ptr = NULL;
	float best_time = INFINITY;

	for (map<nsaddr_t, traveltime_list_t*>::iterator i = timemap_.begin();
			i != timemap_.end(); i++)
	{
		if ((*i).first == dest)
			// select only times related to given destination
		{
			traveltime_list_t *ttlist = (*i).second;
			for (list<travel_time*>::iterator t = ttlist->begin();
					t != ttlist->end(); t++)
			{
				if ((*t)->time() < best_time) {
					// Update best time and ptr to return
					best_time = (*t)->time();
					best_ptr = (*t);
				}
			}
		}
	}
	//if (best_ptr == NULL)
	//	best_ptr = new travel_time(-1, 0.0);
	return best_ptr;
}

travel_time* LocalTrafficModel::getLastTravelTime(nsaddr_t dest)
{
	// Select destination
	for (map<nsaddr_t, traveltime_list_t*>::iterator i = timemap_.begin();
			i != timemap_.end(); i++)
		if ((*i).first == dest)
			// return last entry of attached list
			return (*i).second->back();
	// travel_time not found
	return NULL;
}


/*************************************************************
 *	D E B U G
```

```
 ***********************************************************/

void LocalTrafficModel::print()
        /** Just for debugging purposes
         */
{
        printf("*********************************************************\n");
        printf("*         LocalTrafficModel @ node %2d              *\n", agent->node_address());
        printf("*********************************************************\n");
        printf(" Dest | Nexthop | Travel time |    Mean    | Variance \n");
        for (map<nsaddr_t, traveltime_list_t*>::iterator i = timemap_.begin();
                i != timemap_.end(); i++)
        {
                traveltime_list_t *tl = (*i).second;
                for (traveltime_list_t::iterator t = tl->begin(); t != tl->end(); t++)
                        printf(" %2d   | %2d      | %f   | %f   | %f \n", (*i).first,
                                (*t)->nexthop(), (*t)->time(),
                                mean((*i).first), var((*i).first));
                printf("*********************************************************\n");
        }
        printf("*********************************************************\n");

}

// EOF
```

A2.7-Ndp.cc

```
/*
 *  Neighbor Discovery Protocol (NDP) for Ant Routing protocol
 */
#include "antnet.h"

// Debug output macros
#define DEBUG_NDP 0
#define DEBUG_NDP_MSG(msg)
//#define DEBUG_NDP_MSG(msg) cout << agent_->node_address() <<":"<< CURRENT_TIME <<" # "<<
__FUNCTION__ <<" # "<< msg << "\n"


/*************************************************************
 *          Timers (Hello, Neighbor cache)
 *************************************************************/

NDPHelloTimer::NDPHelloTimer(NDP* a)
{
        ndp_m = a;
}
NDPNeighborTimer::NDPNeighborTimer(NDP* a)
{
        ndp_m = a;
}

void NDPHelloTimer::handle(Event*)
{
        ndp_m->sendHello();
        // The interval at which hello packets are sent is chosen
        // randomly between MinHelloInterval and MaxHelloInterval
        double interval = MinHelloInterval +
                ((MaxHelloInterval - MinHelloInterval) * Random::uniform());
        assert(interval >= 0);
        Scheduler::instance().schedule(this, &intr, interval);
}

void NDPNeighborTimer::handle(Event*)
{
        ndp_m->nb_purge();
        // This is to purge the neighbors who didn't say hello soon enough
        Scheduler::instance().schedule(this, &intr, HELLO_INTERVAL);
}


/*************************************************************
 *             NDP constructor
 *************************************************************/

NDP::NDP(ANTNET* a) : htimer(this), ntimer(this)
{
        // Steal variables from ANTNET
        LIST_INIT(&nbhead);
        agent_ = a;
}
```

```
/*************************************************************
 *              Neighbor management methods
 *************************************************************/

void NDP::nb_insert(nsaddr_t id)
{
        Neighbor *nb = new Neighbor(id);
        assert(nb);

        // How many "hello packets" could be sent before
        // deciding that neihgbor is no longer available
        nb->nb_expire = CURRENT_TIME +
                (1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
        LIST_INSERT_HEAD(&nbhead, nb, nb_link);
}

Neighbor* NDP::nb_lookup(nsaddr_t id)
        /** Looks for a neighbor with address 'id' in the
         *  neighbor cache.
         */
{
        // Returns NULL at the very beginning,
        // when no neighbor has been discovered yet
        Neighbor *nb = nbhead.lh_first;

        for(; nb; nb = nb->nb_link.le_next) {
                if (nb->nb_addr == id)
                        break;
        }
        return nb;
}

void NDP::nb_delete(nsaddr_t id)
        /** Called when we receive *explicit* notification
         *  that a neighbor is no longer available, or when
         *  the neighbor in question failed to say hello in time.
         */
{
        Neighbor *nb = nbhead.lh_first;

        for (; nb; nb = nb->nb_link.le_next)
        {
                if (nb->nb_addr == id)
                {
                        DEBUG_NDP_MSG("Node " << id << " is not longer a neighbor.");

                        LIST_REMOVE(nb, nb_link);
                        // The NDP, having discovered a missing node, calls
                        // ANTNET protocol, to update routing table
                        agent_->rtable.updateOnLostNeighbor(nb->nb_addr);
                        delete nb;
                        break;
                }
        }

        // Try to salvage the packets that were waiting in
        // the ifqueue to be sent towards this neighbor
        agent_->salvagePacketsFromIFQ(id);
```

```
}

void NDP::nb_purge()
        /**
         * Purges all timed-out Neighbor entries (runs every
         * HELLO_INTERVAL * 1.5 seconds)
         */
{
        Neighbor *nb = nbhead.lh_first;
        Neighbor *nbn;
        double now = CURRENT_TIME;
        int nowint = (int) floor(now);
        int frequency = REFRESH_INTERVAL;

        for (; nb; nb = nbn)
        {
                nbn = nb->nb_link.le_next;
                if (nb->nb_expire <= now)
                        nb_delete(nb->nb_addr);
                if (DEBUG_NDP)
                {
                        if ((agent_->node_address() == OID) && ((nowint % frequency)==0))
                                // Frequency at which the list of neighbors
                                // is written on the file
                        {
                                nb_print();
                        }
                } // if DEBUG_NDP
        }
}

void NDP::nb_print()
{
        Neighbor* cnb = nbhead.lh_first;
        fprintf(stdout, "%d:%f current neighbors [ ", agent_->node_address(), CURRENT_TIME);
        for (; cnb; cnb = cnb->nb_link.le_next)
                fprintf(stdout, "%d ", cnb->nb_addr);
        fprintf(stdout, "] \n");
}

int NDP::nb_size()
{
        int counter = 0;

