BISON
IST-2001-38923

Biology-Inspired techniques for Self Organization in dynamic Networks

Demonstrator 1: Ant-based monitoring on software IP routers

Deliverable Number:	D14
Delivery Date:	March 2006
Classification:	Public Circulation
Contact Authors:	Poul Heegaard, Ingebrigt Fuglem
Document Version:	Final (15th March 2006)
Contract Start Date:	1 January 2003
Duration:	36 months
Project Coordinator:	Università di Bologna (Italy)
Partners:	Telenor ASA (Norway), Technische Universität Dresden (Germany), IDSIA (Switzerland)

Project funded by the European Commission under the Information Society Technologies Programme of the 5th Framework (1998-2002)

Abstract

The purpose of *AntPing*, an ant-based routing and monitoring system, is to demonstrate a working implementation on a software based IP router. The AntPing is used to establish, maintain, and monitor virtual paths through a 10 node network. The network is a replication of the Telenor core Internet backbone topology.

The demonstrator visualizes the inner workings of the ant algorithm by animation of ants moving and dropping in the network, and topology changes like link failures and restorations. The animation also shows the ants that do not find the destination but are dropped because the TTL is expired. In addition, the changes in cost values of each virtual path is plotted as a function over time, both the cost of the current best cost, and the last cost, even when rejected by the elite selection.

To supplement the demonstrator, a series of simulations in *ns2* are conducted. The results from these simulations show how the AntPing compares to Ping with link state routing under network dynamics, and how AntPing scales with increasing number of virtual connections and with increasing network size.

Contents

Part I
Introduction

Proper network management and provisioning of service guarantees and differentiation in the Internet requires high quality measurement data. It is a great challenge to obtain sufficient, necessary, correct, and fresh data for various network management and traffic engineering aspects, security and fraud detection, and accounting. Performance monitoring collects and aggregates quality attributes about a service, or a set of services, e.g. by use of active monitoring with *Ping* or *traceroute*. It is typically of interest to verify that a service is delivered according to the specifications, e.g. as given in Service Level Agreements.

As a supplement to best practice in performance monitoring it is of interest to look into the potential of collection and aggregation of performance data from indices of a complex adaptive system based on swarm intelligence. The idea is to observe temporal variables of a swarm intelligence system designed e.g. for solving reliable path management problems [17, 15] or optimizing routing [13, 3] in the Internet. These variables will change as the network changes and, hence, as the corresponding quality of the delivered service. Previous studies [4] of the transient behaviour of *CE ants* [5], a Cross Entropy based Ant system for path management, identified that several adaptive components of such a system can be used as indicators of the network condition status with respect to traffic load level and topology. The indicators are the stochastic routing matrix (the pheromones), routing path probability, cost values, and grade of convergence (denoted temperature in CE ants).

To demonstrate its feasibility, a prototype implementation of a CE ants in software IP router is prepared. The prototype is named *AntPing* and will demonstrate the CE ants principles in a small-scale network. The implementation is based on *Click* [7], a modular software router running on standard Linux PCs, and *hping*3[1], providing a rich TCL based API for socket programming. A previous working prototype implementation based on a Mobile Agent System (Java based) demonstrated the implementation feasibility [8]. However, the Java solution suffers from severe performance limitations even in a small-scale demo network. Providing new modules to the click router will provide useful insights in the complexity of implementing swarms in real operating routers, and detect potential performance bottleneck introduced by the CE ants system.

The report is organized as follows. Part II describes in brief the theoretical background for the Cross-Entropy based Ant routing algorithm used in the demonstrator. In Part III details of the specification and implementation are given. Part IV describes the scenario used to demonstrate the handling of dynamics in this monitoring system and some simulation results to complement the demonstrator with respect to scalability. The document is closed by some concluding remarks in Part V.

[1] www.hping.org

Part II

Theoretical background - the elite CE ants algorithm

Finding the best path from node s to d is a routing problem in a network, $G = (V, E)$, representing a bidirectional connected graph. The $V = \{v_i\}$ are nodes (vertexes) in G, where v_i is node i. The $E = \{e_{ij}\}$ are the links (edges) in G, where e_{ij} is the link between node i and j, $v_i - v_j$. N_e is the number of links in each node. The $c(e)$ is the cost of link e, and $L(\pi) = \sum_{\forall e \in \pi} c(e)$ is the total cost of path π. The network topology can change and $G(t)$ is the graph at time t. The path $\pi_{t,s}^{(d)}$ is the path (trajectory) found from source node s to destination node d after iteration t. The basic steps are described in more detail in Deliverable 05 [4].

The routing approach is either

- *hop-by-hop routing* (e.g. OSPF, BGP) - the routing information in the intermediate nodes from s to d is only destination specific and contains information about the first hop of the best path from the current intermediate router to the destination d.
- *virtual path* (e.g. MPLS) - the routing information in the intermediate nodes is source and destination specific and contains the information about the best path all the way from s to d.

In the following, the algorithm is described for virtual path management. However, the same principle applies for the hop-by-hop routing as well. The cost of the different paths from s to d changes over time due to network dynamics like topology changes (nodes and links move, (re)appear, and disappear) and changes in traffic pattern.

The algorithm consists basically of:

- *Forward search* - at each node i along the path from the source node, s, to the destination node d choose the next edge e_{ij} at random according to the routing probabilities in node i, $p_{t,ij}^{(d)}, \forall j \in v_i$,
- *Path evaluation* - determine the cost value, $L(\pi_t)$ of the path of iteration t, π_t. Let $c(e)$ be the cost of link e. In this section, $c(e)$ is the delay experienced by the agent traveling over this link, including queuing and processing at the originating end. Hence, the object function used for illustration in this section is the end-to-end delay of the path $\pi_s^{(d)}$ from source node s to destination d.

$$L(\pi) = \sum_{\forall e \in \pi} c(e) \tag{1}$$

- *Backward updates* - return to source node s and update the routing probabilities in each node along the path π_t.

The path evaluation and pheromone updates are guided by Cross Entropy, initially proposed by Rubinstein for rare event theory [12]. In Helvik and Wittner [6] a distributed variant of this is developed and combined with ant algorithm for the primary backup path problem. In order to control and reduce the overhead of this CE ants algorithm, the use of elite selection is proposed in Heegaard, Wittner, Nicola, Helvik [5].

In order to implement a path management strategy based on the elite CE ants approach outlined above, basically four algorithms must be implemented in every node that can be source, destination and/or routing nodes. In this section a brief description of the algorithms are described. A simulator implementing these algorithms are described in Deliverable D06 [2], and examples of use is given in Deliverable D07 [1].

1 Generate ants

An implementation of CE ants must have a function that generates ants with the mission to find the best path to the destination d. Each ant species has a unique task of finding the best path between a specific source and destination pair. A sequence of ants of the same species is sent out from a given source s to a destination d. Each ant species might have individual set of rules (e.g. how to search, how to update) and specific parameters (e.g. memory, ant frequency). When more and more information about paths between s and d is obtained, the frequency of ants submitted can be decreased to reduce the management overhead. In a static network the ant frequency can be set to 0 when the optimal (or at least a good) solution is found. However, under dynamic network conditions and environments the frequency must always be greater than 0 in order to detect network changes. The frequency might be regulated in accordance to the dynamics of the network, i.e. high frequency of (significant) changes will require a high frequency of ants.

The i'th ant of a specific species is submitted from the source node s to the destination d, at time epoch t_i. The ant interarrival time, i.e. the time between epochs $t_{i+1}$and t_i, is either deterministic of following a negative exponential distribution, $f(t) = \lambda e^{-\lambda t}$, where $1/\lambda$ is the expected time between two ants (of the same species).

2 Forward searching ants

An *exploration* ant uses no information in the nodes of the quality of the different alternatives when selecting the next hop. This is simply done by a unguided random walk over the existing alternative outgoing links. A *normal* ant (i.e. exploitation, or also called maintenance ant) will in its forward search use the following routing probability at time t in node i

$$p_{t,ij}^{(l)} = T_{t,ij}^{(l)} / \sum_{\forall k} T_{t,ik}^{(l)}$$

where $T_{t,ij}^{(l)}$ and is the pheromone value at time t over interface j in node i for species l (i.e. for a specific source destination pair).

The forward searching ants will at each node randomly select the next hop by sampling over the outgoing interfaces from this node. The sampling probability is given by the normalized pheromone values updated by the backward ants. Hence, the higher the pheromone value, the more likely it is to select a given interface. A random selection of interface can efficiently be implemented by the following algorithm:

```
Let f_i be pheromone value on interface i,
f_0 = sum_{i=1}^{n} f_i is the sum of pheromones over all interfaces

Sample U = U_01 . f_0, where U_01 is uniformly distributed over [0,1].
Initialize i = 1 and q = f_1:

WHILE U > q DO
   i = i + 1
   q = q + f_i
Returns i - the sampled interface
```

An example of implementation is given in Listing 1 (quasi, Simula-like):

3 Path evaluation

At the destination node the cost value $L(\pi_t)$ of t'th path found, π_t, is calculated. Based on this cost value, a performance function h_t is obtained that includes the cost value and some cost value history. The historical cost values are recorded through an autoregressive formulation with a memory parameter β. This formulation enables a compact representation of previous cost values weighted decreasingly as time goes by and new paths are found. The performance function is (see details in [6]):

$$h_t = \beta h_{t-dt} + (1-\beta)e^{-L(\pi_t)/\gamma_t} \tag{2}$$

The γ_t is the scaling parameter that is the result of the optimization of the change of measure f in the routing probability matrix. Each ant species has a unique scaling parameter. The scaling parameter is referred to as the "temperature", and this parameter is identified as a useful monitoring parameter for the stability of a path. In order to avoid storage of all (or a part) of previous cost values, the γ_t is calculated through a first order Taylor expansion (See [6] and [15] for details):

$$\begin{aligned}
\gamma_t &= \frac{B+L(\pi_t)\cdot\exp(-\frac{L(\pi_t)}{\gamma_{t-\Delta t}})}{(1+\frac{L(\pi_t)}{\gamma_{t-\Delta t}})\exp(-\frac{L(\pi_t)}{\gamma_{t-\Delta t}})+A-\rho\frac{1-\beta^{M+1}}{1-\beta}} \\
A &\leftarrow \beta(A+\left(1+\frac{L(\pi_t)}{\gamma_t}\right)\exp(-\frac{L(\pi_t)}{\gamma_t})) \\
B &\leftarrow \beta(B+L(\pi_t)\exp(-\frac{L(\pi_t)}{\gamma_t})) \\
\gamma_{t-\Delta t} &\leftarrow \gamma_t \\
M &\leftarrow M+1
\end{aligned} \tag{3}$$

where the initial values are $A = B = M = 0$ and $\gamma_0 = -L(\pi_0)/\ln(\rho)$. The node only needs to store γ, A, B, and M instead of the complete observation window. After iteration t, the ants are

Listing 1: Forward ants

```
! *** FORWARD ANTS BY STOCHASTIC ROUTING ***;

! tmptab - contains pheromones;
tmptab(0) := 0;
! tmpI - identity function on every edge: 1=eligable, 0=non-eligable;
tmpI(0):=0;
! run through all edges for the current node;
FOR i:=1 STEP 1 UNTIL eTab(nodeID) DO
BEGIN
  ! revisits to a node is not allowed;
  revisit:=FALSE;
  IF NOT explore THEN
  BEGIN FOR j:=1 STEP 1 UNTIL pck.hop-1 DO
    revisit := revisit OR (pck.path(j) = neigh(i));
  END;

  ! disable selection interface down, or revisit to cell;
  IF < Link down > OR revisit THEN
  BEGIN
    tmpI(i) := 0;
    tmptab(i) := 0;
  END
  ELSE
  BEGIN
    tmpI(i) := 1;
    IF pck.explore THEN
      tmptab(i) := 1.0
    ELSE
      tmptab(i) := metric(tmpflag, pck.token, i);
  END;
  tmptab(0) := tmptab(0) + tmptab(i);
  tmpI(0) := tmpI(0) + tmpI(i);
END;

! tmptab now contains pheromone values, tmptab(0) the sum of these;

! NORMAL CASE: ;
! *** check if packet not too old ***;
IF tmptab(0) > 0.0 AND pck.hop < TTL THEN
! at least one valid next step exits;
BEGIN
  ! sample a value uniformly between 0 and sum of pheromones;
  samp := tmptab(0)*unif01.sample;
  tmp := tmptab(1);
  i:=1;
  ! do while the sampled number is less than acc. sum;
  WHILE tmp<samp AND i<eTab(nodeID) DO
  BEGIN
    i:=i+1;
    tmp := tmp + tmptab(i);
  END;

! => the selected edge / interface is i;

! update cost value;
pck.T := pck.T + cost(i);
```

carrying information about the path found, π_t, information about the observed cost value and historical values.

Only ants with cost values less than the elite limit (calculated as a function of the current temperature, see [5] for further details), are returned to the source and updating the pheromone (routing table) of the intermediate nodes, see Section 4.

A quasi-code (Simula) of an implementation of this is given in Listing 2. The cost value T is in the payload of the backtracking ant (`pck`).

4 Backward updates

The backward agents updates the pheromones T_j (for simplicity the species l, the iteration time t and node i indices are suppressed) and the corresponding $p_j = T_j / \sum_{\forall k} T_k$. The pheromone is given as

$$T_j = I(\{j\} \in \pi_t) e^{-\frac{L(\pi_t)}{\gamma_t}} + A_j + \begin{cases} -\frac{B_j}{\gamma_t} + \frac{C_j}{\gamma_t^2} & \frac{1}{\gamma_t} < \frac{B_j}{2C_j} \\ \frac{-B_j^2}{4C_j} & \text{otherwise} \end{cases} \tag{4}$$

where

$$\begin{aligned} A_j &\leftarrow \beta(A_j + I(\{j\} \in \pi_t) e^{-\frac{L(\pi_t)}{\gamma_t}} (1 + \frac{L(\pi_t)}{\gamma_t}(1 + \frac{L(\pi_t)}{2\gamma_t}))) \\ B_j &\leftarrow \beta(B_j + I(\{j\} \in \pi_t) e^{-\frac{L(\pi_t)}{\gamma_t}} (L(\pi_t) + \frac{L(\pi_t)^2}{2})) \\ C_j &\leftarrow \beta(C_j + I(\{j\} \in \pi_t) e^{-\frac{L(\pi_t)}{\gamma_t}} \frac{L(\pi_t)^2}{2}) \end{aligned}$$

This means that each node must store a set of A_j, B, C_j, one for each destination d and for each outgoing link j. The initial values of (4) are $A_j = B_j = C_j = 0$.

A quasi-code (Simula) of an implementation of this is given in Listing 3. The cost value T is in the payload of the backtracking ant (`pck`).

Listing 2: Path evaluation

```
! *** PACKET ARRIVED AT DESTINATION NODE;
! swap source and destination addresse;

! update score values, the T is in the packet payload;

! CEAnts: calculate score vaule, L(pi)=T;
T := pck.T;

! get temperature over all samples;
! the token is in the packet payload and identifies the ant species;
tmpT := -ga0tot(tmpflag, pck.token)*Ln(rho(pck.token));

! get temperature over elite samples;
tmpTresh := -ga0(tmpflag, pck.token)*Ln(rho(pck.token));

! elite selection: dynamic;
! ie. update the pheromones if T is less than or equal to tmpT;
update := pck.T <= tmpT;

! remember src and dst are swapped;
d := pck.src;
s := pck.dst;
tok := pck.token;

! update temperature for updating ants;
IF pck.hop<TTL AND update THEN
BEGIN
  pck.update := TRUE;
  IF ga0(tok) EQ 0 OR M(tok) EQ 0 THEN
  BEGIN
    ga0(tok) := -T/Ln(rho(pck.token));
  END
  ELSE
  BEGIN
    tmp := T/ga0(tok);
    tmpA := A(tok);
    tmpB := B(tok);
    tmpE := Exp(-tmp);
    ga0(tok) := (tmpB + T*tmpE)/((1+tmp)*tmpE +

      (tmpA-rho(pck.token)*((1-beta**(M(tok)+1))/(1-beta))));
    A(tok) := (tmpA + (1+tmp)*Exp(-tmp))*beta;
    B(tok) := (tmpB + T*Exp(-tmp))*beta;
  END;
  M(tok) := M(tok)+1;

  ! ** EXCEPTION HANDLING, gamma very small or even less than 0;
  IF ga0(tok) < verysmall THEN
  BEGIN
    ga0(tok) := -T/Ln(rho(pck.token));
  END;

  pck.ga0 := ga0(tok);

END;

! update temperature for all ants, same procedure except that ;
! ga0 -> ga0tot; ! M -> Mtot; ! A -> Atot; ! B -> Btot;
```

Listing 3: Backward ant

```
! *** BACKWARD AGENT - UPDATE ROUTING TABLES ***;

! *** CEAnts: use cross entropy approach ***;

! remember that src and dst are swapped;
! *** the ants follows the reverse path;
s:= pck.src;
d:= pck.dst;
tok:= pck.token;
T := pck.T;
tmpga0 := pck.ga0;
FOR i:=1 STEP 1 UNTIL eTab(nodeID) DO
BEGIN
  tmpA := Ars(tok, i);
  tmpB := Brs(tok, i);
  tmpC := Crs(tok, i);
  IF i EQ pck.infpath(pck.hop) THEN
  ! along selected path;
  BEGIN
    tmpE := Exp(-T/tmpga0);
    tmpA := tmpA + tmpE*(1 + T/tmpga0*(1+T/(2*tmpga0)));
    tmpB := tmpB + tmpE*(T + (T**2)/tmpga0);
    tmpC := tmpC + tmpE*(T**2)/2;
  END;

  Ars(tok, i) := tmpA*beta;
  Brs(tok, i) := tmpB*beta;
  Crs(tok, i) := tmpC*beta;
  tmp := tmpA;

  IF tmpB*tmpC*tmpga0 > 0 THEN
  BEGIN
  IF 1/tmpga0 < tmpB/(2*tmpC) THEN
    tmp := tmp + tmpC/tmpga0**2 - tmpB/tmpga0
  ELSE
    tmp := tmp - (tmpB**2)/(4*tmpC);
  END ;

  IF i EQ pck.infpath(pck.hop) THEN
    ! along selected path;
    metric(pck.token, i) := tmpE + tmp
  ELSE
    metric(pck.token, i) := tmp;
END;
```

Part III

AntPing - design and implementation

The AntPing sends monitoring and routing packets from a source to a specific destination N times per second. The main differences between AntPing and ordinary ICMP Ping [11] are:

- AntPing does both routing and monitoring, while Ping only does monitoring
- AntPing is routed according to a stochastic ant routing algorithm, Ping is routed according to link state (e.g. OSPF, ISIS) or distance vector (e.g. BGP) routing algorithm.
- AntPing records forward route (route records). Some implementations of Ping supports the -R flag that records route if supported in the routers.
- In AntPing the datagrams are sent backwards on reversed route (path) using source routing, see Section B.2.
- AntPing adds extra statistics in the routers as implemented by Click. The IP address at every hop is stored in IP header option field [10] and time stamps (e.g. the static link metrics) added to the payload.
- AntPing sends UDP datagrams on destination port 51234, Ping uses ICMP.
- AntPing indicates ant type (normal, exploration, forward, backward, update) in payload
- AntPing supports both deterministic or random interarrival times with adjustable sender rate.
- AntPing collects animation data for demo purpose.

The AntPing requires implementation of a *generator* of ant datagrams (UDP datagrams), a *receiver* of these ant datagrams, and processing support in the routers to *forward* and *update* these ant datagrams. Figure 1 shows an example of how the AntPing works. In the source host S, an ant datagram is generated. The time this datagram was generated is added to the payload, source address is set to S and destination address to D (more details on datagram field settings in the following). At router $R1$, the datagram's destination address is read and the next hop is either selected randomly according to a uniform distribution over all available interfaces (random-walk), or according to the ant routing method presented in Part II. Random-walk routing is applied when the ant type is *exploration*, i.e. when the ants do not smell any pheromone values in its search for new paths. After selected next hop, but before transmitting it, the ant datagram is updated; the host address of the first router visited is recoded and stored in the datagram header (the optional IP header). Then the datagram is forwarded. This is repeated for every router along the path until the host address is equal to the destination address D. At host D, it is checked whether this is the first ant from the source S. If yes, a new connection is established (source port number will be unique for this species). Otherwise, the cost value of the path traveled is calculated, the temperature updated and added to the payload. An update packet is now sent where the source address is D, and the final destination address is S.

In order to update pheromone values according to the total path quality in all intermediate routers between D and S, the AntPing will follow the reversed path back to S and hence the destination address of the first hop is $R3$, the second $R2$, and so on until the next hop is S. This is done by using loose source routing (see Section B.2). This works as follows: In D the reversed route is added to the datagram's header. The first hop, $R3$, is placed in the destination address field, the final destination, S, is placed at the end of the reversed path, and the next hop pointer is pointing at $R2$ (in boldface). At $R3$, the destination address is changed to the value pointed to by next hop pointer, and the pointer is moved to the next element in the reversed path, $R1$. This is repeated for every hop along the reversed path until the destination address is S. When the ant datagram is received by S, essential monitoring information is written to a log, if this is not done in D.