        Neighbor *nb = nbhead.lh_first;
        for (; nb; nb = nb->nb_link.le_next)
                counter++;
        return counter;
}


/*************************************************************
 *            Packet reception routines
 *************************************************************/

void NDP::recv(Packet *p)
{
        struct hdr_cmn *ch = HDR_CMN(p);
```

```
	if (ch->ptype() == PT_ANTNET)
	{
		Neighbor *nb;

		//DEBUG_NDP_MSG("Looking for neighbors...");

		nb = nb_lookup(ch->prev_hop_);
		// Check if the neighbor is already on the neighbor list
		if (nb == 0)
		{
			nb_insert(ch->prev_hop_);
			if (DEBUG_NDP && (agent_->node_address() == OID))
			{
				DEBUG_NDP_MSG("Node " << ch->prev_hop_ << " registered as a
neighbor.");
				nb_print();
			}
			// Add new neighbor in routing table
			nb = nb_lookup(ch->prev_hop_);
			agent_->rtable.updateOnNewNeighbor(nb->nb_addr);
		}

		// Consider nb as a neighbor for another period of time
		nb->nb_expire = CURRENT_TIME +
			(1.5 * ALLOWED_HELLO_LOSS * HELLO_INTERVAL);
	}
	else
	{
		fprintf(stderr,"-- a non-ANTNET packet received by NDP::recv, dropped \n");
		Packet::free(p);
		exit(1);
	}
}

void NDP::sendHello()
{
	Packet *p = Packet::alloc();
	struct hdr_cmn *ch = HDR_CMN(p);
	struct hdr_ip *ih = HDR_IP(p);
	struct hdr_antnet *ah = HDR_ANTNET(p);

	ch->ptype() = PT_ANTNET;
	ch->size() = IP_HDR_LEN + sizeof(*ah);
	ch->iface() = -2;
	ch->error() = 0;
	ch->addr_type() = NS_AF_NONE;
	ch->prev_hop_ = agent_->node_address();

	ih->saddr() = agent_->node_address();
	ih->daddr() = IP_BROADCAST;
	ih->sport() = RT_PORT;
	ih->dport() = RT_PORT;
	ih->ttl_ = 1;		// only one hop away neighbors

	ah->ah_type = NDP_HELLO_MSG;
	ah->seq_no = 0;

	//DEBUG_NDP_MSG("Sending 'Hello' (0x" << ah->ah_type << ") to possible neighbors");
	agent_->target_->recv(p, (Handler*) 0);
```

```
}
```

A2.8-Ndp.h

```
#ifndef ns_ndp_h
#define ns_ndp_h

class NDP;

/*
 * Timers (Hello, Neighbor cache)
 */
class NDPHelloTimer : public Handler {
public:
        NDPHelloTimer(NDP*);
        void handle(Event*);
private:
        NDP *ndp_m;
        Event intr;
};

class NDPNeighborTimer : public Handler {
public:
        NDPNeighborTimer(NDP*);
        void handle(Event*);
private:
        NDP *ndp_m;
        Event intr;
};


/*
 * Neighbor class
 */
class Neighbor
{
        friend class NDP;
        friend class ANTNET;
        friend class AntRoutingTable;
        friend class NextHopEntry;

public:
        Neighbor(u_int32_t a) { nb_addr = a; }

protected:
        LIST_ENTRY(Neighbor) nb_link;
        nsaddr_t   nb_addr;
        double     nb_expire;     // ALLOWED_HELLO_LOSS * HELLO_INTERVAL
};
LIST_HEAD(ncache, Neighbor);


/*
 * Neigbor Discovery Protocol (NDP)
 */
class NDP
{
        friend class ANTNET;
        friend class AntRoutingTable;
        friend class NDPHelloTimer;
```

```
        friend class NDPNeighborTimer;
        friend class StartSimulationTimer;

public:
        NDP(ANTNET* a);
        ~NDP() {}
        void recv(Packet *p);

protected:
        ANTNET*             agent_;
        ncache          nbhead;
        int             off_ANTNET_;

        // Neigbor management
        void            nb_insert(nsaddr_t id);
        Neighbor*       nb_lookup(nsaddr_t id);
        void            nb_delete(nsaddr_t id);
        void            nb_purge(void);
        void            nb_print(void);
        int             nb_size(void);

        // Packet TX routines
        void            sendHello();

        NDPHelloTimer       htimer;
        NDPNeighborTimer    ntimer;
};
```

A3- Scripts to setup experiments

A3.1- config-experiment.sh

```
# Simulation script to compare MANET performance
#
# ./config-experiment.sh will create a $EXPERIMENT_NAME/config with the parameters of simulation
# ./run-experiment.sh $EXPERIMENT_NAME will actually run it
#

BASE="${HOME}/ant-routing/test"

#
# Default settings to write in config file
#

DESCRIPTION="\n\\\\
- Write additional settings here.\n\\\\
"
CODE_REV=""
PROTOCOLS="ANTNET DSR AODV"
TESTS="delay pdr rov hop"
MERGE_TESTS="pdr rov"
CBR_SRCS="10 20 30"
NUM_SCEN=10
NNODES=50
MAXX=1500
MAXY=300
RATE=4.0
PKTSIZE=64
MINSPEED=1
MAXSPEED=20
TRAINING_PERIOD=100
SIMTIME=900
PAUSETIMES="0 30 60 120 300 600 ${SIMTIME}"


#
# Prompt user to change default parameters
#
# (will dot it later... now we simply copy variables to $EXPERIMENT/config)

if [ $1 ]; then
        EXPERIMENT=${1};
else
        echo -n "Please introduce MANET experiment's name: ";
        read EXPERIMENT;
fi
echo "* MANET experiment name is: ${EXPERIMENT}"

#
# Create folders and config file
#
OUTDIR="${BASE}/experiments/${EXPERIMENT}"

echo -n "* Generating folder tree ... "
if [ ! -d ${OUTDIR} ]; then mkdir ${OUTDIR}; fi;
```

```
for FILES_DIR in cbr_files scenario_files trace_files plot_files; do {
        if [ ! -d ${OUTDIR}/${FILES_DIR} ]; then mkdir ${OUTDIR}/${FILES_DIR}; fi;
}; done;
echo "Done."

echo -n "* Generating config file ... "
echo "### AUTO-GENERATED CONFIG FILE ###

# Description of experiment
# Note: put '\n\\\\' at the end of each line to help GNUPlot display the label
DESCRIPTION=\"${DESCRIPTION}\"

# Subversion code revision. Just in case we want to run
# simultaneous experiments that need changes in NS-2 code
CODE_REV=\"${CODE_REV}\"

# Protocols that can be tested: AODV, DSR, DSDV
PROTOCOLS=\"${PROTOCOLS}\"

# Tests to be performed
#       pdr:    packet-delivery-ratio
#       rov:    routing-overhead
#       delay:  end-to-end-delay
#       hop:    hop-count-for-packets
TESTS=\"${TESTS}\"
MERGE_TESTS=\"${MERGE_TESTS}\"

# List of CBR sources to generate
CBR_SRCS=\"${CBR_SRCS}\"

# Number of random scenarios to be generated
NUM_SCEN=${NUM_SCEN}

# Number of nodes
NNODES=${NNODES}

# X dimension
MAXX=${MAXX}

# Y dimension
MAXY=${MAXY}

# Number of packets per second send by CBR sources
RATE=${RATE}

# Size of data packets in bytes
PKTSIZE=${PKTSIZE}

# Minimum speed of nodes in m/s (should be greater than zero)
MINSPEED=${MINSPEED}

# Maximum speed of nodes in m/s
MAXSPEED=${MAXSPEED}

# Training period for ANTNET in secs
TRAINING_PERIOD=${TRAINING_PERIOD}

# Simulation stop time
SIMTIME=${SIMTIME}
```

```
# Simulation pause times
PAUSETIMES=\"${PAUSETIMES}\"

### EOF" > ${OUTDIR}/config

# Edit config file for changes
vi ${OUTDIR}/config

echo "Done."

echo -n "* Copying MANET test file to '${EXPERIMENT}' folder ..."
if cp ${BASE}/manet-test.tcl ${OUTDIR}
then
        echo "Done."
else
        echo "Failed."
fi

# EOF
```

A3.2- run-experiment.sh

```
# Script to compare MANET protocols performance

BASE="${HOME}/ant-routing/test"
CODE="${HOME}/ant-routing/code"

#############################################################################
#
#               Configure MANET experiment
#
#############################################################################

# Print usage message if wrong input
function usage() {
   echo "Usage: $0 [-gdb] [Experiment name]";
          echo ""
          echo "Note: use [-gdb] switch to debug simulations with GDB."
          echo ""
   exit 1;
}

#
# Read options and experiment name
#
if [ $1 ]; then
          if [ "$1" == "-gdb" ]; then
                    if [ $2 ]; then
                    EXPERIMENT=${2}
                              GDB="gdb --args"
                              LOG_CLUSTER=""
                    else
                              usage
                    fi
          else
          EXPERIMENT=${1}
                    GDB=""
                    LOG_CLUSTER="< /dev/null 1> /dev/null 2>
${BASE}/experiments/${EXPERIMENT}/error.log"
          fi
else
          usage
fi

#
# Read tuned settings from config file
#
OUTDIR="${BASE}/experiments/${EXPERIMENT}"
CONFIG_FILE="${OUTDIR}/config"

if [ -f ${CONFIG_FILE} ]; then
          echo "* Parsing config file ... "
          source ${CONFIG_FILE}
          echo ""
          echo "  Experiment description: "
          echo "  ${DESCRIPTION}"
          echo ""
          echo "  Protocols to be compared: ${PROTOCOLS}"
```

```
        echo "  Scenario conditions (x${NUM_SCEN}):"
        echo "    Nodes          = ${NNODES} "
        echo "    Area           = ${MAXX}x${MAXY} m^2 "
        echo "    Speed          = ${MINSPEED} ~ ${MAXSPEED} m/s "
        echo "    CBR sources    = ${CBR_SRCS} "
        echo "    Data rate      = ${RATE} pkts/s "
        echo "    Training period = ${TRAINING_PERIOD} s "
        echo "    Pausetimes     = ${PAUSETIMES} s "
        echo "  Tests applied: "
        if echo "${TESTS}" | grep -q "delay"; then echo "    Average End-to-End Delay"; fi
        if echo "${TESTS}" | grep -q "pdr";   then echo "    Packet Delivery Ratio"; fi
        if echo "${TESTS}" | grep -q "rov";   then echo "    Routing Overhead"; fi
        if echo "${TESTS}" | grep -q "hop";   then echo "    Hop Count for Packets"; fi
        echo ""
else
        echo "Error reading config file. Are you sure is there?"
        exit 1
fi

#
# Save last pattern simulated just in case of crashing =:-O
#
LAST_TRACE=${OUTDIR}/last-tracefile
LAST_PLOT=${OUTDIR}/last-plotfile

#
# Only _FIVE_ simulations at a time, please!
#
MAX_CPUS=8
CPU_FREE="false"
for ((i=1; i <= ${MAX_CPUS}; i++))
do
        PIDFILE[${i}]=${OUTDIR}/pid_${i}
        if [ ! -f ${PIDFILE[${i}]} ]; then
                CPU_FREE="true"
                echo "$$" > ${PIDFILE[${i}]}
                PIDFILE=${PIDFILE[${i}]}
                break
        fi
done
if [ ${CPU_FREE} == "false" ]; then
        echo "MANET experiment already running with PIDs:"
        for ((i=1; i <= ${MAX_CPUS}; i++))
        do
                echo "* `cat ${PIDFILE[${i}]}`"
        done
        echo "Please stop them before restarting."
        if [ -f ${LAST_TRACE} ]; then echo "Last trace generated: `cat ${LAST_TRACE}`"; fi
        if [ -f ${LAST_PLOT} ]; then echo "Last trace post-processed: `cat ${LAST_PLOT}`"; fi
        exit 1
fi


# Additional settings that should not be overwritten
TRAFFIC_TYPE="cbr"
SEED=1.0
SPEEDSEL=1
PAUSESEL=2
```

```
#
# Checking for programs and folders
#
if [ "${CODE_REV}" == "" ]; then
        NS="ns"
else
        NS="ns-r${CODE_REV}"
fi
NS=${CODE}/ns2/${NS}
CBRGEN=${BASE}/cbrgen.tcl
SETDEST=${BASE}/setdest
MANET_TEST=${OUTDIR}/manet-test.tcl
FILTERS=${BASE}/filters

##########################################################################
#
#               Start MANET experiment
#
##########################################################################

cd ${OUTDIR}
echo "* Starting experiment [${EXPERIMENT}] in `pwd` at `date` "

# ------------------------------------------------------------
#             I N I T I A L I Z A T I O N
# ------------------------------------------------------------