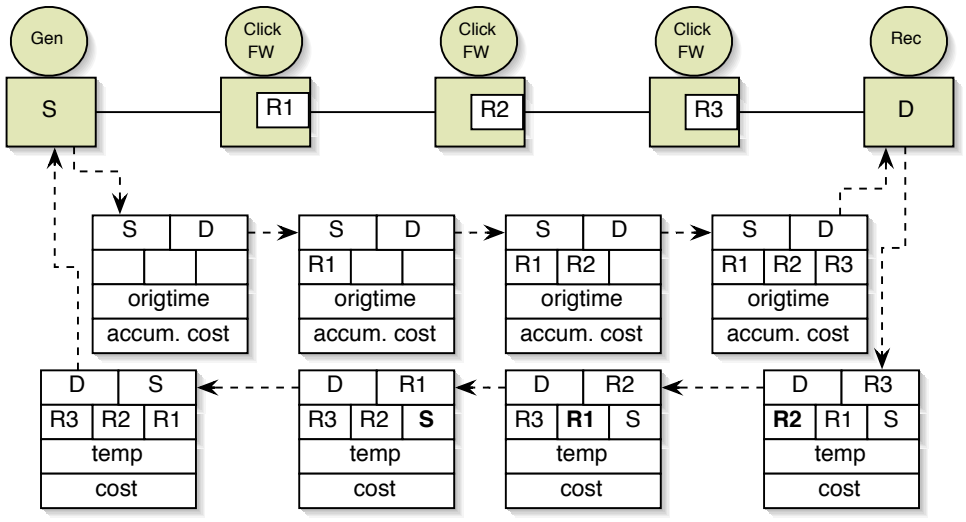


Figure 1: Route record for forward ants and source routing for backward ants.

The sequence chart in Figure 2 gives a more detailed description of the exploration and connection establishment phase and the maintenance and improvement phases.

From this description, the following components of AntPing are identified:

- *AntGen* (the ant nest) - here the ants are generated and received if and when they return to the nest
- *AntRec* (the food source) - the ants that reach the destination will accumulate the cost of the path (trail) and update of the temperature and return to the nest with this information.
- *Router* (the path (trail)) - responsible for routing and forwarding of the ants.

In the following sections these 3 components are described.

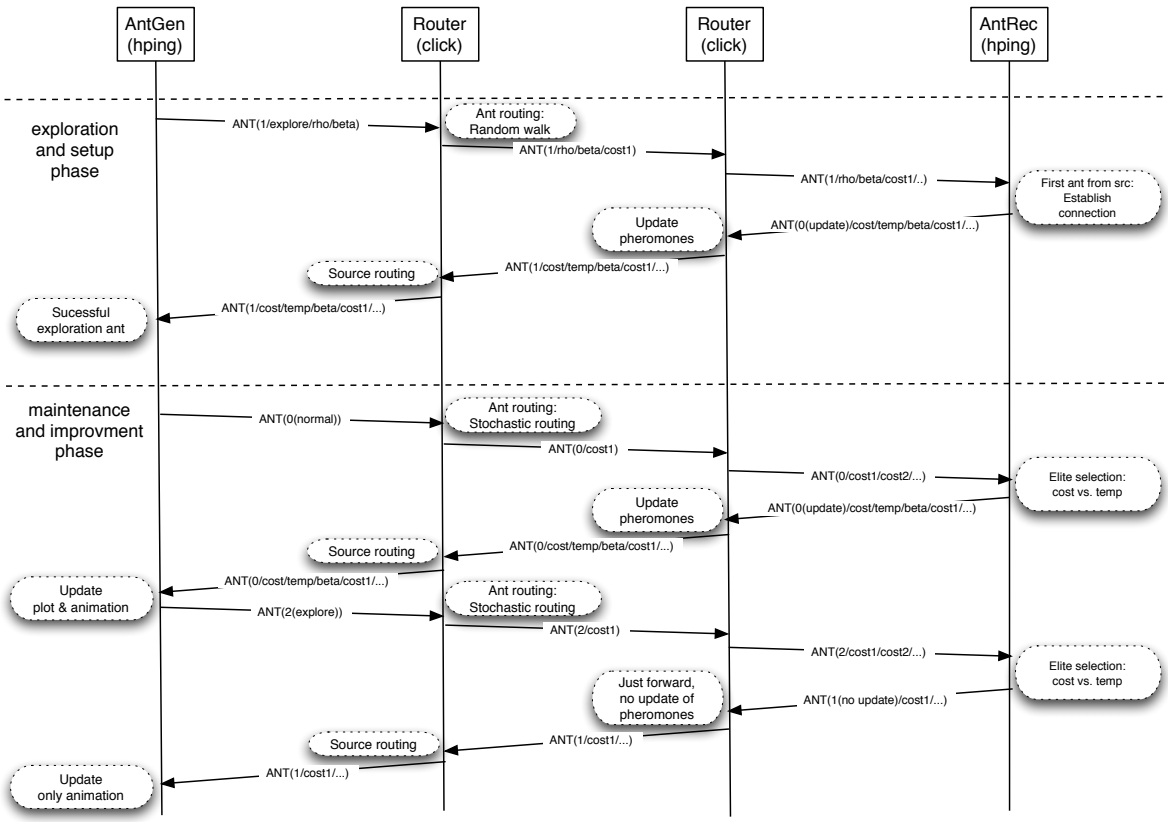


Figure 2: Sequence chart of the AntPing protocol behaviour

5 AntGen - "the ant nest"

The AntGen ("the ant nest") consists of two sub processes; generator of UDP datagrams that represent the ants, and receiver of the UDP datagrams that returns to the AntGen ("the ant nest") with information about the path ("trail") quality to the destination ("food source").

5.1 Methods

The generator creates an UDP datagram, insert the generation time stamp, and sends it over an IP socket to the default gateway. The following methods are required to implement this:

generate_ants(explore, rate, det, portno, destaddr) - sends an explore (if *explore* is TRUE, otherwise a normal (exploitation/maintenance) ant is sent) ant to the destination *destaddr* on port *portno* (51234 is used as an ant datagram identifier) on average *rate* times per second. The interarrival times are deterministic if *det* is TRUE, otherwise it is negative exponential, i.e. the ant generator process is Poisson. The following sub-methods are identified:

CEA = create() - creates a reference to an ant datagram (uses UDP, see Section 8)

set_typeAnt(CEA, value) - set ant type according to Table 1 on page 24.

set_forward_flag(CEA, 51234) - sets forward flag by setting destination port=51234 in the UDP header.

set_address(CEA, address) - sets destination address of CEA to *address*.

set_specid(CEA, specID) - sets a unique ID given by source and destination addresses; the ID is stored in the SRC_PORT field in UDP header.

send_ant(CEA) - sends ant datagram to socket.

The receiver process inside the AntGen needs to recognize the ants generated inside AntGen and extract data regarding the quality of the path, both the total cost and the hop-by-hop cost of every step along the path.

return_ants(CEA, portno, inf) - listen on *portno* (default=51234) at interface *inf*.

read_payload(CEA) - extract the path and cost data from the payload

print_animation_log(CEA) - print data to animate ants, see Section 9 for description of the animation format.

print_log(CEA, temp, cost, rrlist) - print monitoring data to log-file.

print_plot(CEA, temp, cost, rrlist) - print monitoring data to plot-file.

5.2 Implementations

The AntGen is implemented by use of *hping*[2], version 3 with embedded TCL. This enables simple customization of IP packets, e.g. in for the demo this means implementation of sending of UDP datagrams, and receiving IP packets and extracting header fields and payload data.

The script in Listing 4 implements the generating of ants as specified by the methods above (see http://wiki.hping.org/ for a hping and TCL wiki):

Listing 4: Generator in AntPing implemented in hping

```
# Send ants every $freq [ms] to destination as specified in $target
proc genAnts {} {
  global target freq sentind recvind explAnts explRatio antId rho beta delay accTime

  # explore = 0 : maintenace ants and data packets
  # explore = 1 : exploration ants
  # explore = 2 : exploration ants, initialisation phase
  # cost = 0: ant packets
  # cost = 1: data packets

  # the ant is sent as exploration ant at startup and later as maintenance
  # the initial exploration phase is over when the predefined number of ants
  # are received
  if {$recvind <= $explAnts} {
    set explore 2
```

[2]See http://wiki.hping.org/

```
  } else {
    set explore [expr rand() < $explRatio ]
  }

  if { $data == 1 } {
    set explore 0
  }

  # construct packet to be sent to $tagret and with route record
  set pck "ip(daddr=$target,ttl=8)+ip.rr"
  append pck "+udp(sport=$antId,dport=51234)"
  set cost $data

  if { $explore == 2 } {
    # Send initialisation exploration ant to reset
    # the variables on receiver for port number $antId
    append pck "+data(str=$explore/$rho/$beta/$cost)"
  } else {
    # cost values for testing of NAM trace generation
    append pck "+data(str=$explore/$cost)"
  }
  # send packet
  incr sentind 1

  puts "Generated: $pck"
  hping send $pck

  # next packet using "bootstrapping"
  after $freq genAnts
}
```

The send script also contains support for the printing of log files, see Section 9 for animation details and Appendix C.1 for listing of the full "send-nam.htcl" script.

6 AntRec - "the food source"

The AntRec implements the "food source" which is located at the destination node of the virtual path that is to be established. From the sequence chart in Figure 2 it is seen that in the initial phase, exploration ants are sent discovering paths to the destination. The ant receiver listens to port 51234. When the first of these ants arrives, the AntRec process detects that this is a new, unknown, connection (i.e. a new species) and creates an object in the receiver process that contains counter and temperature variables. The ant is returned to the source node with information about the initial animation time counter. The preceding (both exploration and maintenance) ants will update the temperature of this species based on its cost value and the current temperature. The elitism principle, as described in [5], is implemented, meaning that the AntRec checks whether the ant's cost value is almost the best so far before the ant is returned with information of pheromone updates and not just animation data.

6.1 Methods

The following methods are required:

CEA = listen_to_ant_port(portno) - connects to socket and listens on port *portno* assigns pointer to incoming ant datagram.

reply_ants(CEA) - return an ant on reversed route with updated temperature information and forward flag set to FALSE.

cost = calculate_delay(CEA) - reads time stamp from CEA datagram and compares with current system time on receiving host. This option is used in the simulation results in Section 14.

cost1 = calculate_hops(CEA) - alternative 1 cost function: reads the number of hops, i.e. the number of elements in the recorded route list.

cost2 = calculate_cost(CEA) - alternative 2 cost function: reads the accumulated cost in the payload of the packet. This can be the number of hops or sum of interface metrics read and accumulated by the routers. This option is used in the demo trials in Section 14.

specID = read_ID(CEA) - reads species identity from source port field of UDP header, see Section 8.

rrlist = read_recorded_route(CEA) - reads recorded route from optional IP header (see description in Section B.1).

temp = update_temperature(specid, cost) - updates temperature of species *specid* reflecting last *cost* observation.

elite(cost, temp) - determines whether this ant meets the selection criterion according to the elitism principle or not.

add_source_route(CEA, rrlist) - reverses recorded route and adds it to optional header to perform source routing in click as described in Section B.2.

add_temp(CEA, temp) - adds updated temperature temp to CEA.

add_animation_data(CEA, rrlist, rcost) - adds information about the path (*rrlist*) and the cost if each hop along the path (*rcost*)

send_ant(CEA) - sends ant datagram to socket.

6.2 Implementations

The AntRec is also implemented by use of *hping*, see Appendix C.2 for listing of the full "recv-nam.htcl" script.

7 Ant forwarding in Click

The software IP router is implemented based on Click Modular router (Click `http://www.read.cs.ucla.edu/click/`). The router is modified to detect UDP datagrams on destination port 51234 and treat them as ants. The Click router is flow oriented, i.e. the sequence of

operations on the datagram through the router is described. In order to route ants and update the ant routing tables it is necessary to intercept the datagram flow. A few extensions to the existing, and some new Click modules are required. The ants datagrams are treated differently dependent on whether they are forward or backward datagrams.

7.1 Methods

1. Forward ants (dport=51234): read destination address and species identity. They look up the ant routing table and select randomly the next hop. The following AntPing specific methods are required:

   CEA = listen_to_ant(portno) - the router recognizes an ant datagram (portno=51234) and a pointer to the datagram object is assigned, denoted CEA here.

   specID = read_ID(CEA) - reads species identity from source port field of UDP header, see Section 8.

   antType = read_type(CEA) - reads ant type from payload.

   vector = lookupAntTable(specID, antType) - the routing vector for ant species *specID* is assigned vector. The *antType* value (see Table 1) determines whether this is an *exploration* ant (routed according to uniform distribution over the available interfaces (simple random walk)) or a *normal* ant (uses the pheromone values).

   next = rand_getnext(vector) - the next hop is randomly chosen from the probabilities determined by the normalized list in *vector*.

   set_next_addr(CEA, next) - the CEA address is updated.

   add_rraddr(CEA, CURRENT_IP) - adds current IP address to the record route field in the optional IP header (see description in Section B.1).

   cost = read_cost(CEA) - reads ant datagram to next module in click flow.

   update_cost(CEA, cost) - read interface cost and adds to the accumulative cost read from the packet payload and write the new value back to the payload.

2. Backward ants (dport=51234): read destination address and species ID. They update all pheromone values in the ant routing table for this species identity. The backward ants are routed according to source routing information found in the optimal IP header of these ants. The following AntPing specific methods are required:

   CEA = listen_to_ant(portno) - the router recognizes an ant datagram (portno=54321) and a pointer to the datagram object is assigned, denoted CEA here.

   temp = read_temp(CEA) - reads the temperature from the payload of the ant datagram.

   expCost = read_exp_cost(CEA) - reads the exponential value of the cost value of this ant, $e^{-L(\pi)}$.

   specID = read_ID(CEA) - reads species identity from source port field of UDP header, see Section 8.

   antType = read_type(CEA) - reads ant type from payload.

updateAntTable(temp, specID, expCost, antType) - updates the pheromones of the ant routing table if antType is 0 (normal).

next = get_next_addr(CEA) - reads optional IP header and gets next address (and move the header pointer one position).

set_next_addr(CEA, next) - the CEA address is updated.

send_ant(CEA) - sends ant datagram to next module in click flow.

7.2 Implementations

To make the demonstrator portable, the ant routing is implemented on home routers flashed with Linux OS. The demo network described in this document consists of 10 wired routers to represent the core of the Telenor backbone. We have used LinkSys WRT54GS, v4.0. Other home routers do also support the OpenWRT `http://OpenWRT.org/` Linux distribution. The LinkSys WRT54GS, v.4.0 has 16 Mbytes of memory and 4 Mbytes flash. The 4 switch ports are reconfigured to be IP interfaces and the up-link interface is defined for monitoring and configuration of the router. The wireless interface was disabled because the swarm based method was not developed for wireless broadcast routing, but also because it was expected potential radio interference between the 10 routers that will be rather close to each other in the demo setup. In Figure 3 an illustration of the demo setup in our lab is given.

Click (Click `http://www.read.cs.ucla.edu/click/`) is a modular software router developed by MIT LCS's Parallel and Distributed Operating Systems group. A Click router is a collection of modules called elements. The elements control every aspect of the router's behavior, from communicating with devices to packet modification to queuing, dropping policies and packet scheduling. It is easy to write new elements in C++. The router configuration is describing the packet "flow" through the router be gluing the required elements together with a simple language. The AntPing router has extended the click library by 5 new elements:

1. *SourceRoute* - provides support for source routing of IP packets based on Loose Source Routing as described in Appendix B.2. The backward ants is routed according to source routing, see methods *get_next_addr(CEA)* and *set_next_addr(CEA, next)* listed above.

2. *AntTable* - element holding persistent information about the interfaces and the routing, like cost and pheromone values.

3. *BackwardAnt* - updates the pheromones of the ant routing table if antType of the backward ant is 0 (normal), see method *updateAntTable(temp, specID, expCost, antType)* above. All other ants are returned without updating the pheromone values.

4. *AntLookup* - provides support for stochastic routing according either to uniform distribution (exploration ants does random-walk) or normalized pheromone values (normal ants). The ant type is determined according to the settings in Table 1. After the next hop is selected, the record route and cost values are updated for the IP packet implementing the ant. See methods *lookupAntTable(specID, antType)*, *rand_getnext(vector)*, *set_next_addr(CEA, next)*, *add_rraddr(CEA, CURRENT_IP)*, *update_cost(CEA, cost)* as listed above. This element also notices link failure and TTL expiration.

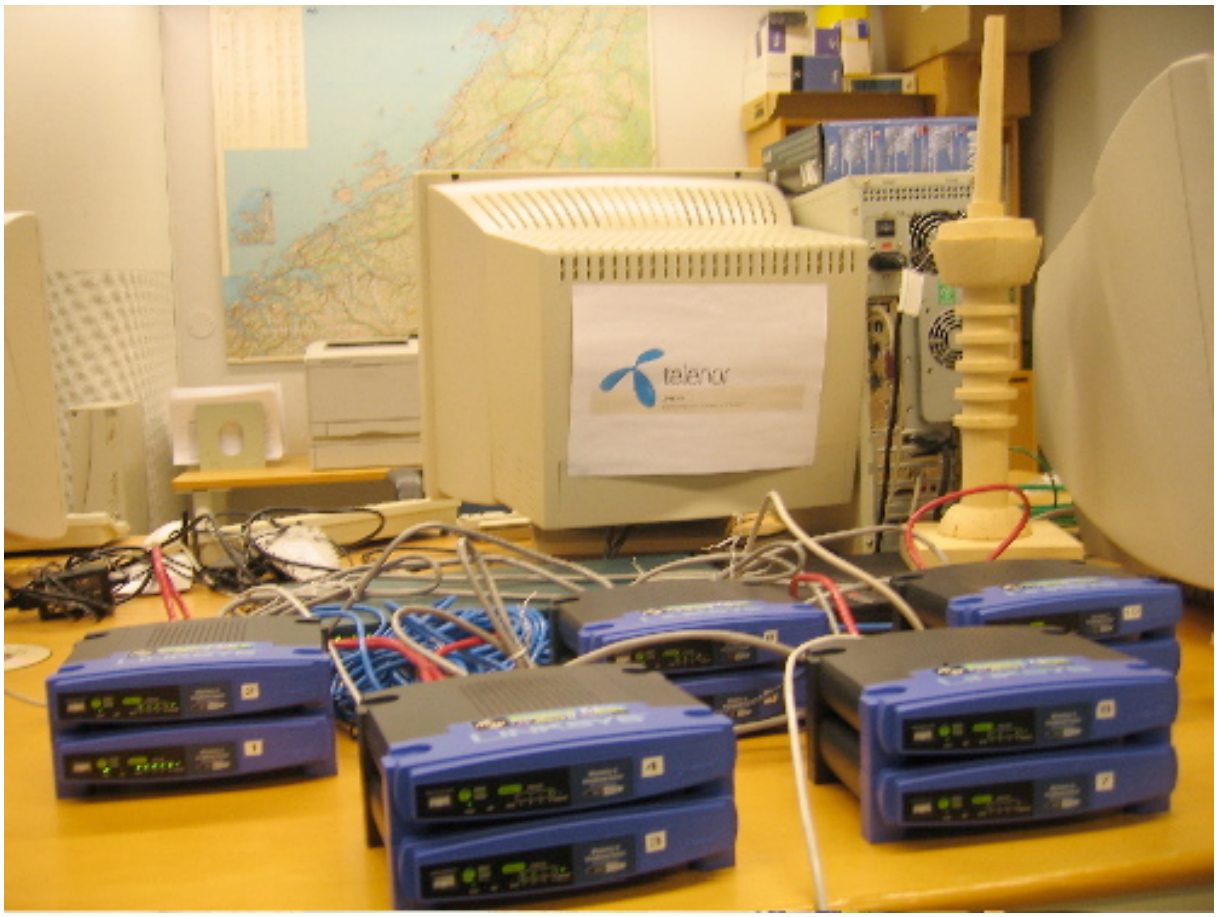


Figure 3: The AntPing installation of the Telenor lab in Trondheim. To the right a wooden copy of the Tyholt tower (TV/radio and a rotating restaurant) just outside our office window.

5. *AntTrafficMonitor* - Measures the byte and packet rate through the router, and updates the AntTable. These measurements can be used as cost values in the CE-ant algorithm, but are currently not in use.

In Figure 4 the configuration of the packet flow for AntPing is illustrated. Packets are read from the network interface in FromDevice. The packet flows from element to element until it reaches ToDevice or ToHost. Each element may alter the content of the packet. If an element has two or more outputs, the content of the packet usually decides which output it is sent to. Most of the elements are necessary for normal IP routing, like ARP handling, broadcast dropping, decreasing TTL, static routing, sending ICMP error messages when necessary etc. In addition we filter out forward- and backward ant packets, and send them to the ForwardAnt and BackwardAnt elements to do the swarm based routing. ForwardAnt decides which output the packet goes to, based on the swarm intelligence, while the BackwardAnt sends all packets to SourceRoute which finds the next hop.