#
# Only one CPU allowed here
#
INIT_PID=${OUTDIR}/init.pid
if [ -s ${INIT_PID} ]; then
        echo "Process `cat ${INIT_PID}` is already initializing experiment."
        echo "Wait a few minutes and try again."
        exit 1
fi
echo "$$" > ${INIT_PID}

#
# Create additional folders for every PROTOCOL to be compared
#
for FILES_DIR in cbr_files scenario_files trace_files plot_files; do {
        if [ ! -d ${OUTDIR}/${FILES_DIR} ]; then mkdir ${OUTDIR}/${FILES_DIR}; fi;
}; done;
for PR in ${PROTOCOLS}; do {
        for FILES_DIR in trace_files plot_files; do {
                if [ ! -d ${OUTDIR}/${FILES_DIR}/${PR} ]; then mkdir
${OUTDIR}/${FILES_DIR}/${PR}; fi;
        }; done;
}; done;

#
# Generate connection patterns that will be used in all scenarios
#
for MC in ${CBR_SRCS};
do
        CP_FILE="cbr_files/nn${NNODES}-mc${MC}-r${RATE}.${TRAFFIC_TYPE}"
        if [ ! -s ${CP_FILE} ]; then
                echo -n "* Generating connection pattern with ${MC} max connections ... "
```

```
                ${NS} ${CBRGEN} -type ${TRAFFIC_TYPE} -nn ${NNODES} -seed ${SEED} -mc
${MC} \
                        -rate ${RATE} -warmup ${TRAINING_PERIOD} -pktsize ${PKTSIZE} >
${CP_FILE}
                echo "Done."
        fi
done

#
# Generate all scenario file for each pause time (0=mobile, SIMTIME=static)
#
SC_PATTERN="nn${NNODES}-${MAXX}x${MAXY}-m${MINSPEED}-M${MAXSPEED}"
for (( SC=0; SC < NUM_SCEN; SC++ ))
do
        for PAUSE in ${PAUSETIMES};
        do
                SC_FILE="scenario_files/${SC_PATTERN}-s${SC}-p${PAUSE}.scen"
                if [ ! -s ${SC_FILE} ]; then
                        echo -n "* Generating movement pattern ${SC} with pause time ${PAUSE}s ... "
                        ${SETDEST} -v 2 -n ${NNODES} -s ${SPEEDSEL} -m ${MINSPEED} -M
${MAXSPEED} -t ${SIMTIME} -P ${PAUSESEL} \
                                -p ${PAUSE} -x ${MAXX} -y ${MAXY} > ${SC_FILE}
                        echo "Done."
                fi
        done
done

#
# Now, allow other process to simulate CONCURRENTLY by removing INIT_PID
# They will find connection and movement patterns already generated
# and continue through this script
#
rm ${INIT_PID}


# -----------------------------------------------------------------
#       C O N C U R R E N T   S I M U L A T I O N S
# -----------------------------------------------------------------

#
# Now start the hard work, simulate all the scenarios and connection patterns (70*3=210)
#
for PR in ${PROTOCOLS};
do
        for MC in ${CBR_SRCS};
        do
                CP_FILE="cbr_files/nn${NNODES}-mc${MC}-r${RATE}.${TRAFFIC_TYPE}"
                for (( SC=0; SC < NUM_SCEN; SC++ ));
                do
                        TR_PATTERN="nn${NNODES}-mc${MC}-r${RATE}-${MAXX}x${MAXY}-
m${MINSPEED}-M${MAXSPEED}-s${SC}"
                        for PAUSE in ${PAUSETIMES};
                        do
                                SC_FILE="scenario_files/${SC_PATTERN}-s${SC}-p${PAUSE}.scen"
                                TR_FILE="trace_files/${PR}/${TR_PATTERN}-p${PAUSE}.tr"

                                # Check if another process is working on this simulation
                                # If yes, skip and continue with next one
                                LOCK_FILE="${TR_FILE}.pid"
```

```
                        if [ -s ${LOCK_FILE} ]; then
                                echo "* Process `cat ${LOCK_FILE}` already working on
${TR_FILE}. Skipping..."
                                continue;
                        fi
                        # Otherwise, lock it
                        echo "$$" > ${LOCK_FILE}

                        # If we have the plot files, we don't need the trace file, should check
before wasting CPU time!!!
                        NEED_TRACE="no"
                        for TEST in ${TESTS};
                        do
                                if [ ! -s "plot_files/${PR}/${TEST}-${TR_PATTERN}-
p${PAUSE}.xgr" ] && \
                                  [ ! -s "plot_files/${PR}/${TEST}-${TR_PATTERN}-all.xgr"
]; then
                                        NEED_TRACE="yes"
                                        break
                                fi
                        done

                        TRACE_GZIPPED="no"
                        if [ ${NEED_TRACE} == "yes" ]; then
                                echo "* Testing ${PR} routing protocol on scenario
${TR_PATTERN}-p${PAUSE} ... "
                                if [ -s ${TR_FILE}.gz ]; then
                                        echo "  Trace file found."
                                        TRACE_GZIPPED="yes"
                                        AWK_FEED="gunzip -c ${TR_FILE}.gz"
                                elif [ ! -s ${TR_FILE} ]; then
                                        echo "  Trace file not found. Simulating ... "
                                        ${GDB} ${NS} ${MANET_TEST} -adhocRouting
${PR} -cp ${CP_FILE} -sc ${SC_FILE} \
                                                -tr ${TR_FILE} -x ${MAXX} -y ${MAXY} -
nn ${NNODES} -stop ${SIMTIME} ${LOG_CLUSTER} || continue
                                        AWK_FEED="cat ${TR_FILE}"
                                fi

                                # Update last pattern fully simulated for recovery purposes
                                echo "${PR}-${TR_PATTERN}-p${PAUSE}" >
${LAST_TRACE}

                                # Once we have the trace file (either generated or
decompressed), we run all the tests
                                echo -n "  Running tests: "
                                for TEST in ${TESTS};
                                do
                                        PL_FILE="plot_files/${PR}/${TEST}-
${TR_PATTERN}-p${PAUSE}.xgr"
                                        if [ ! -s ${PL_FILE} ]; then
                                                echo -n "${TEST} ... "
                                                ${AWK_FEED} | awk -f
${FILTERS}/${TEST}.awk -v nn=${NNODES} pause=${PAUSE} > ${PL_FILE}
                                        fi
                                        echo "${PR}-${TR_PATTERN}-p${PAUSE}-
${TEST}" > ${LAST_PLOT}
                                done
                                echo "Done."
```

```
                                        # And then we "gzip" the file to save space on the server
                                        if [ ${TRACE_GZIPPED} == "no" ]; then
                                                echo -n "  Compressing trace file to save space ... "
                                                gzip ${TR_FILE}
                                                echo "Done."
                                        fi
                                else
                                        echo "* Routing protocol ${PR} already tested on scenario
${TR_PATTERN}-p${PAUSE}."
                                fi

                                # Removing lock (not really necessary, but safer)
                                rm ${LOCK_FILE}
                        done
                done
        done
done
echo "* MANET experiment [${EXPERIMENT}] finished on `date` "
rm ${PIDFILE}

# EOF
```

A3.3- average-plots.sh

```
#!/bin/bash
# Script to calculate mean value of plots generated by simulation

BASE="${HOME}/ant-routing/test"

#
# Read experiment name
#
if [ ! $1 ]
then
	echo -n "Which experiment do you want to average results? "
	read EXPERIMENT
else
	EXPERIMENT="$1"
fi

#
# Read settings
#
CONFIG_FILE="${BASE}/experiments/${EXPERIMENT}/config"
if [ ! -s ${CONFIG_FILE} ]
then
	echo "Error reading config file. Exiting ... "
	exit 1
fi
source ${CONFIG_FILE}

#
# Average plots
#
PLOT_FILES=${BASE}/experiments/${EXPERIMENT}/plot_files
FILTERS=${BASE}/filters

for PR in ${PROTOCOLS};
do
	for MC in ${CBR_SRCS};
	do
		# Merge plot files with different pausetime in a unique file with tag "all"
		#
		for MT in ${MERGE_TESTS};
		do
			for ((S=0; S<${NUM_SCEN}; S++))
			do
				PATTERN="${PLOT_FILES}/${PR}/${MT}-nn${NNODES}-
mc${MC}-r${RATE}-${MAXX}x${MAXY}-m${MINSPEED}-M${MAXSPEED}-s${S}"
				MERGED_FILE="${PATTERN}-all.xgr"
				if [ ! -s ${MERGED_FILE} ]; then
					echo -n "* Merging plot files for test '${MT}' (scenario ${S}) ...
"
					for PAUSE in ${PAUSETIMES};
					do
						echo -n "${PAUSE} "
						IN="${PATTERN}-p${PAUSE}.xgr"
						if [ -s ${IN} ]; then
							cat ${IN} >> ${MERGED_FILE}
							rm ${IN}
```

```
                                        else
                                                echo -n "(failed) "
                                        fi
                                done
                                echo "Done."
                        fi
                done
        done

        # Average plot files from different scenarios and put tag "avg"
        #
        for T in ${TESTS};
        do
                PATTERN="${PLOT_FILES}/${PR}/${T}-nn${NNODES}-mc${MC}-
r${RATE}-${MAXX}x${MAXY}-m${MINSPEED}-M${MAXSPEED}"
                echo "* Averaging ${PATTERN} files ..."
                if [ "$T" == "hop" ]; then
                        for P in ${PAUSETIMES};
                        do
                                IN=${PATTERN}-s*-p${P}.xgr
                                OUT=${PATTERN}-avg-p${P}.xgr
                                cat ${IN} | grep ^[0-9] | awk -f ${FILTERS}/mean-${T}.awk -v
pause=${P} nn=${NNODES} > ${OUT}
                        done
                elif [ "$T" == "delay" ]; then
                        # Need to merge data for every pausetime before averaging delay data
                        #
                        for (( S=0; S < ${NUM_SCEN}; S++ ))
                        do
                                DELAY_MERGED=${PATTERN}-s$S-all.xgr
                                echo "### TitleText: Average End-To-End Delay (scenario $S
merged)" > ${DELAY_MERGED}
                                for P in ${PAUSETIMES};
                                do
                                        IN=${PATTERN}-s$S-p$P.xgr
                                        # last line has average delay
                                        tail -1 ${IN} >> ${DELAY_MERGED}
                                done
                        done
                        #
                        # End processing
                        IN=${PATTERN}-s*-all.xgr
                        OUT=${PATTERN}-avg.xgr
                        cat ${IN} | grep ^[0-9] | awk -f ${FILTERS}/mean-${T}.awk > ${OUT}
                else
                        # WARNING! Should not be ${PATTERN}-s*-p*.xgr files
                        # 2006-04-21 Added trail '-all' when merging data for every pausetime to
avoid previous warning
                        IN=${PATTERN}-s*-all.xgr
                        OUT=${PATTERN}-avg.xgr
                        cat ${IN} | grep ^[0-9] | awk -f ${FILTERS}/mean-${T}.awk > ${OUT}
                fi
        done
    done
done
```

A3.4- gen-gnuplot-commands.sh

```
#!/bin/bash
# Script to create plots generated by simulation using GNUPLOT

BASE="${HOME}/ant-routing/test"

#
# Read experiment
#
if [ ! $1 ]; then
        echo "Usage: ./$0 [Experiment] [Terminal]"
        echo -n "What experiment do you want to plot? "
        read EXPERIMENT
else
        EXPERIMENT="$1"
fi

#
# Read settings
#
CONFIG_FILE="${BASE}/experiments/${EXPERIMENT}/config"
if [ ! -s ${CONFIG_FILE} ]; then
        echo "Error reading config file. Exiting ... "
        exit 1
fi
source ${CONFIG_FILE}

#
# Create a descriptive title by writing all info related to
# current experiment, such as numeber of nodes, pausetimes, etc.
#
DESC_TITLE="[#nodes=${NNODES} #scenarios=${NUM_SCEN} area=${MAXX}x${MAXY} \
speed=${MINSPEED}~${MAXSPEED} srcs=${CBR_SRCS} rate=${RATE} \
training=${TRAINING_PERIOD}]"

DESC_LABEL="Additional settings for experiment ${EXPERIMENT}: ${DESCRIPTION}"

#
# Additional settings
#
if [ ! $2 ]; then
        TERMINAL="windows" # can also be set to 'latex'
else
        TERMINAL="$2"
fi

#
# Generate GNUPLOT commands
#
GP_SCRIPT="${BASE}/experiments/${EXPERIMENT}/plot_files/commands.gp"
echo "* Writing GNUPLOT commands to '${GP_SCRIPT}' ... "
echo "# GNUPLOT commands
set autoscale
set terminal ${TERMINAL}
### Extra information
set x2label \"${DESC_TITLE}\"
set label 55 \"${DESC_LABEL}\" at screen 0.2, screen 0.7
```