Running Click in home routers implies that we must consider the rather restrictive constraint

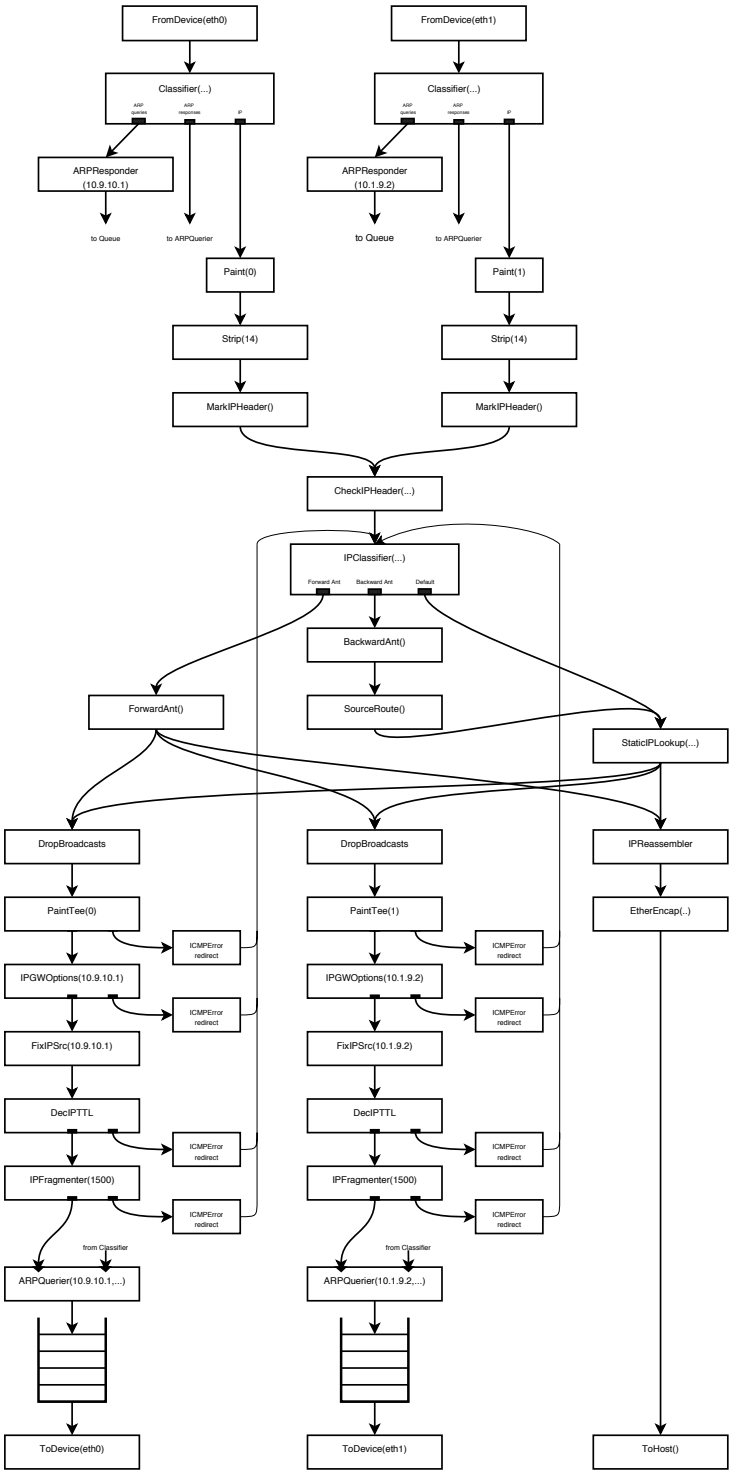


Figure 4: Click module flow chart for AntPing

of 16 Mbytes of RAM and only 4 Mbytes of flash memory. The Click elements are C++ and compiling with C++ standard library will result in code larger than 16 Mbytes. The alternative used in this implementation is *uClibc++* (uClibc++ `http://cxx.uclibc.org/`) which is a library optimized with respect to size rather than performance for embedded systems.

The flashing of LinkSys and installation of OpenWRT Linux distribution on these boxes are described in Appendix E. A brief description of how to configure the network interfaces on the LinkSys switch port is described in Appendix F. Other details of the demo setup procedures, including in installation of and starting click, starting hping on source and destination nodes, and running live animation and cost plotting, are given in Appendix D.

8 Ant datagram format

The ants are implemented by use of UDP datagram. In Figures 5 and 6 the IP- and UDP-headers are shown, as well as the payload that contains information to ensure correct operation of the AntPing. The total size of the packet is less than 100 bytes. The protocol type is UDP (17 according to [9]).

All ants are UDP datagrams identified by destination (forward) and source (backward) port equal to 51234. The ant species are distinguished by unique source port number (greater than 50000) given by the unique source and destination addresses.

Table 1: antType encoding

Update?	Init?	Expl?	Backw?	sport	dport	antType	Comments on ant type
yes	no	no	no	5000*	51234	0	maintenance and improvement ants
yes	yes	yes	no	5000*	51234	1	the exploration ants in the initial phase
yes	no	yes	no	5000*	51234	2	explore the networks in stable phase
yes	no	no	yes	51234	5000*	0	updates pheromones along reversed path
no	no	no	yes	51234	5000*	1	no updates, sends animation data “home”

The first 2 bits of the payload are used for indication of whether this is an explore or normal (exploitation/maintenance) ant (Figure 5). Forward and backward ants are distinguished by dport=51234 and sport=51234, respectively. See Table 1 for details of the antType and port encoding. The UDP source port contains the ant species identity (e.g. given by the IP source and destination addresses).

The payload format depends on the ant’s role (distinguished by antType values and source and destination ports):

- Initialization exploration ants (dport=51234 and antType=1):
  - `payload = 1/`ρ`/`β (search focus (ρ) and memory parameters (β), added to the payload by the AntGen generator process)
- Normal (maintenance and improvement) ants (dport=51234 and antType=0):
  - `payload = 0/cost1/cost2/.../cost`N (cost value for each of the N hops, added to the payload by the IP router)
- Exploration ants (dport=51234 and antType=2):
  - `payload = 2/cost1/cost2/.../cost`N
- Pheromone update ants (sport=51234 and antType=0):
  - `payload = 0/`$e^{-L(\pi)/\gamma}$`/`$L(\pi)$`/`γ`/`β`/NAMtime/cost1/cost2/.../cost`N (the exponential value is added to avoid problems with float number manipulations in the IP router, see Section 7.2. This, and total cost ($L(\pi)$), current temperature (γ), memory parameter (β), current animation time, and the cost vector are added by the AntRec process).
- Animation update ants (sport=51234 and antType=1):
  - `payload = 1/`$e^{-L(\pi)/\gamma}$`/`$L(\pi)$`/`γ`/`β`/NAMtime/cost1/cost2/.../cost`N

As described in previous sections, the optional IP header is non-empty because this is used for recording route by forward ants, and for strict source routing by backward ants. According to RFC791 [10], the maximum optional IP header is 39 bytes. This implies that when storing 4 bytes per hop, at most 9 hops can be recorded and given in the source route. However, since the demo implements the parsing of ant datagrams we are in the position to extend this limit, if necessary. For the demo the upper limit of 9 hops is sufficient. For larger scale implementation, the AntPing could be implemented using IPv6 where this header limitation do not exist.

9 Animations

To visualize the ant movements and the network dynamics, the AntPing demo has included support for animation through the Network Animator (NAM)[3] which is in widespread use as visualization tool together with Network Simulator (ns)[4]. The demonstrator visualizes discrete events in the ant routing used for detecting, establishing and maintaining virtual paths in the demo network described in the next section.

The NAM format consists of a static part where the topology is described:

[3]See http://www.isi.edu/nsnam/nam/
[4]See http://www.isi.edu/nsnam/ns/

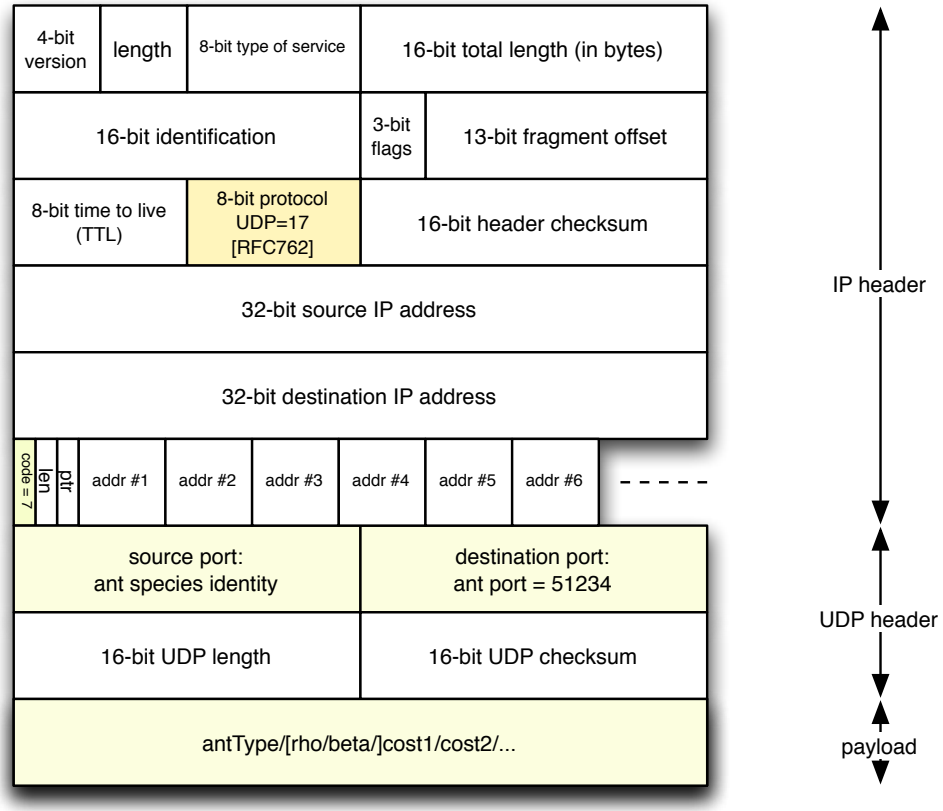


Figure 5: Format of forward ant datagrams with details in IP and UDP headers

- The nodes given by their identity.
- The links specified by the end node identities. The node positions are given relatively by the length of the link and its angle.
- The delay and capacity of the links.

The dynamic part of the NAM trace defines the events. In the demo, the following event are visualized

- ant leaving a node - start of transmission over a link, speed given by link delay, and size of ant is given by observed size of UDP datagram relative to given link capacity
- ant arrives at a node - end of transmission over a link, speed and sizes as above.
- ant dropped at a node - packet dropped from the node and downwards to the bottom border of the window.
- link fails - the color of the link is changed to red
- link restored - the color of the link is changed (back) to black

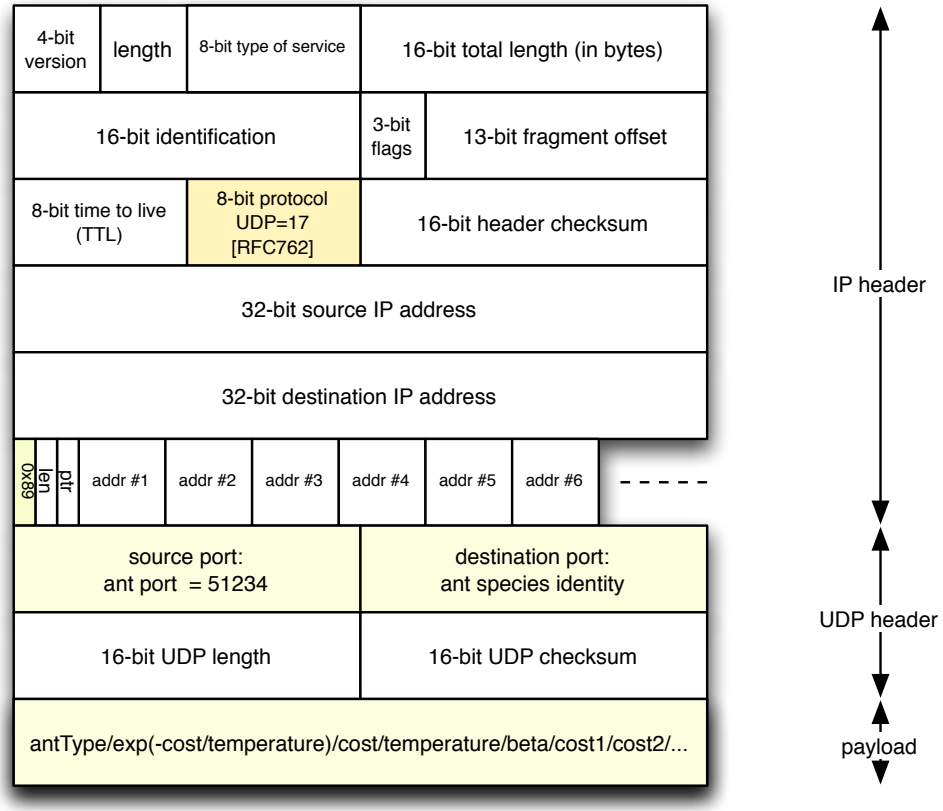


Figure 6: Format of backwards ant datagrams with details in IP and UDP headers

- new connection detected at AntRec - a box (an agent) appears next to the destination node

To log the events listed above, only three nam entry types are required:

- "h" - packet is sent on link (leaving the node), add to log file: `h -t <time> -s <from node> -d <to node> -a <color>`
- "d" - packet is dropped on link, add to log file: `d -t <time> -s <from node> -d <to node> -a <color>`
- "l" - change link status, add to log file: `l -t <time> -S UP | DOWN`

The AntGen process instance writes the NAM trace to a file. The nam process instance reads this from stdin and presents this "live" as the running AntGen process instance continuously updates the dynamic part of the trace with new events. To give an impression of this, a snapshot is include in Figure 7 where the source (AntGen) is connected to node 8 and the destination (AntRec) to node 3. The NAM is run "live" with initial step of 50ms by the following command

```
%> tail -f $namtracefile | nam -r 50ms -
```

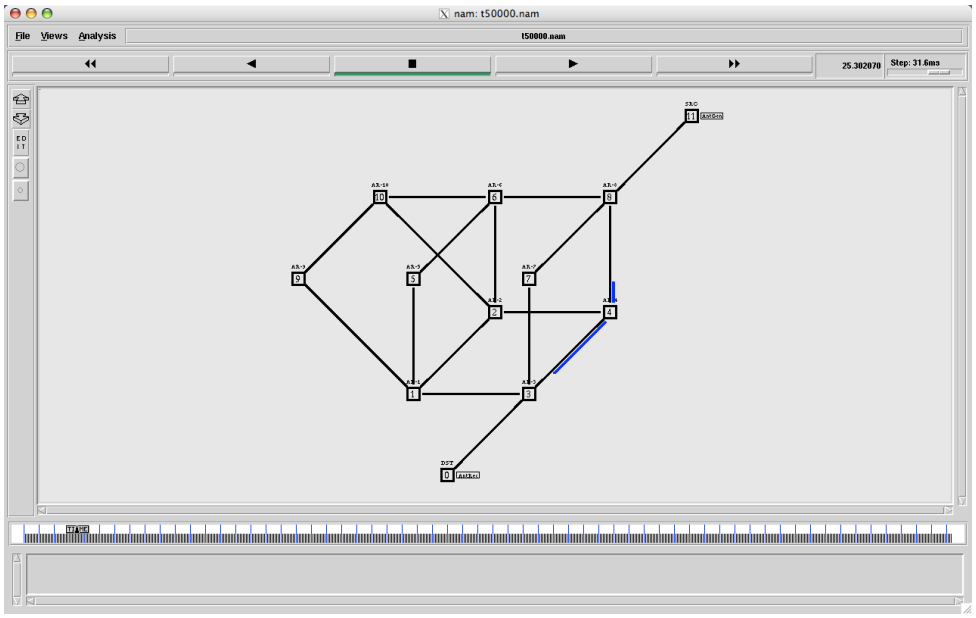


Figure 7: Snapshot of NAM live animation. In this snapshot example the source is connected to node 8, and destination to node 3.

Part IV

Description of scenario

As described in previous section, the demonstrator implements the AntPing on Click modular software routers software. See Section 7.2 for more details. To make the demo portable, it is chosen to run the implementation on small home routers, LinkSys WRT54GS (version 4.0), see Figure 22 for a picture of the setup. Live monitoring data from the demo trials are logged to files and continuously presented in graphs using *gnuplot*[5], while animation of the ants movements and link status are presented by use of *NAM (network animator)*[6], see Section 9 for more details.

This section describes the topology, delivered service, and the network dynamics scenarios defined for the demo. Some observations from the demo trials and supplementary simulations are also included.

10 Topology details and implementation

The topology of the demo is similar to the core topology of *t.net*, the former IP platform of Telenor. It is a 10 node network with 15 edges, i.e. out-degree of 3 edges per node. All links are

[5]http://www.gnuplot.info/
[6]http://www.isi.edu/nsnam/nam/index.html

bidirectional with the same cost and capacity in both directions. An illustration of the topology is given in Figure 8.

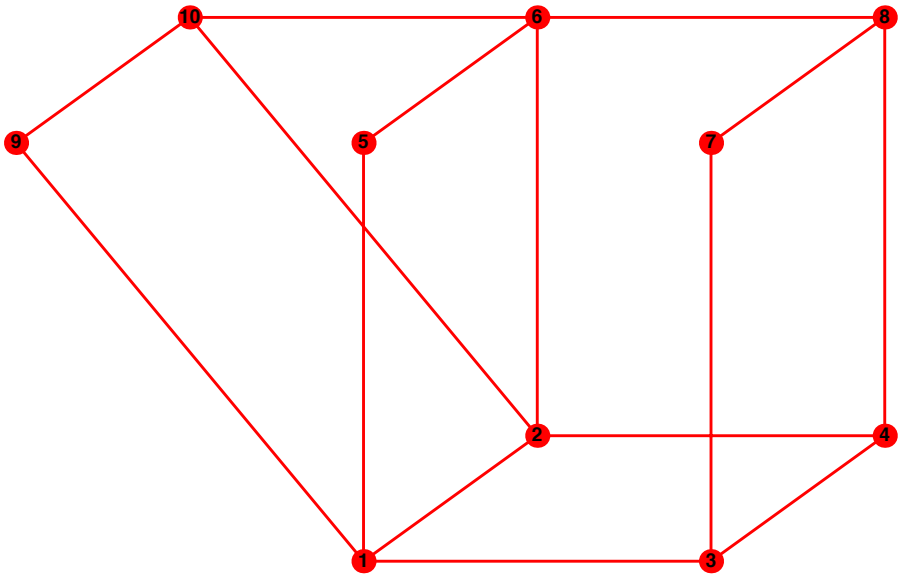


Figure 8: Demo topology inspired by t.net core topology

The physical implementation and sub-netting in this topology is illustrated in Figure 9. Each link and the interface at either end constitutes a subnet. The assigned subnet numbers are 10.B.C.0/30, where B and C refer to the router identity number of the endpoints of the link. The router entity number is from 1 to 10 prefixed by AR-, see Figure 9. The interface at the endpoint with the lowest cardinal number, is assigned *.*.*.1, while the interface at the other end is *.*.*.2. E.g. the interface in AR-3 connecting AR-3 and AR-4, is given the number 10.3.4.1, while the corresponding interface at AR-4 is 10.3.4.2. They both belong to subnet 10.3.4.0/30[7].

Using shortest path routing, the complete static routing table of the topology in Figure 9 is given in Table 2. Observe that only the B.C.D of the 10.B.C.D address is given. In Tables 3 and 4 the minimum end-to-end cost and delay for each node pair are listed. The cost and delay of each link in the network are assumed symmetric, see the values in Figure 9.

11 Monitoring quality of virtual connections

Monitoring in the demo means monitoring of the quality of a specific service that is delivered. The *service* of the demo is establishment and management of virtual connections between a specific set of end-nodes. In the current demo, the *quality* of this service is estimated by the end-to-end delay[8]. Monitoring of the quality of this service is essential in the establishment and

[7]Mask 30 means a subnet with addresses in the range 10.3.4.0 to 10.3.4.3 where .0 is the network address and .3 is the broadcast. address. This leaves only 2 host addresses, one for each end.

[8]In the demo the delay per link is estimated from predefined metrics specified for each interface.