```
###
" > ${GP_SCRIPT}
for T in ${TESTS};
do
	# Set title and labels for each test
	case $T in
	delay)
		TITLE="Average End-to-End Delay"
		LEGEND_POS="right top"
		X_LEG="Pause time (sec)"
		Y_LEG="Delay (sec)"
		END_PATTERN="-avg.xgr"
		;;
	rov)
		TITLE="Routing Overhead"
		LEGEND_POS="right top"
		X_LEG="Pause time (sec)"
		Y_LEG="Number of Packets"
		END_PATTERN="-avg.xgr"
		;;
	pdr)
		TITLE="Packet Delivery Ratio"
		LEGEND_POS="right bottom"
		X_LEG="Pause time (sec)"
		Y_LEG="# Packets received / # Packets sent"
		END_PATTERN="-avg.xgr"
		;;
	*)
		echo "  No plot settings defined for test '$T'. Skipping ..."
		continue
		;;
	esac

	# different plot for each protocol
	for P in ${PROTOCOLS};
	do
		if [ "${TERMINAL}" == "latex" ]; then
			ADDITIONAL_SETTINGS="set output \"$T-$P.tex\""
		else
			ADDITIONAL_SETTINGS=""
		fi
		echo -n "# Test: $T, AdhocRoutingProtocol: $P
set title \"${TITLE} for $P\"
set key ${LEGEND_POS}
set xlabel \"${X_LEG}\"
set ylabel \"${Y_LEG}\"
${ADDITIONAL_SETTINGS}
plot " >> ${GP_SCRIPT}
		for S in ${CBR_SRCS};
		do
			PATTERN="$P/$T-nn${NNODES}-mc$S-r${RATE}-${MAXX}x${MAXY}-
m${MINSPEED}-M${MAXSPEED}${END_PATTERN}"
			echo -n "\"${PATTERN}\" using 1:2 title \"$S sources\" with lines," >>
${GP_SCRIPT}
		done
		echo "0 title \"\" with lines lw 0" >> ${GP_SCRIPT}
		echo "pause -1 \"Press RETURN to see next plot\"
		" >> ${GP_SCRIPT}
	done
```

```
	# comparison plot with every protocol
	if [ "${TERMINAL}" == "latex" ]; then
		ADDITIONAL_SETTINGS="set output \"$T-all.tex\""
	else
		ADDITIONAL_SETTINGS=""
	fi
	echo -n "# Test: $T, AdhocRoutingProtocol: all
set title \"${TITLE}\"
${ADDITIONAL_SETTINGS}
plot " >> ${GP_SCRIPT}
	for P in ${PROTOCOLS};
	do
		for S in ${CBR_SRCS};
		do
			PATTERN="$P/$T-nn${NNODES}-mc$S-r${RATE}-${MAXX}x${MAXY}-
m${MINSPEED}-M${MAXSPEED}${END_PATTERN}"
			echo -n "\"${PATTERN}\" using 1:2 title \"$P, $S sources\" with lines," >>
${GP_SCRIPT}
		done
	done
	echo "0 title \"\" with lines lw 0" >> ${GP_SCRIPT}
	echo "pause -1 \"Press RETURN to see next plot\"
	" >> ${GP_SCRIPT}

done
echo "* Done."

exit 0
```

A3.5- purge-plots.sh

```
#!/bin/bash
# Script to purge useless plots generated by simulation

purge() {
	for PR in ${PROTOCOLS};
	do
		for T in ${MERGE_TESTS};
		do
			for MC in ${CBR_SRCS};
			do
				IN=${BASE}/plot_files/${PR}/${T}-nn${NNODES}-mc${MC}-
r${RATE}-${MAXX}x${MAXY}-m${MINSPEED}-M${MAXSPEED}-s*-p*.xgr
				if rm ${IN}; then echo "o"; else echo "x"; fi
			done
		done
	done
}

#
# Read experiment
#
if [ ! $1 ]; then
	echo -n "What experiment do you want to purge? "
	read FOLDER
else
	FOLDER="$1"
fi

#
# Read settings
#
BASE="${HOME}/ant-routing/test/experiments/${FOLDER}"
if [ ! -s ${BASE}/config ]; then
	echo "Error reading config file. Exiting ... "
	exit 1
fi
source ${BASE}/config

#
# Warning and purge files
#
echo ""
echo "All the individual plotfiles (those finishing with '-p*.xgr') from "
echo "${BASE} will be ERASED."
echo -n "Are you sure? (yes/no): "

read answer
case ${answer} in
	yes)
		echo -n "Deleting ... "
		purge
		echo "Done."
		;;
	*)
		echo "Not deleting."
		;;
esac
```

```
exit 0
```

A3.6- optimal-hop-count.sh

```
#!/bin/bash
# Script to get difference between actual hop count and optimal value

BASE="${HOME}/ant-routing/test"

#
# Read experiment name
#
if [ ! $1 ]
then
        echo -n "Which experiment do you want to get optimal hop count? "
        read EXPERIMENT
else
        EXPERIMENT="$1"
fi

#
# Read settings
#
CONFIG_FILE="${BASE}/experiments/${EXPERIMENT}/config"
if [ ! -s ${CONFIG_FILE} ]
then
        echo "Error reading config file. Exiting ... "
        exit 1
fi
source ${CONFIG_FILE}

#
# Average plots
#
PLOT_FILES=${BASE}/experiments/${EXPERIMENT}/plot_files
FILTERS=${BASE}/filters

for PR in ${PROTOCOLS};
do
        for MC in ${CBR_SRCS};
        do
                T="hop"
                PATTERN="${PR}/${T}-nn${NNODES}-mc${MC}-r${RATE}-
${MAXX}x${MAXY}-m${MINSPEED}-M${MAXSPEED}"
                PATTERN_FILE="${PLOT_FILES}/${PATTERN}"
                echo "* Getting optimal hop count from ${PATTERN} ..."

                OUT=${PATTERN_FILE}-avg.xgr
                if [ -s ${OUT} ]; then
                        echo "  Output file exists. Should remove it before overwriting."
#                       exit 1
                fi

                echo "### Number of packets following optimal hop-count (mean & standard deviation)" >
${OUT}
                echo "### [pausetime] [hops] [# pkts forwarded] [# pkts diff optimal value] [% pkts_hops
[mean] [std]] [% pkts_diff [mean] [std]] " >> ${OUT}

                for P in ${PAUSETIMES};
```

```
                do
                        IN=${PATTERN_FILE}-avg-p${P}.xgr
                        if [ ! -s ${IN} ]; then
                                echo "  Cannot find input file. Should run 'average-plots.sh' before."
                                exit 1
                        fi
                        ###
                        ### Line starting with 0 contains number of packets that follow optimal hop-count
                        ###
                        echo -n "${P} " >> ${OUT}
                        cat ${IN} | grep ^[0] >> ${OUT}
                done
                echo "  Done."
        done
done
```

A3.7- error-rate-plot.sh

```
#!/bin/bash
# Script to prepare plot for error rate

BASE="${HOME}/ant-routing/test"

#
# Settings
#
EXP_BASE="random-50-drop"
EXP_DROP="05 10 15 20"
CBR_SRCS="10 20"
PAUSE="0"
TEST="pdr"
PROTOCOLS="ANTNET DSR AODV"

#
# Do it!
#
for PR in ${PROTOCOLS};
do
        for MC in ${CBR_SRCS};
        do
                PATTERN="${TEST}-nn50-mc${MC}-r4.0-1500x300-m1-M20"
                OUT=${BASE}/experiments/${EXP_BASE}-${PR}-${PATTERN}-p${PAUSE}.xgr

                echo "### Packet delivery ratio for pausetime ${PAUSE}" > ${OUT}
                echo "### [error_rate] [pausetime] [pdr] " >> ${OUT}

                # Also w/o error
                        PLOT_FILES=${BASE}/experiments/random-50-normal/plot_files
                        IN="${PLOT_FILES}/${PR}/${PATTERN}-avg.xgr"
                        echo -n "00 " >> ${OUT}
                        cat ${IN} | grep ^[${PAUSE}] >> ${OUT}