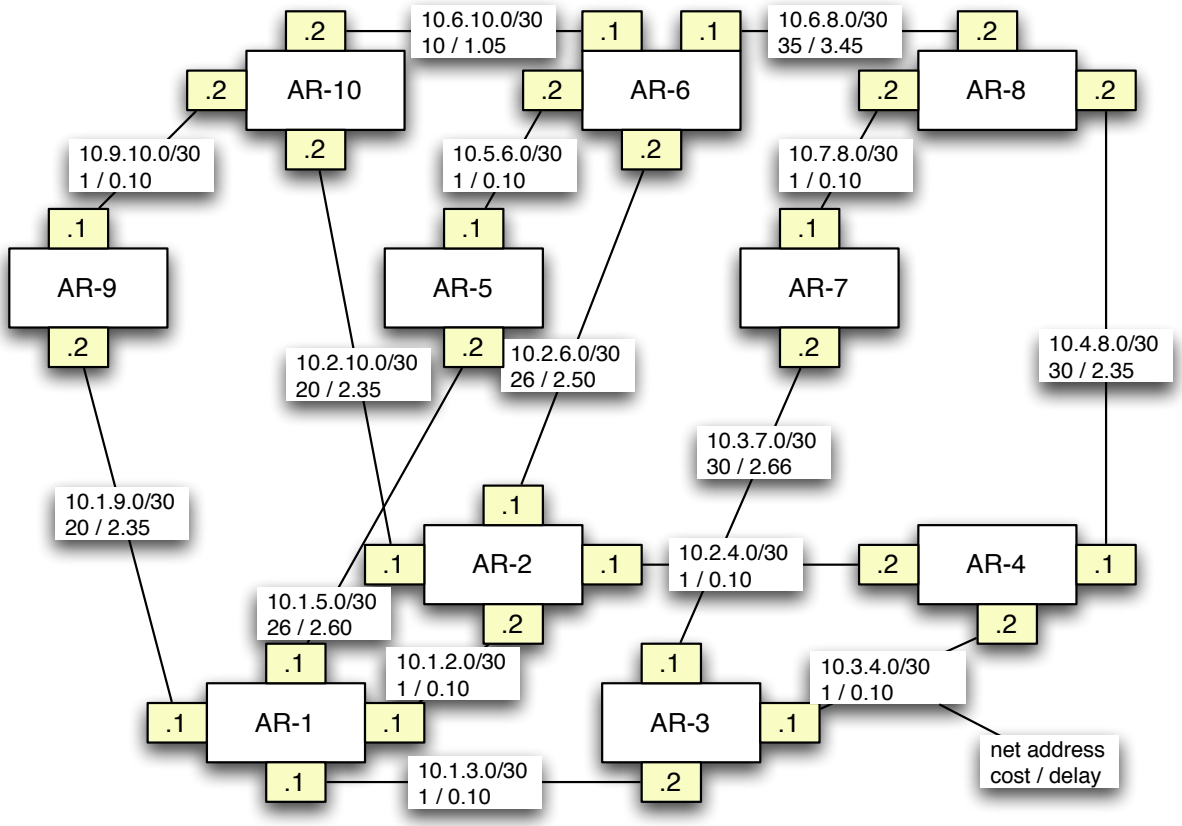


Figure 9: The physical topology of the demo

Table 2: Static routing (next hop) in demo network (only B.C.D of 10.B.C.D is given)

from \ to	AR-1	AR-2	AR-3	AR-4	AR-5	AR-6	AR-7	AR-8	AR-9	AR-10
AR-1	-	1.2.1	1.3.1	1.2.1	1.5.1	1.2.1	1.3.1	1.2.1	1.9.1	1.2.1
AR-2	1.2.2	-	2.4.1	2.4.1	2.6.1	2.6.1	2.4.1	2.4.1	2.10.1	2.10.1
AR-3	1.3.2	3.4.1	-	3.4.1	1.3.2	3.4.1	3.7.1	3.4.1	1.3.2	3.4.1
AR-4	3.4.2	2.4.2	3.4.2	-	4.8.1	4.8.1	4.8.1	4.8.1	3.4.2	2.4.2
AR-5	1.5.2	5.6.1	1.5.2	5.6.1	-	5.6.1	5.6.1	5.6.1	1.5.2	5.6.1
AR-6	5.6.2	2.6.2	6.8.1	6.8.1	5.6.2	-	6.8.1	6.8.1	6.10.1	6.10.1
AR-7	3.7.2	7.8.1	3.7.2	7.8.1	7.8.1	7.8.1	-	7.8.1	3.7.2	7.8.1
AR-8	6.8.2	6.8.2	7.8.2	4.8.2	6.8.2	6.8.2	7.8.2	-	6.8.2	6.8.2
AR-9	1.9.2	9.10.1	1.9.2	9.10.1	1.9.2	9.10.1	1.9.2	9.10.1	-	9.10.1
AR-10	9.10.2	2.10.2	9.10.2	2.10.2	6.10.2	6.10.2	6.10.2	6.10.2	9.10.2	-

management of virtual connections. Changes in quality will be detected as events presented in the animation, and traced and visualized through the plotting functions in the demo.

Table 3: The end-to-end cost in demo network.

from \ to	AR-1	AR-2	AR-3	AR-4	AR-5	AR-6	AR-7	AR-8	AR-9	AR-10
AR-1	0	1	1	2	26	27	31	32	20	21
AR-2	1	0	2	1	27	26	32	31	21	20
AR-3	1	2	0	1	27	28	30	31	21	22
AR-4	2	1	1	0	28	27	31	30	22	21
AR-5	26	27	27	28	0	1	38	37	12	11
AR-6	27	26	28	27	1	0	37	36	11	10
AR-7	31	32	30	31	38	37	0	1	48	47
AR-8	32	31	31	30	37	36	1	0	47	46
AR-9	20	21	21	22	12	11	48	47	0	1
AR-10	21	20	22	21	11	10	47	46	1	0

Table 4: The end-to-end delay [ms] in demo network.

from \ to	AR-1	AR-2	AR-3	AR-4	AR-5	AR-6	AR-7	AR-8	AR-9	AR-10
AR-1	0	0.1	0.1	0.2	2.6	2.6	2.65	2.55	2.35	2.45
AR-2	0.1	0	0.2	0.1	2.6	2.5	2.55	2.45	2.45	2.35
AR-3	0.1	0.2	0	0.1	2.7	2.7	2.55	2.45	2.45	2.55
AR-4	0.2	0.1	0.1	0	2.7	2.6	2.45	2.35	2.55	2.45
AR-5	2.6	2.6	2.7	2.7	0	0.1	3.65	3.55	1.25	1.15
AR-6	2.6	2.5	2.7	2.6	0.1	0	3.55	3.45	1.15	1.05
AR-7	2.65	2.55	2.55	2.45	3.65	3.55	0	0.1	4.7	4.6
AR-8	2.55	2.45	2.45	2.35	3.55	3.45	0.1	0	4.6	4.5
AR-9	2.35	2.45	2.45	2.55	1.25	1.15	4.7	4.6	0	0.1
AR-10	2.45	2.35	2.55	2.45	1.15	1.05	4.6	4.5	0.1	0

12 Monitoring indices

A study of potential candidates for monitoring indices in an ant-based routing system [4] has shown that the most promising with respect to detect significant changes in the network conditions are:

- convergence index (temperature, or the elite limit that is a function of this)
- cost value index (path delay, or loss ratio, available bandwidth)
- pheromone values (in nodes)

A summary of the study and how to apply the results is given in Table 5.

Table 5: Examples of use of CAS indices (from[4])

Metric	Observations	"Health"	Alarms
Ant route table	Deviation from data routing table	Misconfiguration in routing, interface overload	Significant deviation (in time or space)
Pheromone values	Increase by $x\%$ in n sec.	Node/link/path down	Check configuration
Convergence index	Decrease by $x\%$ in n sec.	New node/link/path discovered	A lower delay path for the VC exists
Cost value index	Average over n sec. decreased by $x\%$ last minute	Aftereffect of change in network (still exploration)	None
Path probability	Close to max. for last minute	Stable network	None

In the demonstrator, the cost (delay) of a virtual connection is monitored. The cost value, convergence index (temperature), and pheromone values are plotted. Changes in traffic load and topology are introduced to illustrate how this affects the plotted values.

The results presented in this documents are based on the running demo network, and some supplementary simulations of this, and some simulations of a larger network, see Section 14.

13 Network dynamics in the demonstrator

The demonstrator starts with a fully operational topology as described above and with no virtual connection (VC) established. In the simulations, the VCs between node 1 and all other nodes $(2, \cdots, 10)$ are established and monitored. All references in the following to "preferred path of a VC" means a path connecting the end-points constituting an VC with the current lowest (or amongst the lowest) end-to-end delays. The scenario consists of the following phases (the focus is on VC7):

- *Phase 1*, time [0, 10] : Initialization, no connections are established, the ants are exploring the network to find the best quality virtual connections (VC).
- *Phase 2*, time [10, 50]: Stable phase, the VC(s) has converged and is monitored. It is expected that Ping (following routes given by link state (LS) routing) and AntPing find the same connections and observe the same delay. If the routing metrics (the static cost values for the LS routing) are not correctly reflecting the link delays the Ping will observe larger delays than AntPing. This will demonstrate that AntPing is delay sensitive.
- *Phase 3*, time [50, 100]: Link [4,8] increases its delay from 2.35 ms to 4.00 ms to emulate increased traffic. It is expected that the preferred route from node 1 to 8 is changed from {1,3,4,8} (or {1,2,4,8}) to {1,3,7,8} (see the delay estimates in Table 6 and 7 for VC7). This

increase in delay will not affect the LS routing because the cost values are not delay sensitive. This demonstrates AntPing's ability to detect increases in a bottleneck on a preferred path for a VC, and shortly after propose an alternative path for the same VC.

- *Phase 4*, time [100, 150]: Link [2,4] is down. This will not affect the path preferred by AntPing because it follows {1,3,7,8}. However, the LS routing will now recalculate the shortest path routing tables, and the Ping packets will now be routed either along {1,3,4,8} or {1,3,7,8}. Hence, in this case the link failure will result in decreased observed Ping delays .
- *Phase 5*, time [150, 200]: Link [1,3] is down. This will affect both the AntPing and the LS/Ping. The preferred route is now {1,5,6,8} with a significant increase in cost and delay. This is expected to be detected by both AntPing and LS/Ping.
- *Phase 6*, time [200, 250]: Link [2,4] is up. This will affect both the AntPing and the LS/Ping. The minimum delay route is now {1,2,4,3,7,8} with a significantly decrease in cost and delay compared to Phase 5. This is expected to be detected by both AntPing and LS/Ping.

The demo trials open for dynamics introduced by the audience where they are allowed to unplug and plug cables in the demo network.

The optimal paths, and the corresponding minimum cost, and end-to-end delay for each of the VC between node 1 and all other node in Phase 2 are given in Table 6. In the simulator all VC are established. The effects of the changes for the other VCs can be determined from the Tables 6 to 10.

Table 6: Shortest path in original system from Figure 8

VC	src	dst	primary path (nodes)	cost	delay [ms]
1	1	2	{1, 2 }	1	0.1
2	1	3	{1, 3}	1	0.1
3	1	4	{1, 2, 4}, {1, 3, 4}	2	0.2
4	1	5	{1, 5}	26	2.6
5	1	6	{1, 2, 6}	27	2.7
6	1	7	{1, 3, 7}	31	(2.76)
			{1, 3, 4, 8, 7}, {1, 2, 4, 8, 7}	(33)	2.65
7	**1**	**8**	**{1, 3, 4, 8}, {1, 2, 4, 8}**	**32**	**2.55**
8	1	9	{1, 9}	20	2.35
9	1	10	{1, 2, 10}, {1, 9, 10}	21	2.45

In Phase 3, the delay on link [4,8] is increased. The new optimal solution and the corresponding cost values and delays are listed in Table 7.

In Phase 4 link [2,4] is down. The new optimal solution and the corresponding cost values and delays are listed in Table 8.

In Phase 5 link [1,3] is down. The new optimal solution and the corresponding cost values and delays are listed in Table 9.

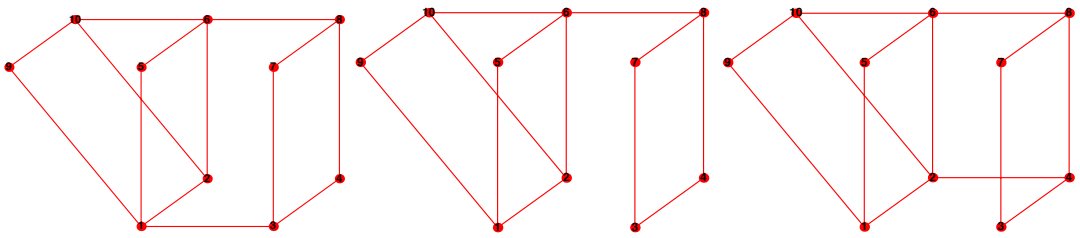
Table 7: Shortest path in Phase 4 (changes indicated by *)

VC	src	dst	primary path (nodes)	cost	delay [ms]
1	1	2	{1, 2 }	1	0.1
2	1	3	{1, 3}	1	0.1
3	1	4	{1, 2, 4}, {1, 3, 4}	2	0.2
4	1	5	{1, 5}	26	2.6
5	1	6	{1, 2, 6}	27	2.6
6*	1	7	{1, 3, 7}	31	2.76
7*	**1**	**8**	**{1, 3, 4, 8}, {1, 2, 4, 8}**	**32**	**(4.2)**
			{1,3,7,8}	**32**	**2.86**
8	1	9	{1, 9}	20	2.35
9	1	10	{1, 2, 10}, {1, 9, 10}	21	2.45

Table 8: Shortest path in Phase 4 (changes indicated by *)

VC	src	dst	primary path (nodes)	cost	delay [ms]
1	1	2	{1, 2 }	1	0.1
2	1	3	{1, 3}	1	0.1
3*	1	4	{1, 3, 4}	2	0.2
4	1	5	{1, 5}	26	2.6
5	1	6	{1, 2, 6}	27	2.7
6	1	7	{1, 3, 7}	31	2.76
7*	**1**	**8**	**{1, 3, 4, 8}**	**32**	**(4.2)**
			{1, 3, 7, 8}	**32**	**2.86**
8	1	9	{1, 9}	20	2.35
9	1	10	{1, 2, 10}, {1, 9, 10}	21	2.45

In Phase 6 link [2,4] is up again. The new optimal solution and the corresponding cost values and delays are listed in Table 10.



(a) Phase 4 (b) Phase 5 (c) Phase 6

Figure 10: Demo topology changes

Table 9: Shortest path in Phase 5 (changes indicated by *)

VC	src	dst	primary path (nodes)	cost	delay [ms]
1	1	2	{1, 2 }	1	0.1
2*	1	3	{1, 2, 6, 8, 7, 3}	93	8.81
			{1, 2, 6, 8, 4, 3}	93	(10.15)
3*	1	4	{1, 2, 6, 8, 4}	92	(10.05)
			{1, 2, 6, 8, 7, 3, 4}	(94)	8.91
4	1	5	{1, 5}	26	2.6
5	1	6	{1, 2, 6}	27	2.6
6*	1	7	{1, 2, 6, 8, 7}	63	6.15
7*	**1**	**8**	**{1, 2, 6, 8}**	**62**	**6.05**
8	1	9	{1, 9}	20	2.35
9	1	10	{1, 2, 10}, {1, 9, 10}	21	2.45

Table 10: Shortest path in Phase 6 (changes indicated by *)

VC	src	dst	primary path (nodes)	cost	delay [ms]
1	1	2	{1, 2 }	1	0.1
2*	1	3	{1, 2, 4, 3}	3	0.3
3*	1	4	{1, 2, 4}	2	0.2
4	1	5	{1, 5}	26	2.6
5	1	6	{1, 2, 6}	27	2.6
6*	1	7	{1, 2, 4, 3, 7}	33	2.96
7*	**1**	**8**	**{1, 2, 4, 8}**	**32**	**(4.2)**
			{1, 2, 4, 3, 7, 8}	**(34)**	**3.06**
8	1	9	{1, 9}	20	2.35
9	1	10	{1, 2, 10}, {1, 9, 10}	21	2.45

14 Observations from demo trials and simulations

The experiments reported in this document are both from the demo trial using the *LinkSys* equipment, and from simulations using an *ns-2* implementation of the AntPing method. In both cases the quality (measured as delay) for a set of virtual connection (VC) is monitored. The results given in this section is related to the monitoring of this quality, and the overhead introduced by means of AntPing. The AntPing is compared to the current practice of using Ping and traceroute to monitor virtual routes and paths in an interdomain network typically routed by *OSPF* (link state routing protocol).

14.1 Demo trials

The *LinkSys* demo implements establishment and maintenance of virtual connection between node 3 and node 8 or node 10. It is possible to change the end nodes to another pair of nodes, but more equipment (more laptops) are required to monitor more than one VC at the time. The dynamics are up to the audience in the sense that they are allowed to unplug and plug the cables. Changes in interface capacity and delay are also possible. Adding new interfaces or links that are not predefined are not possible because a discovery protocol are not yet implemented. Further more, dual stack routing logic is not implemented, and therefor it is not possible to run AntPing and Ping over OSPF in parallel.

14.2 Simulations of demo topology

To supplement the *LinkSys* demo with respect to multi-VC behaviour, the co-existence with Ping over OSPF, observing the dynamics in pheromone values in each node, and study the behaviour in larger networks, a simulator is implemented in *ns-2*[9]. In this section a few results are presented simulating the same dynamics as described in previous section. The simulator has both AntPing and Ping, and are setting up VCs between node 1 and all other nodes in the network. *ns-2* has implemented a Link state routing protocol that emulates the "OSPF" routing behaviour in an interdomain.

In the following, some of the results are presented and observations given.

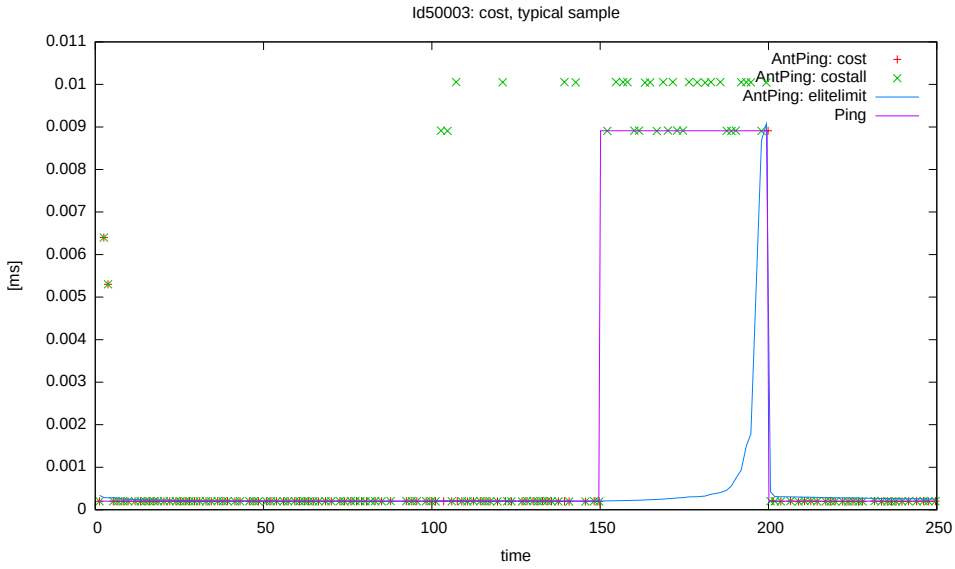
Time plot of cost and elite limit values. Changes in the network topology, and in cost or delay on links, are easily observed both by AntPing and Ping. In Figure 17 the time plots for the cost values are given for VC3 (from node 1 to 4) and VC7 (from node 1 to 8). Both plots include the observed cost values for the ants that returns and update the pheromone values (AntPing: cost), the cost values for *all* ants reaching the destination (AntPing: costall), the elite limit that determines whether the VC should be updated or not (AntPing: elitelimit), and finally the one-way delay from the source to the destination observed by Ping packets.

From the changes in observed delay it is evident e.g. that at time 50 a change occurs that affects VC7 and not VC3. At that point in time the change in delay of AntPing and Ping are different which implies that they follow different paths. The reason in this case is that the link state routing uses static cost values, while AntPing is sensitive to changes in link delays. Hence, if the cost metrics are not consistently set to reflect the (expected or observed) delays, the routing of AntPing and Ping might end up following different routes. The elite limit is a function of the temperature in the CEants method. This is averaging the cost values and might serve as a monitoring indices, in particularly when each VC have many alternative cost values.

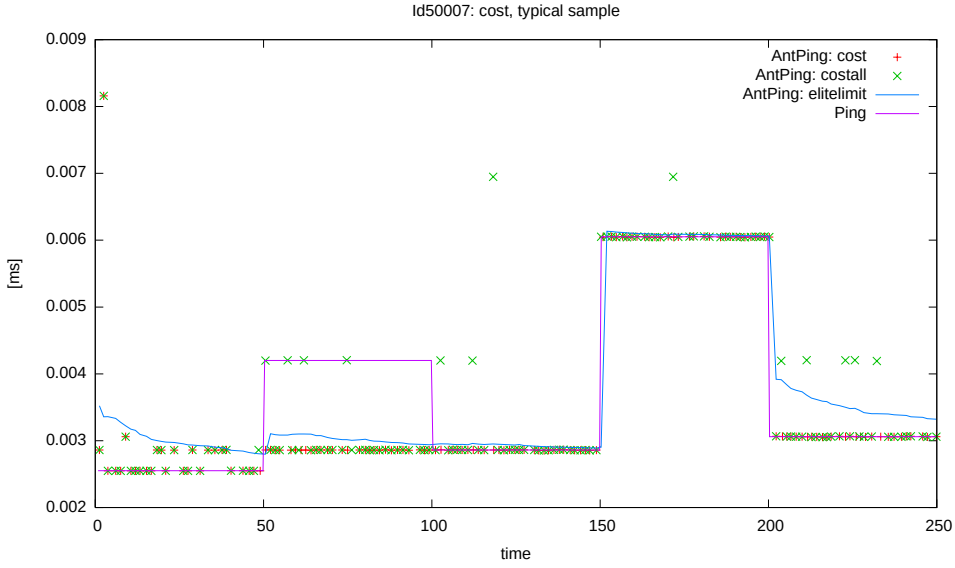
In Figure 12 the average cost values over 25 replications are plotted together with the observed Ping delays. The plot includes the average value and the 95% confidence limits assuming independent replications.

It is observed very little variance in the stable phases, except when link [2,4] is restored. The variance here is due to the fact that in the transient period just after the link is restored, AntPing

[9] http://nsnam.isi.edu/nsnam/index.php/Main_Page



(a) VC3



(b) VC7

Figure 11: Cost and elite limit sample for typical time series for both VC3 and VC7.

alternates between two paths with different cost values, see Figure 11 for a sample from a single simulation.

Time plot of pheromone values of selected nodes and VCs. As an alternative to observing the VC quality in the end systems (in source and/or destination nodes) some information and

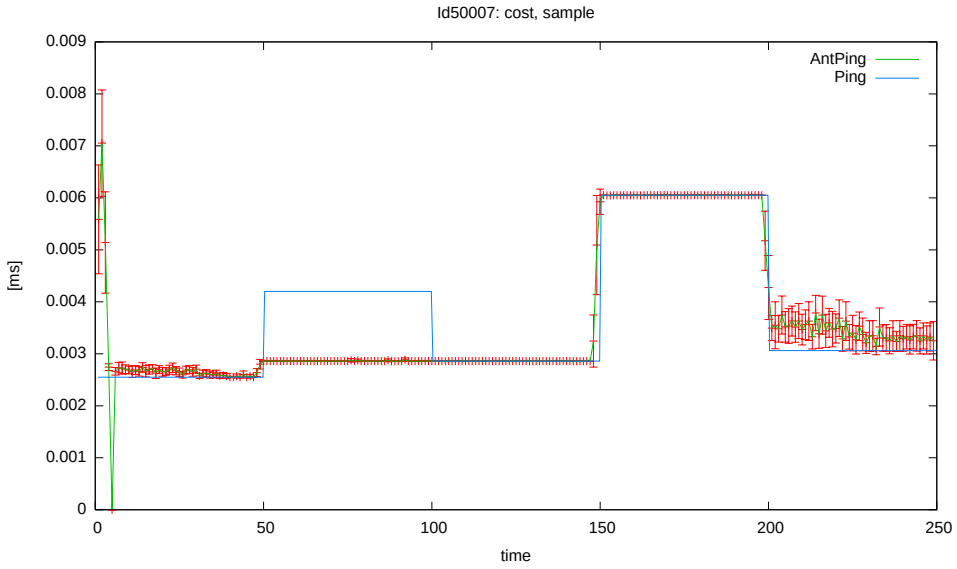


Figure 12: Average cost values (25 replications) for VC7. Delay observed for both Ping and AntPing are plotted.

indications of changes can be obtained by observing the pheromone values of the *CE ants* in the intermediate nodes. In Figure 13 the pheromone values of VC7 are given for nodes 1, 4, and 6. The plots include values for a selection of interfaces. In particular, node 1 contains 2 more interfaces than shown in the figure, but they were not visited during the simulations and therefor removed. In Figure 14 the pheromone values for interface 2 and 3 in node 1 are plotted for VC3 and VC7.

A few observations are made from these plots:

- When a VC has two (or more) alternative paths with almost the same cost value (see phase [0,50] in Figure 13(a)), all the corresponding pheromone values in the nodes will be updated.

- If a pheromone value is increasing, this link is updated and part of a preferred path. If it is decreasing, another link in the same node is part of a preferred path. This observation can be used to detect when a VC changes preferred path, and when a VC has alternative paths through the same node which makes this node more critical. See e.g. at time 150 in node 1 the preferred path is changing from interface 3 (link [1,3]) to 2 (link [1,2]).

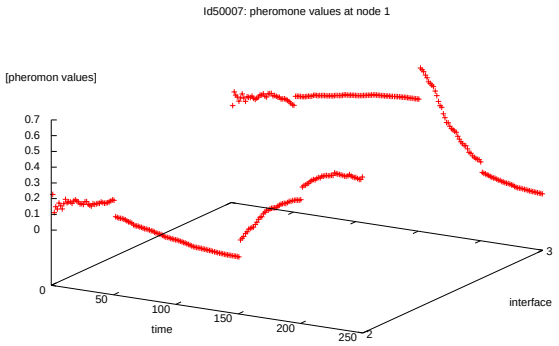
- When the frequency of ants decreases (e.g. at time 50 in Figure 13(b)), the node is no longer part of any of the preferred paths for a given VC. It stops completely, e.g. at time 150 in node 4. This is probably because the node is now unreachable so even the exploration ants will not visit it. Observe also that the pheromone values will not evaporate in the period with no ant visits to the node. This makes it likely to quickly return to the previous path when the links or/and nodes are recovered.

- When the frequency of ants increases (e,g. at time 150 in Figure 13(c)), the node is now becoming a part of one or more preferred paths for a given VC.
- Plotting all VC in a node (Figure 14 shows a sample of VC3 and VC7 in node 1), will visualize the *importance* (the number of VCs that have at least one preferred path through this node), *stability of node region* (the number of VCs that changes the ant frequency and pheromone values) and *criticality* (the number of VCs with more than one preferred path through this node).

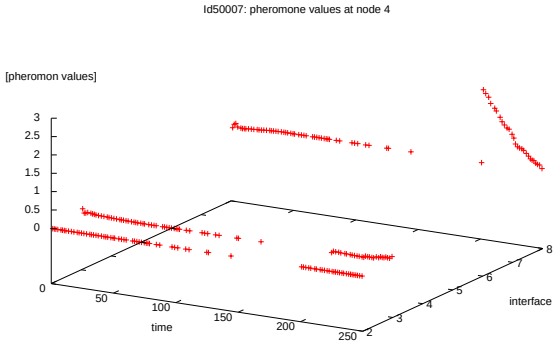
Time plot of accumulated overhead of AntPing and Ping. To give a brief indication of the overhead introduced by the AntPing monitoring system, the sum of packets sent and received by the source node for VC7 is plotted in Figure 15. The figure plots the number of traceroute packets because this is required to determine the path of a given VC. Traceroute is implemented by sending Ping packets with increasing TTL until the destination is reached. This means that if a path consists of n hops, n Ping packets must be sent and received by the source node.

From the figure it is observed that the AntPing has an initial high overhead to explore the network. Ping and traceroute also have an initial overhead because the underlying LS (link state) routing protocol generates $o(N \times L)$ packets where N is the number of nodes, and L is the average number of hops between two nodes. However, since the LS routing establishes the complete routing table between all node pairs, and AntPing in the current implementation is on-demand routing with no sharing of pheromones, the LS routing packets are not included. From the figure it is seen that AntPing has an initial boost of packets, and are boosting packets on changes in the topology or cost values. The traceroute will have constant number of packets unless the number of steps in the preferred path is changing. At time 200 in Figure 15 it is observed that the slope of the traceroute plot is increases. This is a result of a longer path for VC7. A few comments on the validity of this "comparison" is in order:

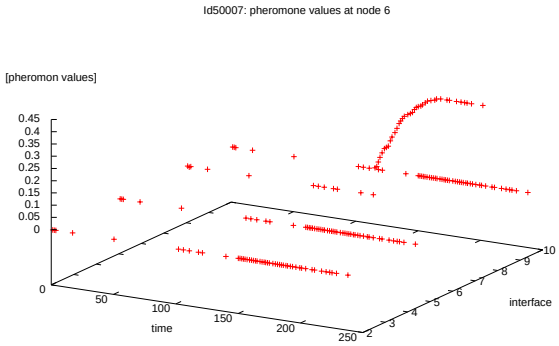
- Traceroute is an inefficient way of tracing the path from source to destination with respect to the number of packets. A better alternative is to use the route record option, but in IPv4 this is limited to 8 hops and it is not activate in all routers. This is improved in IPv6 that has dynamic headers and hence should in principle hold arbitrary number of hops.
- The frequency of packets sent by AntPing and Ping should be a compromise between what is necessary to detect the dynamic changes and to maintain information about the quality. In this simulation no extensive experimentation to determine this parameter is conducted. The same frequency is used by both AntPing and Ping.
- AntPing will in the current implementation create a unique set of pheromone values for each VC. To reduce the storage requirements, and the route convergence times, it is essential to let the different VC partially share the pheromone values, e.g. similar to the approach in [16].
- AntPing does only maintain information about the defined VCs and not all node pairs as required by LS routing.



(a) node 1



(b) node 4



(c) node 6

Figure 13: Pheromone values for VC7 in selected nodes. The nodes are part of preferred paths in different phases of the simulation experiment.

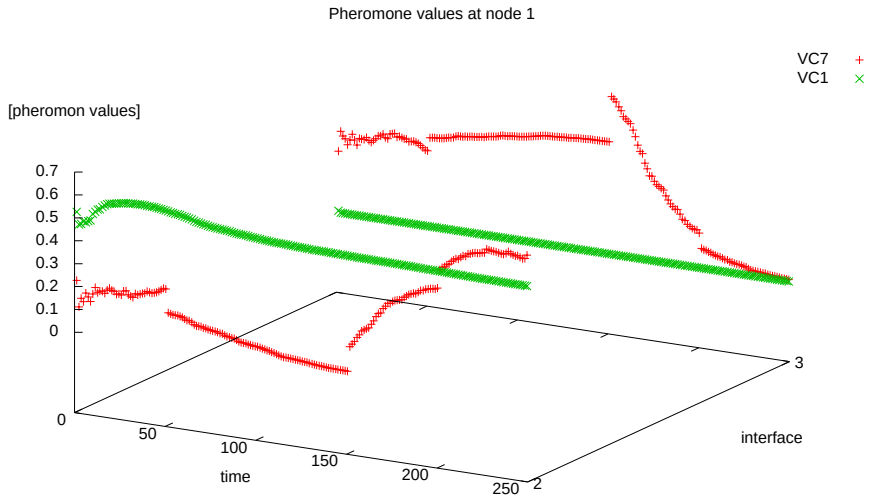


Figure 14: Pheromone values for VC3 and VC7 in node 1. It illustrates that VC7 changes preferred path at time 100 when link [2,4] fails. VC1 is not affected by the same dynamics, it has interface 2 as its preferred path for the entire period.

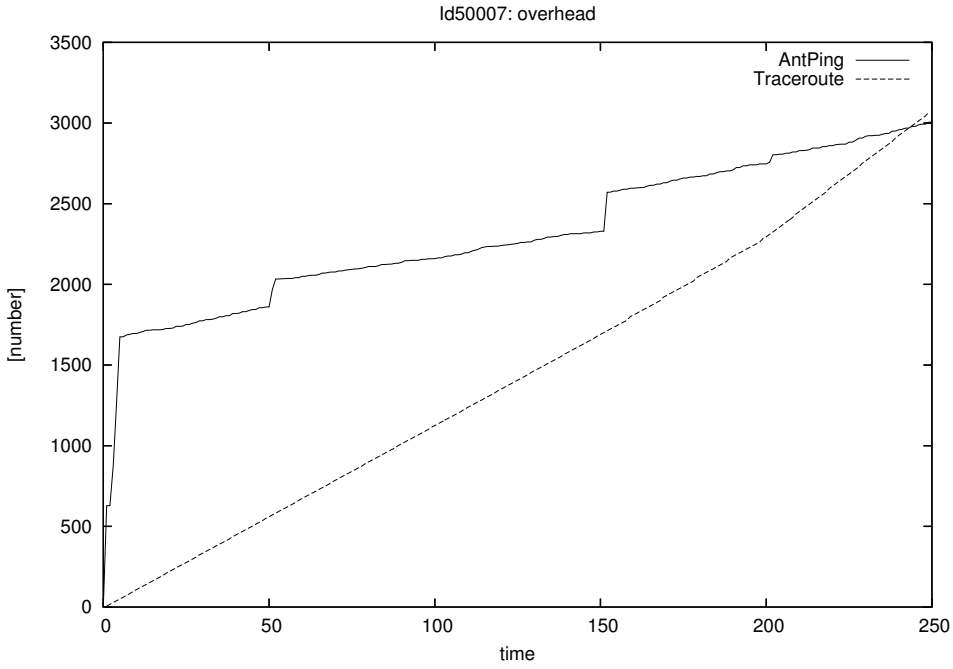


Figure 15: Overhead for AntPing and traceroute (multiple Ping) for monitoring of one VC. The link state routing overhead is not included in the Ping overhead.

14.3 Simulations of extended topology

In order to study how the AntPing and Ping behave when the network increases in size, the demo network is extended from the core of t.net, to the complete backbone of t.net, 216 nodes and 373 links illustrated in Figure 16. This section includes results and observations from the simulations. The scenario consists of the following phases:

- *Phase 1*, time [0, 10] : Initialization.
- *Phase 2*, time [10,50]: Stable phase.
- *Phase 3*, time [50, 100]: Link [2, 4] increases its delay from 0.10 ms to 4.00 ms to emulate increased traffic.
- *Phase 4*, time [100, 150]: Link [1, 42] is down.
- *Phase 5*, time [150, 200]: Link [2, 4] is down.
- *Phase 6*, time [200, 250]: Link [1, 42] is up.

The focus is now on VC1 connecting node 74 and 164 in Figure 16.

Time plot of cost and elite limit values. The following observations are made from the simulation results:

- At time 50 the delay on link [2, 4] is significantly increased. As was commented in the previous simulations, this change is not captured by the Ping over Link State routing because the link state metrics in this simulation case are not changed when the delay is changed. The AntPing will after a few seconds change the preferred path (at approximately 75 in the simulated example given in Figure 17. The same is observed in Figure 18 that shows the cost and elite limit averaged over 15 replications.
- Ping observes higher delay values from time 150 to 250. The reason is again that the static routing metrics does not reflect the link delays. In addition the static cost metrics on the low delay link between node 13 and 14 in Figure 10 was too high to be the shortest path obtained by the link state routing. The AntPing uses the link 13 to 14 as part of its preferred route from node 74 to 164.

Time plot of pheromone values of selected nodes and VCs. The preferred path from node 74 to 164 includes node 1 or 2 in all phases of the simulation experiment. In Figure 19 the pheromone values of a few interfaces (link to node) are given. The following observation are made:

- In phase 2 the interface towards node 42 in node 1, and interface towards node 41 in node 2 have positive values which means that both interfaces are amongst the preferred ones.

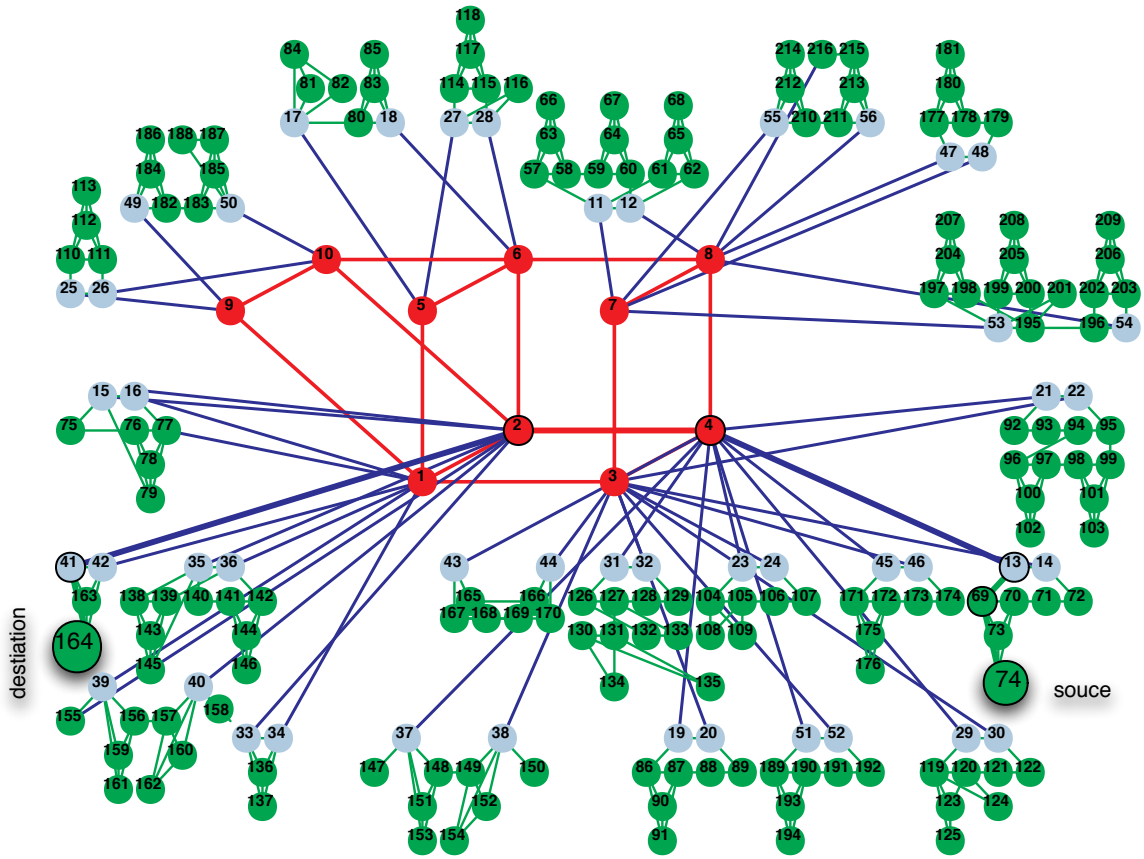


Figure 16: The simulated backbone network (model of *t.net*). The task is to establish and monitor the quality of the virtual connection (VC1) between node 74 (source) and node 164 (destination). The preferred path, {74,69,13,4,2,41,164}, of VC1 in phase 1 (stable phase) is indicated by thick lines.

- At time 50, the delay on link from node 4 to node 2 is significantly increased, and hence the path through node 1 is the preferred one (observe that the pheromone increases) and node 2 is no longer among the preferred ones (observe that the pheromone value decreases).
- At time 100, the link from node 1 to node 42 goes down. This is clearly observed in Figure 19(a) where the pheromone on this interface drops to 0. At the same time observe that interface towards node 41 in node 2 is now increased indicating the the preferred path now includes link from 2 to 41.

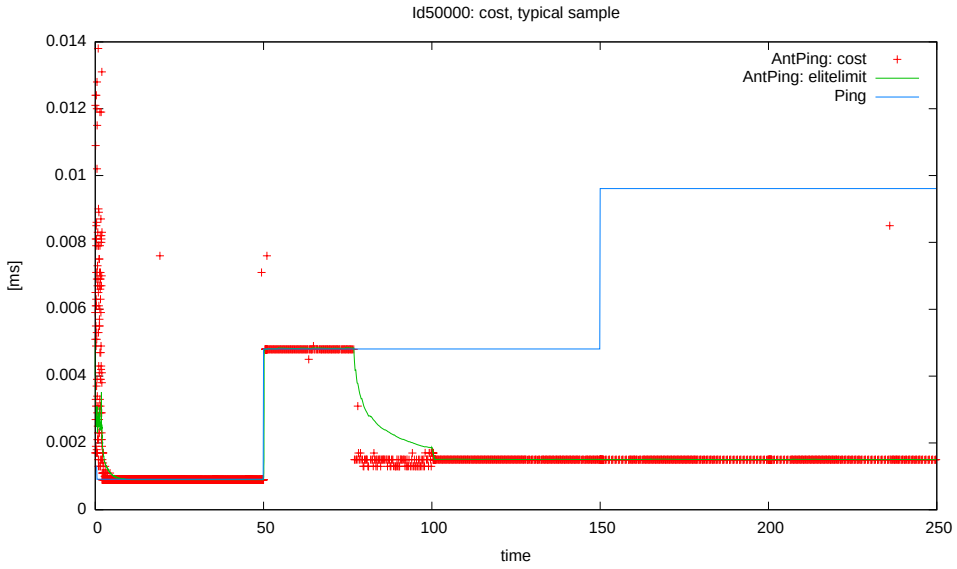


Figure 17: Cost and elite limit sample for typical time series for VC1 between node 74 and 164 in *t.net* backbone.