                for E in ${EXP_DROP}
                do
                        PLOT_FILES=${BASE}/experiments/${EXP_BASE}${E}/plot_files
                        IN="${PLOT_FILES}/${PR}/${PATTERN}-avg.xgr"
                        echo -n "${E} " >> ${OUT}
                        cat ${IN} | grep ^[${PAUSE}] >> ${OUT}
                done
        done
done
```

Annex II - Resum

Introducció:

Aquest projecte neix de la necessitat actual de trobar altres protocols d'enrutació que encaminin la informació d'una manera més eficient a les xarxes mòbils. Actualment, els protocols utilitzats han demostrat la seva eficiència en les comunicacions estàtiques i poc canviants. No obstant, la tendència actual del mercat ens deixa entreveure un futur amb un escenari cada vegada més mòbil i heterogeni, on els antics protocols d'enrutació basats en una matemàtica clàssica no cobreixen totes les necessitats.

Els protocols actuals resulten massa rígids en una xarxa mòbil: el temps que aquests inverteixen per a resoldre una ruta o per a buscar rutes alternatives són paràmetres que s'han de millorar en els futurs protocols. Per tant, la cerca d'un protocol millor esdevé una necessitat evident si volem que les xarxes mòbils siguin més eficients i robustes.

Des de que l'home és home, sempre ha examinat el seu voltant en busca de solucions pels seus problemes. En molts casos, la natura li ha servit d'inspiració per a produir grans invents, des d'eines basades en becs d'ocells, fins a trens d'alta velocitat i passant per complicades teories matemàtiques.

Actualment, han sorgit una sèrie d'algoritmes basats en el moviment dels animals que s'estructuren en eixams. La particularitat d'aquesta organització radica en el fet que tots els elements són iguals entre si, sense que existeixi un individu central que organitzi la seva estructura. Aquest tipus d'estructuració és conegut com a "intel·ligència d'eixam" (*Swarm Intelligence*). En són clars exemples els ramats de Nyus, els bancs d'ocells o de peixos, les abelles, les formigues, etc. Diferents especialistes n'han observat el funcionament d'aquests eixams i n'han descrit les bases per poder crear els algoritmes que els imiten.

Així doncs, un dels principals objectius d'aquest projecte és realitzar un protocol de comunicacions basat amb un algoritme "bio-inspirat", el qual resulti més eficient que els actuals protocols d'enrutament dins de les xarxes mòbils ad-hoc.

La manera amb què s'implementi aquest algoritme resultarà decisiva per poder analitzar la seva viabilitat.

Per poder extreure'n conclusions, per una banda, caldrà comparar el nou protocol bio-inspirat amb un altre protocol comercial, el qual resulti especialment eficient en l'escenari escollit; i per l'altra, caldrà realitzar experiments que així ho demostrin i avaluar tots els paràmetres possibles per tal de donar una justificació sòlida, basada en les dades obtingudes dels experiments.

Objectius:

L'objectiu final del projecte és trobar un protocol d'enrutament basat en la natura, i que demostri ser més eficient en les xarxes mòbils ad-hoc que els protocols utilitzats actualment.

No obstant, aquesta no és l'única finalitat del projecte. Per arribar a la conclusió final, primerament s'han de complir altres objectius intermedis per dotar de rigor les nostres conclusions.

Un primer objectiu serà l'estudi en profunditat del mecanisme de comunicació entre routers. Essencialment, conèixer la seva naturalesa és el primer pas per arribar a comprendre'n el seu funcionament.

El segon objectiu ve donat per l'estudi dels diferents algoritmes utilitzats per a la realització dels protocols d'enrutació. En aquesta secció, també s'estudiaran els protocols actuals i s'avaluaran les seves virtuts i mancances.

El tercer objectiu és un treball de documentació. Cal un treball de camp per recopilar informació referent a algoritmes bio-inspirats. Hi ha molta informació al respecte i cal destriar quina és la informació que ens pot ajudar en la realització del nostre algoritme i quina no.

Així mateix, convé destacar que un cop decidit quin algoritme biològic ens servirà d'inspiració, hem de dissenyar l'algoritme des d'un punt de vista telemàtic per poder-lo implementar posteriorment.

La seva implementació és un dels punts més delicats. Per tant, escollir un bon entorn de proves serà vital per poder obtenir un protocol, el qual es comporti de la manera més semblant a l'algoritme dissenyat.

Un cop tinguem programat l'algoritme, cal comparar-lo amb algun algoritme existent per tal d'avaluar la seva viabilitat. El test es realitzarà en dos escenaris diferents: un petit que simuli un nucli urbà i un altre que simuli un medi més despoblat. La resta de paràmetres tindran els mateixos valors, per tal de comparar els dos protocols en les mateixes condicions.

De la comparativa del protocol bio-inspirat amb el clàssic, n'haurà de sortir una conclusió final que constati quins avantatges té un protocol sobre l'altre i perquè un és millor que l'altre. Convindrà analitzar els resultats dels experiments i, posteriorment, concloure l'explicació amb quins paràmetres seran els més rellevants perquè un protocol resulti eficient.

Cal destacar que és igualment important analitzar els punts dèbils del nostre protocol per orientar futures investigacions en el mateix camp d'estudi.

Conclusions:

Les conclusions obtingudes de la realització d'aquest projecte no es limiten únicament a la valoració del protocol resultant. Per tant, passaré a analitzar cada apartat, ja que el resultat de cadascun d'ells ha estat determinant pel resultat final. A nivell personal, intentaré ser crític en tots ells per tal de trobar altres solucions que puguin encaminar posteriors treballs.

El primer pas que calia decidir era quin esquema social serviria de base al nostre algoritme. Després d'analitzar les diferents opcions, va quedar clar que les més adequades per la implementació en xarxes ad-hoc eren l'estructura social de les abelles i la de les formigues. El sistema utilitzat per les abelles necessita les tres dimensions per a un correcte funcionament, limitant la seva aplicació a elements que es puguin moure lliurement per l'espai com poden ser els avions o els satèl·lits. L'opció d'escollir l'estructura social de les formigues com a columna vertebral de l'algoritme rau en el fet que les formigues utilitzen un mecanisme molt senzill per comunicar-se i per actuar en cas de què l'escenari canviï. Crec que aquesta decisió va ser encertada, ja que de les opcions estudiades, aquesta era la que s'adaptava millor al nostre escenari.

La creació de l'algoritme es va basar en la interpretació que va fer Marco Dorigo a la seva tesi doctoral l'any 2003. No hi ha cap dubte en afirmar que aquest matemàtic va fer la millor aproximació fonamentant la teoria *Ant Colony Optimization*, la qual ha servit com a base per moltes investigacions. Analitzant l'algoritme que he creat, es pot afirmar que és un algoritme totalment operatiu i vàlid. No obstant, segurament una posterior observació més acurada del sistema organitzatiu de les formigues inclouria alguna modificació que milloraria les prestacions de l'algoritme.

La implementació del protocol ha estat un punt clau per l'obtenció del resultat final. Vaig optar per l'opció d'implementar-lo al simulador de xarxes NS-2, ja que aquest programa és lliure, està força documentat i existeixen moltes llibreries útils que he pogut utilitzar per a la creació del protocol. He tingut dificultats a l'hora d'instal·lar-lo, ja que s'havien d'instal·lar molts mòduls i s'havien de modificar diferents paràmetres, depenent de la versió de Linux instal·lada. La utilització d'una màquina virtual va ser una font de problemes. Després de la instal·lació dels mòduls s'havia de fer un per un, depenent de l'ús que li volguéssim donar al nostre simulador. La conclusió d'aquest apartat no es gaire bona ja que vaig perdre molt de temps en preparar l'eina. En estudis futurs, aconsellaria provar amb altres simuladors.

La programació de l'algoritme està feta en C++ i els filtres amb TCL. El llenguatge de programació C++ està orientat a objectes i, tal com el vaig estudiar a la universitat, no he tingut gaires problemes; però el llenguatge TCL —que funciona per events—, era nou per mi i m'ha costat aprendre com funcionava. És en aquest punt en el que Xesc Arbona m'ha prestat la seva ajuda per entendre aquest llenguatge. Al principi tenia moltes errades però, progressivament, he anat millorant la programació en aquest aspecte.

He realitzat uns *scripts* per facilitar la feina d'anar provant els diferents algoritmes amb diferents escenaris. Després de varies proves, vaig decidir que els dos escenaris que em servirien d'exemple serien iguals en número de nodes i de connexions, i només es diferenciarien en la grandària de l'escenari: l'un seria de 1500m x 300m i l'altre de 2000m x 2000m, l'un simularia un entorn urbà i l'altre un entorn menys poblat.

Els ajustos dels diferents paràmetres dels algoritmes han resultat de vital importància. En definitiva, és en aquest apartat on m'ho he passat més bé, ja que he pogut analitzar els baixos rendiments del protocol i aplicar-li les modificacions per millorar-lo. He descobert que si "les formigues que van cap endavant" (*Forward Ants*) descobrint la xarxa només són enviades als routers veïns, el temps de conversió millora dràsticament. Per tant, cal destacar que si ajustem la freqüència d'enviament d'aquestes aconseguirem que la taxa de paquets enviats/rebuts sigui més alta, apropant-se en tots els casos als resultats obtinguts pel protocol AODV.

En conclusió, cal comentar que el protocol creat a partir de l'organització social de les formigues pot ser vàlid per a una primera aproximació, però caldria fer-li certes modificacions perquè esdevingués realment una opció viable.

Bibliography

[1] J. Baras and H. Mehta. A Probabilistic Emergent Routing Algorithm for Mobile Ad hoc Networks (PERA), 2003.

[2] Gianni Di Caro and Marco Dorigo. AntNet: Distributed Stigmergetic Control for Communications Networks. Journal of Artificial Intelligence Research, 9:317–365, 1998. http://citeseer.nj.nec.com/dicaro98antnet.html.

[3] M. Gunes, U. Sorges, and I. Bouazzi. ARA – the ant-colony based routing algorithm for MANETs, 2002.

[4] Information Sciences Institute. NS-2 network simulator. Software Package, 2003. http://www.isi.edu/nsnam/ns/.

[5] David B Johnson and David A Maltz. Dynamic Source Routing in Ad Hoc Wireless Networks. In Imielinski and Korth, editors, Mobile Computing, volume 353. Kluwer

[6] Academic Publishers, 1996. http://citeseer.nj.nec.com/johnson96dynamic.html.

[7] Z. Liu, M. Kwiatkowska, and C. Constantinou. A biologically inspired optimization to AODV routing protocol. pages 106–111, December 2004.

[8] C. Perkins. Ad Hoc On Demand Distance Vector (AODV) Routing, 1997. http://citeseer.nj.nec.com/article/perkins99ad.html.

[9] Otto Wittner. Ant-like mobile agents, December 2005.

[10]Steven C. Potter .Networking Basics: The OSI Model.2008.
http://www.spottek.ca/Tutorials/NetOSI.html

[11]Wikipedia. Internet Protocol version 4 (IPv4). http://en.wikipedia.org/wiki/IPv4

[12]David A. Maltz. The Dynamic Source Routing Protocol.2003.
http://www.cs.cmu.edu/~dmaltz/dsr.html

[13]T. Clausen. Optimized Link State Routing Protocol (OLSR).2003.
http://www.ietf.org/rfc/rfc3626.txt

[14]Frederick Ducatelle.2007.Adaptive Routing in Ad Hoc Wireless Multi-hop
Networks.http://www.idsia.ch/~frederick/anthocnet/thesis_01.pdf

[15]Dervis Karaboga. Artificial Bee Colony (ABC) Algorithm. 2010 .
http://mf.erciyes.edu.tr/abc/