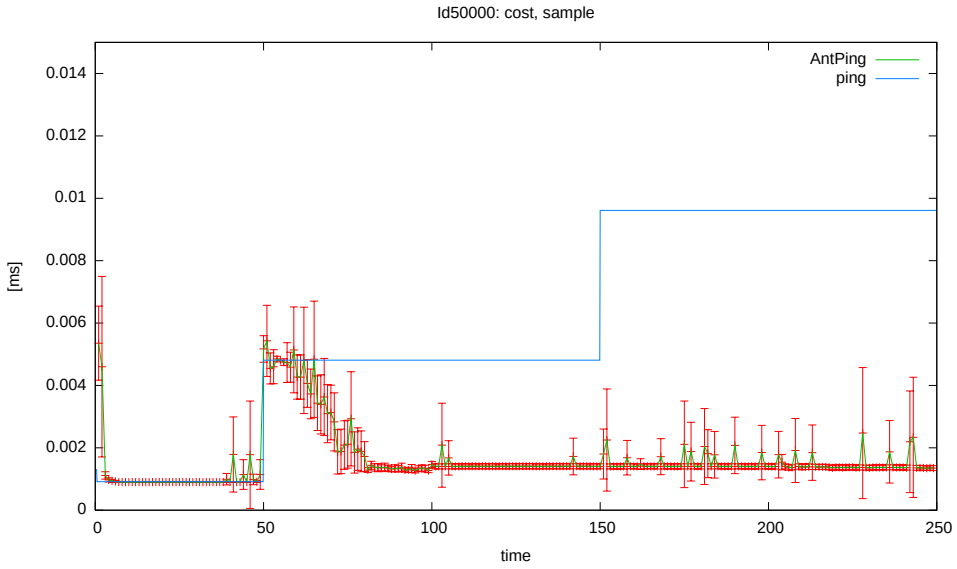
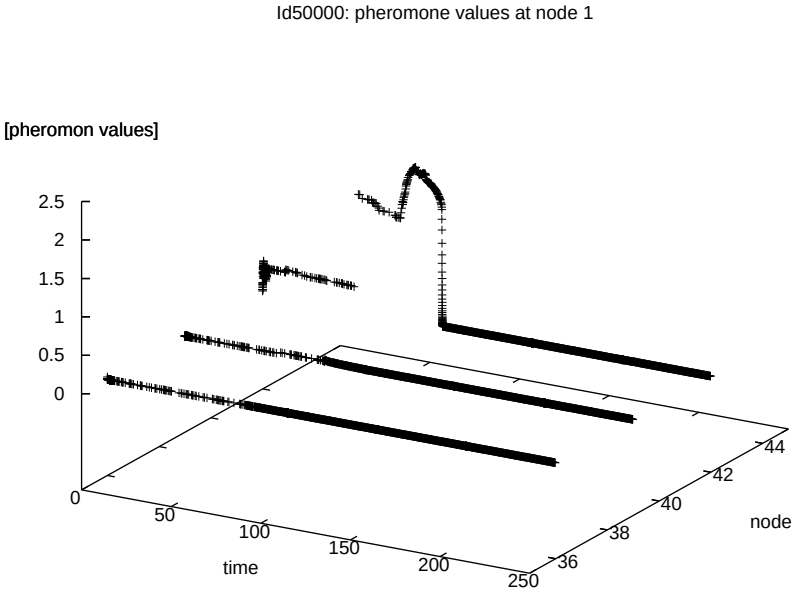
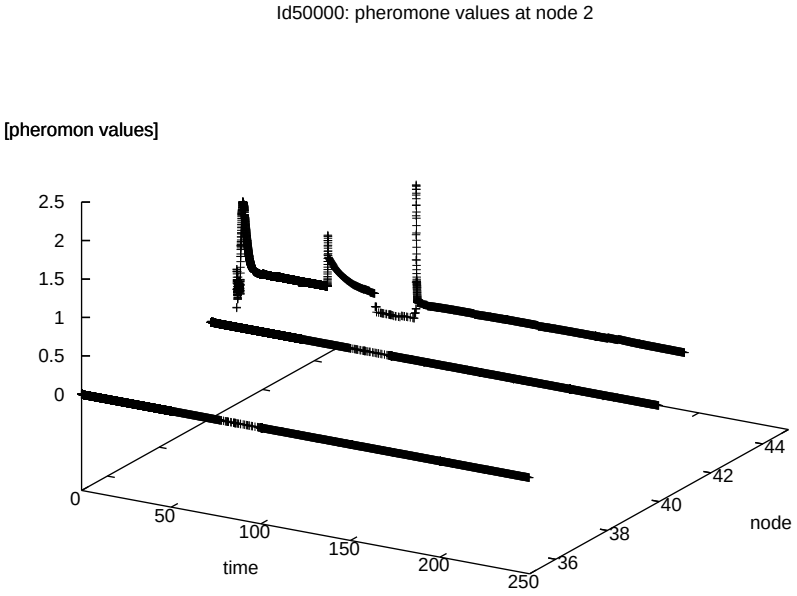


Figure 18: Average cost values (15 replications)



(a) node 1



(b) node 2

Figure 19: Pheromone values for VC1.

Part V
Closing remarks

The purpose of *AntPing* (ant-based routing and monitoring system) is to demonstrate that it is feasible to implement this on a software IP router. The prototype, named *AntPing*, demonstrates the *CE ants* principles in a small-scale network. The implementation is based on *Click* Modular software router system, and *hping*3[10] with TCL as an API for socket programming. The AntPing is implemented on home routers (in this demo *LinkSys WRT54GS(v4.0) routers*) with OpenWRT Linux. This implementation has quite modest hardware and software requirements, which makes the demo inexpensive, flexible, and easily portable. With a few extensions the implementation can be used as (ad-hoc) "hot-spots" in wireless infrastructures, e.g. like the RoofNet `http://www.comclub.org/roofnet/` initiative. It is also feasible to exchange the CE ants swarm routing algorithm with the AntHocNet algorithm for MANET developed in the BISON project.

The demonstrator visualizes the inner workings of the ant algorithm by animation of ants moving and dropping in the network, and topology changes like link failures and restorations. The animation also shows ants that do not find the destination but are dropped because the TTL is expired. In addition, the changes in cost values of each virtual path is plotted as a function over time, both the cost of the current best cost, and the last cost, even when rejected by the elite selection.

Previous studies [4] of the transient behaviour of *CE ants* [5], a Cross Entropy based Ant system for path management, identified that several adaptive components of such a system can be used as indicators of the network condition status with respect to traffic load level and topology. The indicators are the stochastic routing matrix (the pheromones), routing path probability, cost values, and grade of convergence (denoted temperature in CE ants).

The dynamics in the demo trial are up to the audience to define in the sense that they are allowed to unplug and plug the cables. Adding new interfaces or links that are not predefined are not possible because a discovery protocol are not yet implemented. Further more, dual stack routing logic is not implemented, and therefor it is not possible to run AntPing and Ping over OSPF in parallel. This can be solved by adding a working OSPF/RIP/BGP routing implementation, e.g. by Zebra (zebra `www.zebra.org`) the LinkSys boxes can compare Ping and AntPing. Hence, to demonstrate how the AntPing compares to Ping using link state routing, how it scales with increasing virtual connections and network size, a few series of simulations in ns-2 are conducted. The results in this report is related to the monitoring of this quality, and the overhead introduced by means of AntPing. The AntPing is compared to the current practice of using Ping and traceroute to monitor virtual routes and paths in an interdomain network typically routed by *OSPF* (link state routing protocol).

Changes in the network topology, and in cost or delay on links, are easily observed both by AntPing and Ping. But, if the cost metrics are not consistently set to reflect the (expected or observed) delays, the routing of AntPing and Ping might end up following different routes, meaning that the AntPing is able to detect potential misconfiguration.

[10] www.hping.org

An alternative to observing the VC quality in the end systems (in source and/or destination nodes) some information and indications of changes can be obtained by observing the pheromone values of the *CE ants* in the intermediate nodes. When a VC has two (or more) alternative paths with almost the same cost value the corresponding pheromone values in the nodes will be updated. If a pheromone value is increasing, the link is part of a preferred path that is updated. If it is decreasing, another link in the same node is part of a preferred path.

The frequency of packets sent by AntPing and Ping should be a compromise between what is required in order to detect the dynamic and to maintain information about the quality. In this simulation no extensive experimentation is conducted to determine how to set or adjust the frequency of ants and Ping packets. This is current ongoing research.

References

[1] Gianni Di Caro, Frederick Ducatelle, Poul Heegaard, Mark Jelasity, and Roberto Montemanni. Evaluation of basic services in ahn, p2p and grid networks. Deliverable 07 of IST-FET Project BISON (IST-2001-38923), December 2004.

[2] Gianni Di Caro, Frederick Ducatelle, Poul Heegaard, Mark Jelasity, and Roberto Montemanni. Implementation of basic services in ahn, p2p and grid networks. Deliverable 06 of IST-FET Project BISON (IST-2001-38923), December 2004.

[3] Marco Dorigo and Gianni Di Caro. Ant Algorithms for Discrete Optimization. *Artificial Life*, 5(3):137–172, 1999.

[4] P.E. Heegaard. Performance monitoring of routing stability in dynamic networks. Section 5 in Deliverable 05: "Models for basic services in AHN, P2P and Grid networks" in IST-FET Project BISON (IST-2001-38923), December 2003.

[5] Poul E. Heegaard, Otto Wittner, Victor F. Nicola, and Bjarne E. Helvik. Distributed asynchronous algorithm for cross-entropy-based combinatorial optimization. Budapest, Hungary, September 7-8 2004.

[6] Bjarne E. Helvik and Otto Wittner. Using the Cross Entropy Method to Guide/Govern Mobile Agent's Path Finding in Networks. In *Proceedings of 3rd International Workshop on Mobile Agents for Telecommunication Applications*. Springer Verlag, August 14-16 2001.

[7] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[8] Anders Mykkeltveit, Poul Heegaard, and Otto Wittner. Realization of a distributed route management system on software routers. In *Proceedings of Norsk Informatikkonferanse*, Stavanger, Norway, 29. Nov - 1. Dec 2004.

[9] Jon Postel. Rfc 762: Assigned numbers. IETF, January 1980.

[10] Jon Postel. RFC 791: Internet Protocol. IETF, September 1981.

[11] Jon Postel. Rfc 792: Internet control message protocol (icmp). IETF, September 1981.

[12] R. Y. Rubinstein. Combinatorial Optimization, Cross-Entropy, Ants and Rare Events. In S. Uryasev and P. M. Pardalos, editors, *Stochastic Optimization: Algorithms and Applications*. Kluwer Academic Publishers, 2001.

[13] R. Schoonderwoerd, O.E. Holland, J. Bruten, and L. Rothkrantz. Ant-based load balancing in telecommunications networks. Technical Report HPL-96-76, HP Labs, May 1996.

[14] W. Richard Stevens. *TCP/IP Illustrated, Volume 1: The Protocols*. ISBN 0-201-63346-9. Addison-Wesley, 1994.

[15] Otto Wittner. *Emergent Behavior Based Implements for Distributed Network Management*. PhD thesis, The Norwegian University of Science and Technology, November 2003.

[16] Otto Wittner, Poul E. Heegaard, and Bjarne E. Helvik. Scalable distributed discovery of resource paths in telecommunication networks using cooperative ant-like agents. In *Proceedings of Congress on Evolutionary Computation, CEC2003*, Canberra, Australia, December 2003. IEEE.

[17] Otto Wittner and Bjarne E. Helvik. Cross Entropy Guided Ant-like Agents Finding Dependable Primary/Backup Path Patterns in Networks. In *Proceedings of Congress on Evolutionary Computation (CEC2002)*, Honolulu, Hawaii, May 12-17th 2002. IEEE.

A ICMP ping

The Internet Protocol (IP) is used to transport datagrams between Internet hosts. It is known as a connectionless datagram service. The ICMP protocol provides a method of sending information to the source host regarding information about a datagram. ICMP is a separate protocol from IP, but every IP implementation must include the ICMP protocol. ICMP makes use of the IP protocol as a transport. The purpose of ICMP is to provide feedback regarding the network and datagrams. ICMP messages use a basic IP datagram header with the IP data being the ICMP message. The IP source address is that of the host or gateway sending the ICMP message with the destination IP address being that of the original source IP address. RFC 792 [11] states that, where applicable, each ICMP message contains the IP header and first 64 bits of the original datagram which is used to match the datagram to a process. RFC 1812 [12] indicates that this is no longer adequate and an ICMP message, where applicable, should contain as much of the original datagram as possible that can fit within a 576 byte message.

ICMP Ping checks a remote host for availability. Local hosts should normally respond to ping requests within milliseconds. However, on a very congested network it may take up to 3 seconds or longer to receive an echo packet from the remote host. If the timeout is set too low under these conditions, it will appear that the remote host is not reachable (which is almost the truth). ActiveXperts Network Monitor checks servers for availability by sending ICMP Echo commands and wait for the responds. An ICMP timeout failure doesn't necessarily mean that the remote host is actually functioning beyond its ability to echo packets. An ICMP/Ping check has the following parameters:

Hostname or IP address - The DNS name or IP address of the computer you want to ping (can even be a WINS name, but only if the name can be resolved by some WINS server in the network);

Timeout for each reply - Maximum number of milliseconds it may take before a response is received;

Time to Live - Maximum Time to Live (TTL) value;

Number of Echo requests to send - Maximum number of milliseconds it may take before a response is received.

B Route record and source routing

B.1 IP Record Route Option

Excerpt from Stevens [14]:

"The ping program gives us an opportunity to look at the IP record route (RR) option. Most versions of ping provide the -R option that enables the record route feature. It causes ping

[11] http://www.ietf.org/rfc/rfc792.txt
[12] http://www.ietf.org/rfc/rfc1812.txt

to set the IP RR option in the outgoing IP datagram (which contains the ICMP echo request message). This causes every router that handles the datagram to add its IP address to a list in the options field. When the datagram reaches the final destination, the list of IP addresses should be copied into the outgoing ICMP echo reply, and all the routers on the return path also add their IP addresses to the list. When ping receives the echo reply it prints the list of IP addresses.

As simple as this sounds, there are pitfalls. Generation of the RR option by the source host, processing of the RR option by the intermediate routers, and reflection of the incoming RR list in an ICMP echo request into the outgoing ICMP echo reply are all optional features. Fortunately, most systems today do support these optional features, but some systems don't reflect the IP list.

The biggest problem, however, is the limited room in the IP header for the list of IP addresses. We saw in Figure 3.1 that the header length in the IP header is a 4-bit field, limiting the entire IP header to 15 32-bit words (60 bytes). Since the fixed size of the IP header is 20 bytes, and the RR option uses 3 bytes for overhead (which we describe below), this leaves 37 bytes (60-20-3) for the list, allowing up to nine IP addresses. In the early days of the ARPANET, nine IP addresses seemed like a lot, but since this is a round-trip list (in the case of the -R option for ping), it's of limited use today. Despite these shortcomings, the record route option works and provides an opportunity to look in detail at the handling of IP options. Figure 20 shows the general format of the RR option in the IP datagram.

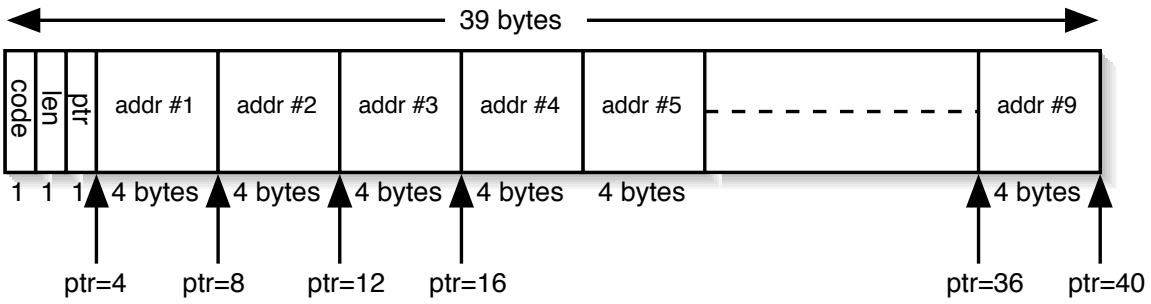


Figure 20: General format of record route option in IP header.

Code is a I-byte field specifying the type of IP option. For the RR option its value is 7. *Len* is the total number of bytes of the RR option, which in this case is 39. (Although it's possible to specify an RR option with less than the maximum size, ping always provides a 39-byte option field, to record up to nine IP addresses. Given the limited room in the IP header for options, it doesn't make sense to specify a size less than the maximum.)

Ptr is called the pointer field. It is a 1-based index into the 39-byte option of where to store the next IP address. Its minimum value is 4, which is the pointer to the first IP address. As each IP address is recorded into the list, the value of ptr becomes 8, 12, 16, up to 36. After the ninth address is recorded ptr becomes 40, indicating the list is full.

When a router (which by definition is multihomed) records its IP address in the list, which IP

address is recorded? It could be the address of the incoming interface or the outgoing interface. RFC 791 [10] specifies that the router records the outgoing IP address. We'll see that when the originating host (the host running ping) receives the ICMP echo reply with the RR option enabled, it also records its incoming IP address in the list. "

B.2 IP Source Routing Option

Excerpt from Stevens [14]:

"Normally IP routing is dynamic with each router making a decision about which next-hop router to send the datagram to. Applications have no control of this, and are normally not concerned with it. It takes tools such as Traceroute to figure out what the route really is.

The idea behind source routing is that the sender specifies the route. Two forms are provided:

- *Strict source routing*. The sender specifies the exact path that the IP datagram must follow. If a router encounters a next hop in the source route that isn't on a directly connected network, an ICMP "source route failed" error is returned.
- *Loose source routing*. The sender specifies a list of IP address that the datagram must traverse, but the datagram can also pass through other routers between any two addresses in the list.

Traceroute provides a way to look at source routing, as we can specify an option allowing us to force a source route, and see what happens.

Some of the publicly available Traceroute source code packages contain patches to specify loose source routing. But the standard versions normally don't include this option. A comment in the patches is that "Van Jacobson's original traceroute (spring 1988) supported this feature, but he removed it due to pressure from people with broken gateways." For the examples shown in this section, the author installed these patches and modified them to allow both loose and strict source routing.

Figure 20 shows the format of the source route option.

This format is nearly identical to the format of the record route option that we showed in Figure 20. But with source routing we have to fill in the list of IP addresses before sending the IP datagram, while with the record route option we allocate room and zero out the list of IP addresses, letting the routers fill in the next entry in the list. Also, with source routing we only allocate room for and initialize the number of IP addresses required, normally fewer than nine. With the record route option we allocated as much room as we could, for up to nine addresses.

The code is 0x83 for loose source routing, and 0x89 for strict source routing. The len and ptr fields are identical to what we described in Section B.1. The source route options are actually called "source and record route" (LSRR and SSRR, for loose and strict) since the list of IP addresses is updated as the datagram passes along the path. What happens is as follows:

- The sending host takes the source route list from the application, removes the first entry (it becomes the destination address of the datagram), moves all the remaining entries left

by one entry (where left is as in Figure 20), and places the original destination address as the final entry in the list. The pointer still points to the first entry in the list (e.g., the value of the pointer is 4).

- Each router that handles the datagram checks whether it is the destination address of the datagram. If not, the datagram is forwarded as normal. (In this case loose source routing must have been specified, or we wouldn't have received the datagram.)
- If the router is the destination, and the pointer is not greater than the length, then (1) the next address in the list (where ptr points) becomes the destination address of the datagram, (2) the IP address corresponding to the outgoing interface replaces the source address just used, and (3) the pointer is incremented by 4.

This is best explained with an example. In Figure 21 we assume that the sending application on host S sends a datagram to D, specifying a source route of R1, R2, and R3.

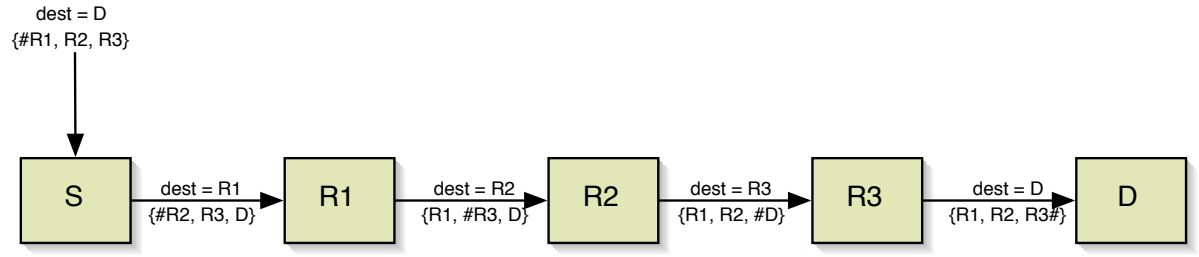


Figure 21: Example of IP source routing.

In this figure the pound sign (#) denotes the pointer field, which assumes the values of 4, 8, 12, and 16. The length field will always be 15 (three IP addresses plus 3 bytes of overhead). Notice how the destination address of the IP datagram changes on every hop.

When an application receives data that was source routed, it should fetch the value of the received route and supply a reversed route for sending replies.

The Host Requirements RFC specifies that a TCP client must be able to specify a source route, and that a TCP server must be able to receive a source route, and use the reverse route for all segments on that TCP connection. If the TCP server later receives a different source route, that newer source route overrides the earlier one."

C Hping scripts

C.1 Send-nam.htcl

```
#
# Name: send-nam.htcl
# Purpose: hping tcl script for generating ants in the BISON demo
# Description: The ants are IP packets with dst port 51234 sent to a
#              process "recv.htcl" that resides on the destination node.
#              This process calculates the temperature based
#              on the cost values seen so far and the cost value carried
```

```
#               by the last ant. For this demo, the TTL is set
#               to 8 because this is the maximum number of steps that can
#               be observed by the route record in IPv4 optional header.
#               The cost and temperature of the returning ants are written
#               to a trace file named "trace"
#
#               Change the use of tos bit. Cause problems when sending from
#               windows machines and through commersial routers.
#
#               Extends "send.htcl" by generating NAM trace data from cost values.
#
# Project: BISON
# Documantation: Deliverable D14 of BISON
# Last modified: 2006-03-10
# Author: Poul Heegaard, Telenor R&D
#

# Check arguements, expects $target and $inf and $antId
if {$argc < 5} {
    puts stderr "Usage: send <target> <inf> <antid> <data> <freq> \[<explAnts>\] \[<rho>\] \[<beta>\] \[<explRatio>\]""
    puts stderr "Example: send 10.7.8.2 eth0 50000 0 1000 \[10 0.05 0.98 0.20\]"
    exit
}

# globals: read argument list
foreach {target inf antId data freq explAnts rho beta} $argv break
# observe that $data == 0 is ants, $data == 1 is data

# CEants parameters (these might also later be given as arguments)
if {$rho == {}} { set rho 0.05 }
if {$beta == {}} { set beta 0.98 }
# the 10 first ants are explaratory ants
if {$explAnts == {}} { set explAnts 10 }
# Default is 20% of the ants are maintenance (explaration) ants
if {$explRatio == {}} { set explRatio 0.2 }
# this means 1 sec between each ant
# set freq 1000

# Nam topology and initialisations
# test
set node(127.0.0.1) 0
# demo net
#set node(10.100.1.1) 0
#set node(10.100.1.11) 1
set node(10.3.11.1) 3
set node(10.3.11.2) 0
set node(10.3.12.2) 0
set node(10.1.2.1) 1
set node(10.1.3.1) 1
set node(10.1.5.1) 1
set node(10.1.9.1) 1
set node(10.1.2.2) 2
set node(10.2.4.1) 2
set node(10.2.6.1) 2
set node(10.2.10.1) 2
set node(10.1.3.2) 3
set node(10.3.4.1) 3
set node(10.3.7.1) 3
set node(10.2.4.2) 4
set node(10.3.4.2) 4
set node(10.4.8.1) 4
set node(10.1.5.2) 5
set node(10.5.6.1) 5
set node(10.2.6.2) 6
set node(10.5.6.2) 6
set node(10.6.10.1) 6
set node(10.6.8.1) 6
set node(10.3.7.2) 7
set node(10.7.8.1) 7
set node(10.4.8.2) 8
set node(10.6.8.2) 8
set node(10.7.8.2) 8
set node(10.1.9.2) 9
set node(10.9.10.1) 9
set node(10.2.10.2) 10
set node(10.6.10.2) 10
set node(10.9.10.2) 10
set node(10.8.11.1) 8
set node(10.8.11.2) 11

# begin demo nam preamble
set delay 0.060
set delayms 60ms
# open nam file for animations
set namfile [open "t$antId.nam" w]
puts $namfile "V -t * -v 1.0a5"
puts $namfile "n -t * -s 11 -v box -c black -z 10 -S UP -b SRC"
puts $namfile "n -t * -s 10 -v box -c black -z 10 -S UP -b AR-10"
```

```
puts␣$namfile␣"n -t * -s 9 -v box -c black -z 10 -S UP -b AR-9"
puts␣$namfile␣"n -t * -s 8 -v box -c black -z 10 -S UP -b AR-8"
puts␣$namfile␣"n -t * -s 7 -v box -c black -z 10 -S UP -b AR-7"
puts␣$namfile␣"n -t * -s 6 -v box -c black -z 10 -S UP -b AR-6"
puts␣$namfile␣"n -t * -s 5 -v box -c black -z 10 -S UP -b AR-5"
puts␣$namfile␣"n -t * -s 4 -v box -c black -z 10 -S UP -b AR-4"
puts␣$namfile␣"n -t * -s 3 -v box -c black -z 10 -S UP -b AR-3"
puts␣$namfile␣"n -t * -s 2 -v box -c black -z 10 -S UP -b AR-2"
puts␣$namfile␣"n -t * -s 1 -v box -c black -z 10 -S UP -b AR-1"
puts␣$namfile␣"n -t * -s 0 -v box -c black -z 10 -S UP -b DST"
puts␣$namfile␣"l -t * -s 9 -d 10 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 8 -d 11 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 7 -d 8 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 6 -d 10 -r 100kb -D $delayms -c black -o 180deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 6 -d 8 -r 100kb -D $delayms -c black -o 0deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 5 -d 6 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 4 -d 8 -r 100kb -D $delayms -c black -o 90deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 3 -d 7 -r 100kb -D $delayms -c black -o 90deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 3 -d 4 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 2 -d 10 -r 100kb -D $delayms -c black -o 135deg -l 141-S UP"
puts␣$namfile␣"l -t * -s 2 -d 6 -r 100kb -D $delayms -c black -o 90deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 2 -d 4 -r 100kb -D $delayms -c black -o 0deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 1 -d 9 -r 100kb -D $delayms -c black -o 135deg -l 141 -S UP"
puts␣$namfile␣"l -t * -s 1 -d 5 -r 100kb -D $delayms -c black -o 90deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 1 -d 3 -r 100kb -D $delayms -c black -o 0deg -l 100 -S UP"
puts␣$namfile␣"l -t * -s 1 -d 2 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"l -t * -s 0 -d 3 -r 100kb -D $delayms -c black -o 45deg -l 50 -S UP"
puts␣$namfile␣"c -t * -i 0 -n Blue"
puts␣$namfile␣"c -t * -i 1 -n Green"
puts␣$namfile␣"c -t * -i 2 -n Brown"
puts␣$namfile␣"c -t * -i 3 -n Yellow"
puts␣$namfile␣"c -t * -i 4 -n Yellow"
puts␣$namfile␣"c -t * -i 5 -n Red"
puts␣$namfile␣"a -t 0 -s $node([hping outifa $target]) -n AntGen"
puts␣$namfile␣"a -t 0 -s $node([hping resolve $target]) -n AntRec"
#␣end␣demo␣nam␣preamble

if␣{␣$data␣==␣0␣}␣{
␣␣#␣print␣ant␣setup
␣␣puts␣"--------------------------------------------------------------------"
␣␣puts␣"Finds connections from [hping outifa $target] to $target ([hping resolve $target])"
␣␣puts␣"sending an ant every $freq ms, with [expr 100*$explRatio]% exploration"
␣␣puts␣"number of initial exploration ants = $explAnts, "
␣␣puts␣"and rho=$rho and beta=$beta."
␣␣puts␣"--------------------------------------------------------------------"
␣␣puts␣" "
}␣else␣{
␣␣#␣print␣data␣setup
␣␣puts␣"--------------------------------------------------------------------"
␣␣puts␣"Data sent on connection from [hping outifa $target] to $target ([hping resolve $target])"
␣␣puts␣"sending data every $freq ms
  puts "--------------------------------------------------------------------"
  puts "␣"
}

# Init variables
set accTime 0
# counts number of received ants
set recvind 0
# counts number of sent ants
set sentind 1
#set tcl_precision 4

# Send ants every $freq [ms] to destination as specified in $target
proc genAnts {} {
  global target freq sentind recvind explAnts explRatio antId rho beta delay accTime

  # explore = 0 : maintenace ants and data packets
  # explore = 1 : exploration ants
  # explore = 2 : exploration ants, initialisation phase
  # cost = 0: ant packets
  # cost = 1: data packets

  # the ant is sent as exploration ant at startup and later as maintenance
  # the initial exploration phase is over when the predefined number of ants
  # are received
  if {$recvind <= $explAnts} {
    set explore 2
  } else {
    set explore [expr rand() < $explRatio ]
  }

  if { $data == 1 } {
    set explore 0
  }

  # construct packet to be sent to $tagret and with route record
  set pck "ip(daddr=$target,ttl=8)+ip.rr"
```

```
  append pck "+udp(sport=$antId,dport=51234)"
  set cost $data

  if { $explore == 2 } {
    # Send initialisation exploration ant to reset
    # the variables on receiver for port number $antId
    append pck "+data(str=$explore/$rho/$beta/$cost)"
  } else {
    # cost values for testing of NAM trace generation
    append pck "+data(str=$explore/$cost)"
  }
  # send packet
  incr sentind 1

  puts "Generated: $pck"
  hping send $pck

  # next packet using "bootstrapping"
  after $freq genAnts
}

# Read incoming ants and write cost and current temperature to $outfile
proc recAnts {} {
  global inf outfile outfile2 namfile recvind antId node delay target accTime

  # receive packet accoring to filter settings
  set p [hping recv $inf 10]
  set pck [lindex $p 0]

  # read source and destionation ports and data field
  set sp [hping getfield udp sport $pck]
  set dp [hping getfield udp dport $pck]
  set dt [hping getfield data str $pck]

  # content of data field:
  # 0 = expldata
  # 1 = exp(-cost/temp)  (for click)
  # 2 = cost
  # 3 = temp
  # 4 = beta
  # 5 = accTime (for nam trace)
  # 6 = #hops in path
  # 7 - (7+#hops-1) = table of costs
  # (7+#hops) - end = route addresses
  set tab  [split $dt /]
  set explore [lindex $tab 0]
  set cost [lindex $tab 2]
  set temp [lindex $tab 3]
  set beta [lindex $tab 4]
  set tmpaccTime [lindex $tab 5]
  if { $accTime < $tmpaccTime } {
    set accTime $tmpaccTime
  }

  # determine number of hops
  set hops [lindex $tab 6]

  # is this an ant? (the filter settings did not work when too restrict)
  if { $sp == 51234 } {
    puts "Received: $pck"
    set dt [hping getfield data str $pck]
    set lsrr [hping getfield ip.lsrr data $pck]
    set lsrl [hping getfield ip.lsrr ptr $pck]
    # Update ant received - write cost and temp to trace file
    if { $explore == 0 } {
      incr recvind 1
      # puts "$recvind $temp $temp2 $cost"
      # puts "Source route: $lsrr"
      # print to "trace"
      # puts $outfile "$recvind $temp $cost"
      # write temp all also (use beta field)
      #puts $outfile "$recvind $temp $cost"
      puts $outfile "$accTime $temp $cost"
      flush $outfile
    }
    if { $explore == 1 } {
      puts $outfile2 "$accTime $temp $cost"
      flush $outfile2
    }

    # read list of hosts
    set sa [hping getfield ip saddr $pck]
    set da [hping getfield ip daddr $pck]
    puts $lsrr
    set thost [split $lsrr /]
    puts "Hosts, pre-pre: $lsrl $thost $hops"
    set hops [expr "$lsrl / 4 - 2"]
    #set thost [lrange $thost 0 [expr $hops-2]]
```

```
set thost [lrange $thost 0 $hops]
# reverse host list
set ttmp {}
set ind [llength $thost]
puts "Hosts, pre: $thost $ind $hops"
while {$ind>=1} {
  lappend ttmp [lindex $thost $ind]
  incr ind -1
}
set thost [lrange $ttmp 1 end]
# append source address of reply packet, i.e. target
lappend thost $sa
# print to check
puts "Hosts: $thost"

if { $explore == 0 }  {
  # read the cost of each hop in the path
  set tcost [lrange $tab 7 [expr 7 + $hops - 1]]
  puts "Costs: $tcost"
}

# run through the routing table to build nam events
# h = sent on link (leave node)
# d = drop on link
# set size [hping getfield ip length $pck]
set size 1000
set i 0
set shost $da
if { $explore < 3 } {
  # forward path to the destination
  foreach dhost $thost {
    # set fixed time addition on each hop to control animation
    set accTime [expr $accTime+$delay]
    set dhost [lindex $thost $i]
    puts $namfile "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 0 -a 0"
    flush $namfile
    puts "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 0"
    set shost $dhost
    incr i 1
  }
}

# if not leading to an update: dropped at destination or at router
if { $explore > 0 } {
  set accTime [expr $accTime+$delay]

  set last [expr [llength $thost]-1]
  set shost [lindex $thost $last]
  if { $last > 1 } {
    set dhost [lindex $thost [expr $last - 1]]
  } else {
    set dhost $da
  }

  if { $explore == 1 | $explore == 2 } {
    puts $namfile "d -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 5 -a $explore"
    flush $namfile
    puts "d -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 5 -a $explore"
  }
  if { $explore == 3 } {
    # determine the other end of the link state event
    set tmp [split $shost .]
    set tmpI [lindex $tmp 3]
    if { $tmpI == 1 } {
      set tmplast 2
    } else {
      set tmplast 1
    }
    set tmp [lrange $tmp 0 2 ]
    lappend tmp $tmplast
    set dhost [join $tmp "." ]
    puts "shost: $shost"
    puts "dhost: $dhost"

    puts $namfile "l -t $accTime -s $node($shost) -d $node($dhost) -S DOWN"
    flush $namfile
    puts "l -t $accTime -s $node($shost) -d $node($dhost) -S DOWN"
  }
  if { $explore == 4 } {
    # determine the other end of the link state event
    set tmp [split $shost .]
    set tmpI [lindex $tmp 3]
    if { $tmpI == 1 } {
      set tmplast 2
    } else {
      set tmplast 1
    }
    set tmp [lrange $tmp 0 2 ]
```

```
        lappend tmp $tmplast
        set dhost [join $tmp "." ]
        puts "shost: $shost"
        puts "dhost: $dhost"

        puts $namfile "l -t $accTime -s $node($shost) -d $node($dhost) -S UP"
        flush $namfile
        puts "l -t $accTime -s $node($shost) -d $node($dhost) -S UP"
      }
    } else {
      set accTime [expr $accTime+$delay]
      puts $namfile "h -t $accTime -s $node($shost) -d $node($target) -e $size -i 0 -a 0"
      flush $namfile
      puts "h -t $accTime -s $node($shost) -d $node($target) -e $size -i 0 -a 0"
      # backward path from the destination
      # reverse host lists
      set ttmp {}
      set ind [llength $thost]
      while {$ind>-1} {
        lappend ttmp [lindex $thost $ind]
        incr ind -1
      }
      set tcost [lrange $ttmp 1 end]
      set thost [lrange $thost 0 end-1]
      set thost [concat $da $thost]
      set i [expr [llength $thost]-1]
      set shost $sa
      foreach n $tcost {
        set accTime [expr $accTime+$delay]
        set dhost [lindex $thost $i]
        puts $namfile "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 1 -a 1"
        flush $namfile
        puts "h -t $accTime -s $node($shost) -d $node($dhost) -e $size -i 1 -a 1"
        set shost $dhost
        incr i -1
      }
      set accTime [expr $accTime+$delay]
      puts $namfile "h -t $accTime -s $node($dhost) -d $node([hping outifa $target]) -e $size -i 1 -a 1"
      flush $namfile
      puts "h -t $accTime -s $node($dhost) -d $node([hping outifa $target]) -e $size -i 1 -a 1"
    }
  }

  # reread $inf after 1 ms
  after 1 recAnts
}

# open trace file
set outfile [open "../trace-$antId" w]
set outfile2 [open "../trace-$antId-all" w]

# Set filter
hping setfilter $inf "udp"

after 1 genAnts
after 1 recAnts

vwait forever
```

C.2 Recv-nam.htcl

```
#
# Name: recv-nam.htcl
# Purpose: hping tcl script for receiving ants in the BISON demo
# Description: This script "recv.htcl" receives ants (UDP packets)
#              on port 51234 and returns ants to $target on the same port.
#              The different roles and directions of the ant packets are
#              indicated by the use of the TOS byte (see D14 of BISON).
#              The ants are sent by the "send-nam.htcl" process on the
#              $target node.
#              Elite selection:
#              The temperature is updated if cost is below as threshold.
#              The update is for each ant species as indicated by the
#              src port number of the received packets. The threshold is
#              updated as new ants and cost values are observed.
#              Ants with cost values that not qualifies for the elite,
#              will return to the source with animation data only and
#              will not cause any update in the intermediate nodes.
#
#              Change the use of tos bit. Cause problems when sending from
#              windows machines and through commersial routers.
#
#              Extends previous version "recv.htcl" by animation tracing,
#              elite selection and type classifications.
#
# Project: BISON
```

```
# Documantation: Deliverable D14 of BISON
# Last modified: 2006-03-10
# Author: Poul Heegaard, Telenor R&D
#

# Check arguements
if {$argc != 1} {
    puts stderr "Usage: send <inf>"
    puts stderr "Example: recv eth0"
    exit
}

# globals: read argument list
foreach {inf} $argv break

set antTab {}
set accTime 0
set verysmall 0.000001
set alpha 0.9

# Set filter
hping setfilter $inf "udp and dst port 51234"

# do forever: read incoming ants (IP packets)
while 1 {
  incr accTime 1
  set p [hping recv $inf]
  set pck [lindex $p 0]

  puts $pck

  # get type (expldata), ports, route record, and cost values from packet
  set target [hping getfield ip saddr $pck]
  set dp [hping getfield udp dport $pck]
  set sp [hping getfield udp sport $pck]
  set rr [hping getfield ip.rr data $pck]
  set rl [hping getfield ip.rr ptr $pck]
  set dt [hping getfield data str $pck]
  set expldata  [lindex [split $dt /] 0]

# is this an ant?
if { $dp == 51234 } {
  set elem [lsearch $antTab $sp]
  if { $elem < 0 } {
    if { $expldata == 2 } {
      # exploration ant in initialisation phase, expldata == 2
      # setup connection and read CE ants parameters
      set rho($sp) [lindex [split $dt /] 1]
      set beta($sp) [lindex [split $dt /] 2]
      #set explAnts($sp) [lindex [split $dt /] 3]
      puts "-------------------------------------------------------------------------------"
      puts "Connected by $target on port $sp "
      #puts "number of initial exploration ants = $explAnts($sp), "
      puts "and rho=$rho($sp) and beta=$beta($sp)."
      puts "-------------------------------------------------------------------------------"
      puts " "

      lappend antTab $sp
      set ga0($sp) 0
      set M($sp) 0
      set A($sp) 0
      set B($sp) 0
      set ga0tot($sp) 0
      set Mtot($sp) 0
      set Atot($sp) 0
      set Btot($sp) 0
    } else {
      puts stderr "Unknown ants dropped"
    }
  } else {
    # the connection exists
    # if route recorded, send back in reversed order
    if { $rl > 0 } {
      # reverse recorded route
      set last [expr "$rl / 4 - 2"]
      set route [split $rr /]
      set route [lrange $route 0 $last]
      set rr [join $route /]
      puts "Route: $rr"
      puts "Route joined $route"
      puts "Last $last"
      set rroute {}
      if ($last>-1) {
        # recorded route is set to be source route path
        set ind [expr $last-1]
        while {$ind>-1} {
          lappend rroute [lindex $route $ind]
          incr ind -1
```

```
    }
    # last hop is final destination
    lappend rroute $target
    # destination address is first hop
    set target [lindex $route $last]
    # source routing
    set rroute [join $rroute "/" ]
  }

  puts "Updated route: $rroute $target"
}

set tcost [hping getfield data str $pck]
set hcosts [split $tcost /]
if { $expldata == 2 } {
  # if this is initial exploration ant, remove expldata field and setup parameters
  set start 3
} else {
  # else remove only expldata field
  set start 1
}

set data [lindex $hcost $start]
set hcosts [lrange $hcosts [expr $start+1] end]

if { $data == 1 } {
  # add load to cost value for path pointed to by $rr
  # this makes the
  if { [info exists $load($rr) ] } {
    set load($rr) [expr $load($rr)*$alpha + (1-$alpha)*1]
  } else {
    set load($rr) 1
  }
} else {
  # read cost values for each hop, and accumlate to total sum
  set tcost [join $hcosts /]
  set hops [llength $hcosts]
  if {[expr $last+2]>$hops} {
     puts "HEI:::::$last $hops"
  } else {
  set cost 0
  foreach n $hcosts {
    set cost [ expr $cost+$n ]
  }

  # elite selection: if initial exploration ants or
  # if cost value is below given threshold (dynamically updated) => accept
  set thresh [expr -log($rho($sp))*$ga0tot($sp)]
  puts "Variables: $thresh $cost $expldata $sp $M($sp)"

  # update temperature over all observed ants
  #calculate temperature
  if {$Mtot($sp) == 0 | $ga0tot($sp) <= 0 } {set ga0tot($sp) [expr -$cost/log($rho($sp))]}
  set tmp [expr $cost/$ga0tot($sp)]
  Set tmpE [expr exp(-$tmp)]
  set ga0tot($sp) [expr ($Btot($sp) + $cost*$tmpE)/((1+$tmp)*$tmpE + ($Atot($sp) - $rho($sp)*((1-pow($beta($sp),$Mtot($sp)+1))/(1-$beta($sp)))
  set tmp [expr $cost/$ga0tot($sp)]
  set tmpE [expr exp(-$tmp)]
  set expCost $tmpE
  incr Mtot($sp) 1
  set Atot($sp) [expr ($Atot($sp) + (1+$tmp)*$tmpE)*$beta($sp)]
  set Btot($sp) [expr ($Btot($sp) + $cost*$tmpE)*$beta($sp)]

  if { $expldata == 2 || [expr  $cost - $thresh < $verysmall] } {
    puts "--> update sent to $target"
    #calculate temperature
    if {$M($sp) == 0 | $ga0($sp) <= 0 } {set ga0($sp) [expr -$cost/log($rho($sp))]}
    set tmp [expr $cost/$ga0($sp)]
    set tmpE [expr exp(-$tmp)]
    set expCost $tmpE
    set ga0($sp) [expr ($B($sp) + $cost*$tmpE)/((1+$tmp)*$tmpE + ($A($sp) - $rho($sp)*((1-pow($beta($sp),$M($sp)+1))/(1-$beta($sp)))))]
    set tmp [expr $cost/$ga0($sp)]
    set tmpE [expr exp(-$tmp)]
    incr M($sp) 1
    set A($sp) [expr ($A($sp) + (1+$tmp)*$tmpE)*$beta($sp)]
    set B($sp) [expr ($B($sp) + $cost*$tmpE)*$beta($sp)]
    set expldata 0
    set ga $ga0($sp)
  } else {
    # no update in routers but need data for animation of forward path
    set expldata 1
    set ga $ga0tot($sp)
  }

  # create and send packet
  set rpck "ip(daddr=$target,ttl=8)"
  if {$rr>0} {
   append rpck "+ip.lsrr(data=$rroute)"
```

```
      }

      # backward packet: set source port = 51234, and destination port = $sp of forwad packet
      append rpck "+udp(sport=$dp,dport=$sp)"
      if {$hops>1} {
        append rpck "+data(str=$expldata/$expCost/$cost/$ga/$beta($sp)/$accTime/$hops/$tcost/$rr)"
      } else {
        append rpck "+data(str=$expldata/$expCost/$cost/$ga/$beta($sp)/$accTime/$hops/$tcost)"
      }
      puts "Return: $rpck"
      hping send $rpck
      }
    }
  }
 }
}
```

D Demo start-up description

The demo consists of

- 10 LinkSys routers,
- 15 interconnection cables (7 red (short) and 8 gray (long)),
- 10 management cables (blue)
- 2 8-port switches
- 2 8-port conductor rail
- 10 power-supplies for LinkSys boxes
- 1 Mac PowerBook
- 1 Linux-PC

In Figure 22 the demo is illustrated, and in Figure 23 the details of network configurations and cable connections are given.

The following steps are necessary to start up the swarm routing demo.

1. Prepare the LinkSys boxes for click, see Section G.3.
2. Generate and move click configuration files to LinkSys boxes, see Section G.4.
3. Start up Ant-click router, see Section G.5.
   # click ant1.click
4. Start up AntPing receiver
   # bison-pc connected to node 3
   # address=10.3.11.2, default gw=10.3.11.1
   %> ifconfig eth0 10.3.11.2 gw 10.3.11.1
   # $HOME = ~poulh/hping/
   # $NAMHOME = $HOME/hping-nam
   %> cd $NAMHOME
   %> sudo hping3 exec recv-nam-v3.htcl eth0

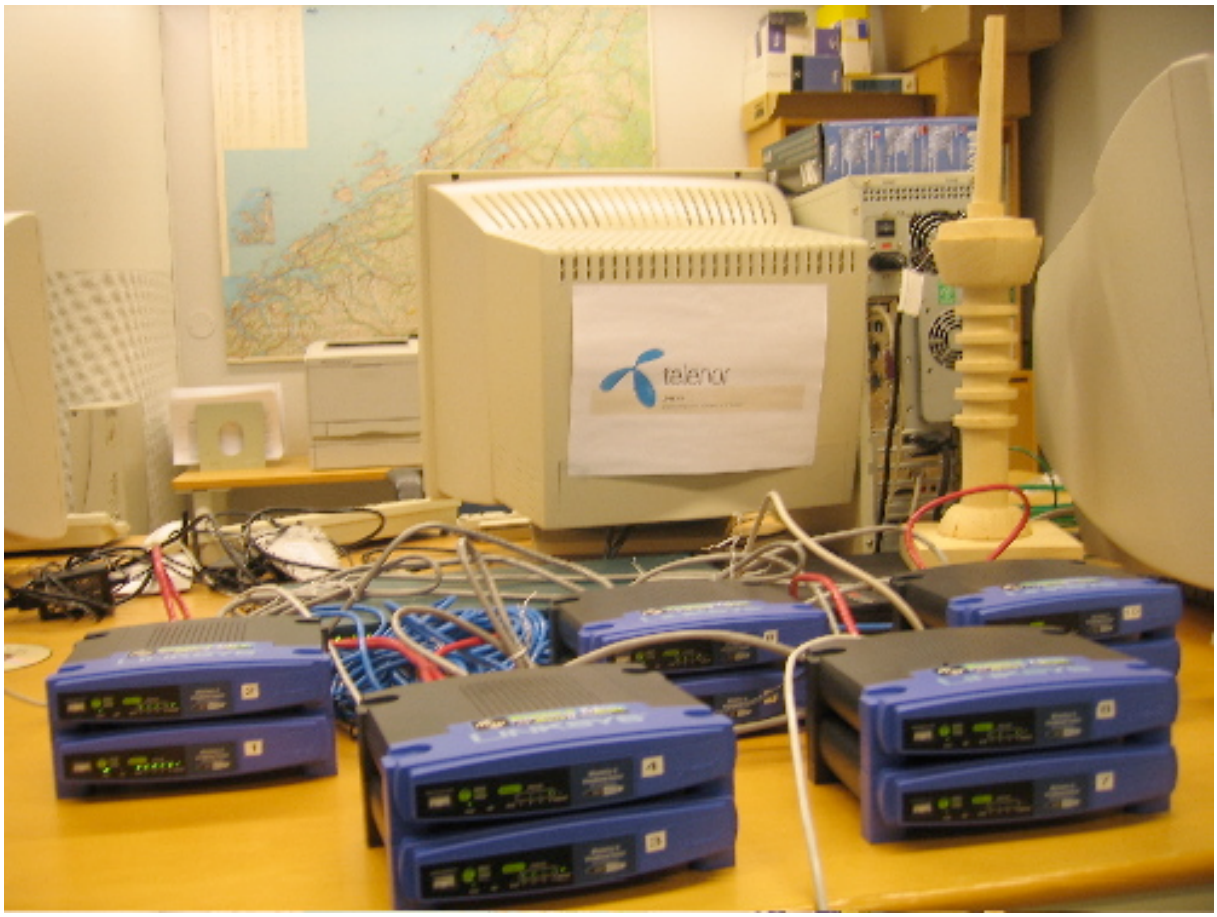


Figure 22: Picture taken at the lab illustrating the demo network

5. Start up AntPing sender
   # mac-box connected to node 8
   # address=10.8.11.2 (demo) with default gw=10.8.11.1
   # In “SystemTools” -> “Networks”: Select place “demo” and “activate” or:
   # $HOME = ~poulh/hping/
   # $NAMHOME = $HOME/hping-nam
   %> cd $NAMHOME
   %> sudo hping3 exec send-nam-v3.htcl 50000 en0 0

6. Run live animation
   # The “send-nam-v3.htcl” records packet events to nam-formatted tracefile “t50000.nam”
   # nam reads from the continuously updated tracefile.
   %> cd $NAMHOME
   %> tail -f -1000 t50000.nam | nam -r 50ms -

7. Plot data tracing and plot generation
   # “makeplot” calls “makegnuplot” which produces “plot.gnu”

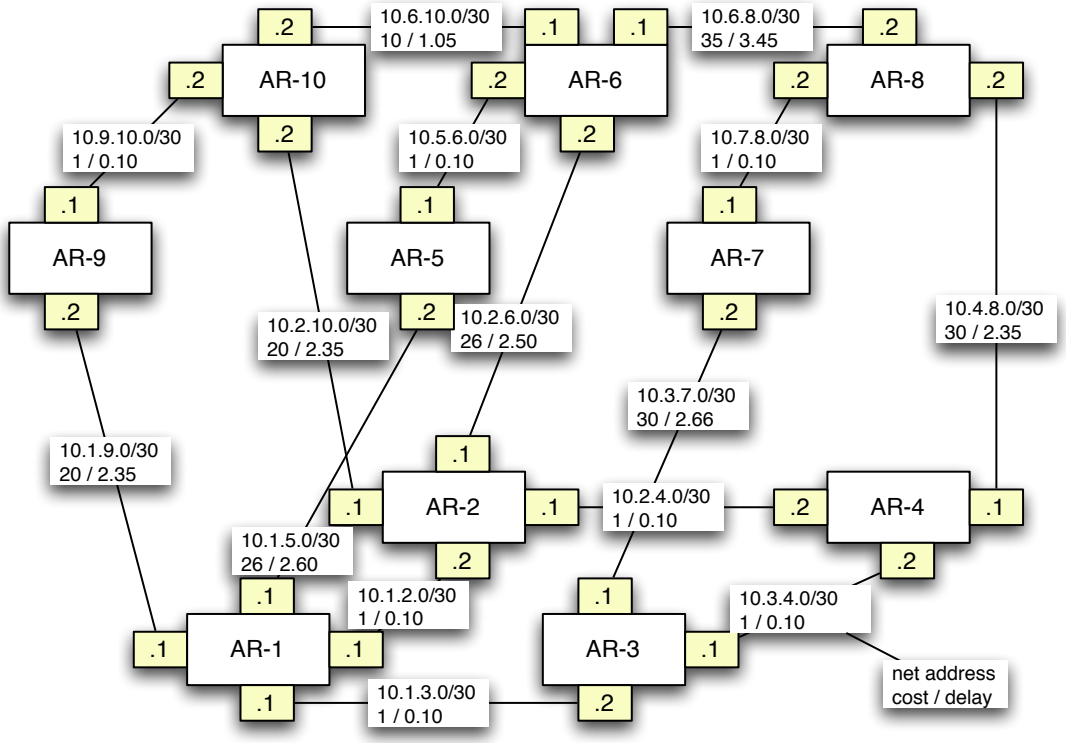


Figure 23: The network configuration

“makeplot” calls gnuplot with “plot.gnu” that reads trace of cost and temperatur data from “trace-50000”
“makeplot” calls gnuplot every second and reproduces graphical plots in png-format
$HOME = ~poulh/hping/
%> $HOME/makeplots 50000

8. View plots in web-browser
“makeplot” produces png-files that can be viewed through “plot.html”
the “plot will autorefresh every second
$HOME = ~poulh/hping/
open the file $HOME/plot.html in the web browser

E Installing OpenWRT on LinkSys WRT54G(S)

This is a brief description of how to flash “OpenWRT Linux” on LinkSys WRT54G(S) routers.

Note: All work was done on Linux, however it should be possible to do this from other OS’es as well

E.1 *About OpenWRT*

OpenWRT [A1] is a Linux distribution for wireless routers. Instead of trying to cram every possible feature into one firmware, OpenWRT provides only a minimal firmware with support for add-on packages. For users this means the ability to customize features, removing unwanted packages to make room for other packages. For developers this means being able to focus on packages without having to test and release an entire firmware.

OpenWRT comes in two versions: jjfs2 and squashfs. We are using squashfs because this is expected to be more stable and is said to have lower risk for turning the router into a brick when flashing. The squashfs versions give you slightly more jffs2 space and are capable of booting even when the jffs2 filesystem is broken or corrupted.

E.2 *Preparing Router for Accepting flash via tftp/thftp and boot_wait*

To minimize the risk of flashing the router to a useless paperweight, we have enabled boot_wait on the routers. Boot wait is done by factory to easy reflash bad routers in production line. Boot Wait is a function that halts the router to accept external flash of firmware to LAN1-port for some seconds on boot.

This function is disabled on routers when they hit the store. Due to some security flaw, it is still possible to re-enable boot_wait.

E.2.1 Boot_wait on routers *older than V4.0.*

Go to administration menu by addressing admin `http://192.168.1.1`. To activate *boot_wait* on LinkSys WRT54G and WRT54GS < V4.0, choose diagnostics from administration menu and execute the following commands one by one:

```
;cp${IFS}*/*/nvram${IFS}/tmp/n
;*/n${IFS}set${IFS}boot_wait=on
;*/n${IFS}commit
;*/n${IFS}show>tmp/ping.log
```

E.2.2 Boot_wait on WRT54GS > V4.0.

First downgrade the firmware to V1.05.2 (from LinkSys firmware download). Boot_wait may be enabled via console on a *perl* script [13]. Execute the commands in the following from the perl console. Activate boot_wait on LinkSys WRT54GS V4.0:

```
/usr/sbin/nvram set boot_wait=on
/usr/sbin/nvram commit
/usr/sbin/nvram get boot_wait
```

[13] See [A4] for information.

E.3 *Flashing the firmware*

Set static IP on the host computer for the flashing. In our case, the settings are applied:

IP: 192.168.1.x (1 < x < 255)

Mask: 255.255.255.0

GW: 192.168.1.1

Now, connect the host computer to LAN1 on the LinkSys router, and start *tftp* and *thftp* from a console: *tftp 192.168.1.1*. The sequence of command in the tftp console are

tftp> binary

tftp> rexmt 1

tftp> timeout 60

tftp> trace

Packet tracing on.

tftp> put openwrt-wrt54gs-squashfs.bin

Before the last step, you have to unpower the router, and power it again followed by *tftp> put openwrt-wrt54gs-squashfs.bin* instantly. If you receive an error, try again. If successful, the output should look like the following, and the router is rebooting. If there is a DMZ-indicator in front, it should light up during boot. When DMZ is off, you should be able to telnet to 192.168.1.1. Set root-password from terminal with the command *passwd root*.

.....

received ACK <block=3017>

sent DATA <block=3018, 512 bytes>

received ACK <block=3018>

sent DATA <block=3019, 0 bytes>

received ACK <block=3019>

.....

You should now have a barebone OpenWRT squashfs installed on your router.

F How to configure network interface on switch ports

Each switch port on the LinkSys router can be treated as a separate, individual Ethernet interface by the use of *robocfg* [A7]. The port is enable or disable by

robocfg port X state <enabled | disabled | rx_disabled | tx_disabled>

An example of a script setting all port on vlan0 is

robocfg switch disable

robocfg vlans enable reset

```
robocfg vlan 0 ports "0 1 2 3 4t"
robocfg switch enable
vconfig add eth0 0
ifconfig vlan0 hw ether XX:XX:XX:XX:XX:00
ifconfig vlan0 xx.xx.xx.xx netmask xx.xx.xx.xx
ifconfig vlan0 up
```

The configuration of MAC address is important when defining several vlans, see [A7].

G Installing Click on OpenWRT

Click [2.3.2] has to be compiled for the OpenWRT platform and the LinkSys processor (Broadcom CPU). The description in the following is based on Click OpenWRT `http://sarwiki.informatik.hu-berlin.de/Hacking_the_Netgear_wgt634u`. The router does not have enough memory to do the compiling itself, so a PC with a recent GNU/Linux distribution is necessary. Click can then be cross-compiled on this platform. The first thing we need is a build environment for OpenWRT.

G.1 OpenWRT Toolchain

You need a Linux computer with approximately 1GB free disk space to download and compile the OpenWRT source code and the toolchain. The following tools/libs are required:

- wget, tftp
- cvs, subversion
- gcc, gcc-c++, bison, flex
- patch, gettext
- autoconf, automake
- zlib-devel
- uClibc++

The OpenWRT toolchain can be downloaded with the following command:

```
cvs -d:pserver:anonymous@openwrt.org:/openwrt
logincvs -d:pserver:anonymous@openwrt.org:/openwrt co obsolete-buildroot
```

We need a C++-compiler, so you'll need to patch $OPENWRT/buildroot/Makefile.

```
INSTALL_LIBSTDCPP:=true
```

That's all: run make

G.2 Compiling Click

To build click you have to create the Makefile click.mk and put it in directory $OPENWRT/buildroot/make

Listing 5: Makefile click.mk

```
############################################################
#
# click 1.4.3
#
############################################################
CLICK_DIR:=$(BUILD_DIR)/click
CLICK_PREFIX:=$(BUILD_DIR)/staging_dir/bin
DEPLOY_DIR:=/tmp

click-source: $(CLICK_DIR)/.unpacked

$(CLICK_DIR)/.unpacked: $(DL_DIR)/$(CLICK_SOURCE)
        (cd $(BUILD_DIR); \
        svn co svn://sarsvn.informatik.hu-berlin.de/archives/click-1.4.3/ $(CLICK_DIR))
        touch $(CLICK_DIR)/.unpacked

$(CLICK_DIR)/.configured: $(CLICK_DIR)/.unpacked
$(CLICK_DIR); rm -rf config.cache; \
                $(TARGET_CONFIGURE_OPTS) \
                CFLAGS="$(TARGET_CFLAGS)" \
                CXXFLAGS="$(TARGET_CFLAGS) -I$(STAGING_DIR)/usr/include  -fno-builtin -nostdinc++ -nodefaultlibs"  \
                AR_CREATEFLAGS="cru" \
                ./configure \
                --build=i686-pc-linux-gnu \
                --host=mipsel-linux \
                --disable-linuxmodule \
                --enable-userlevel \
                --enable-local \
                --disable-aqm --disable-ip6 --disable-ipsec \
                --disable-grid --disable-bsdmodule --disable-radio \
                --disable-test --disable-wifi \
                --enable-tools=no \
                --prefix=$(CLICK_PREFIX) \
        );
        touch  $(CLICK_DIR)/.configured

$(CLICK_DIR)/click: $(CLICK_DIR)/.configured
        $(MAKE) CC=$(TARGET_CC) -C $(CLICK_DIR) \
        LIBS="-L$(STAGING_DIR)/lib -fno-builtin -nostdinc++ -nodefaultlibs -luClibc++ -lc -lm -lgcc"

$(CLICK_PREFIX)/bin/click: $(CLICK_DIR)/click
        $(MAKE) CC=$(TARGET_CC) -C $(CLICK_DIR) install

click: $(CLICK_PREFIX)/bin/click

click-clean:
        $(MAKE) -C $(CLICK_DIR) clean
        rm -rf $(TARGET_DIR)/usr/bin/click

click-dirclean:
        rm -rf $(CLICK_DIR)
```

We also need the Bison elements. Copy the following files to $OPENWRT/buildroot/click/elements/local:

- anttable.cc/hh
- anttrafficmonitor.cc/hh
- backwardant.cc/hh
- forwardant.cc/hh
- sourceroute.cc/hh

They will then be compiled into the click binary file.

That's all: run make click to build click.

The Click binary will end up in $OPENWRT/build_mipsel/staging_dir/bin/bin/ Copy it to directory /sbin on the router.

G.3 Prepare the router for Click

There are a few steps necessary before we can run Click.

Download and install uClibc++ on the router:

```
wget uclibc++_0.1.11-2_mipsel.ipk
ipkg install uclibc++_0.1.11-2_mipsel.ipk
```

Remove unnecessary services (to save memory space):

```
rm /etc/init.d/S45firewall
rm /etc/init.d/S50httpd
rm /etc/init.d/S50telnet
rm /etc/init.d/S50dnsmasq
```

We should also configure some network parameters that will make it easier to access the router from a separate network:

```
nvram set wan_ipaddr=10.100.1.x (1< x < 255)
nvram set wan_proto=static
nvram set wan_gateway=10.100.1.1
nvram set wan_netmask=255.255.255.0
nvram set wl0_radio=0nvram commit
```

We can then connect an external computer to the ?Internet? port on the router and use the 10.100.1.0 segment to do maintenance, logging, debugging etc.

G.4 Generate Click configuration file

Click must be run with a configuration file that tells it what to do when packets arrive on the different interfaces as described in 2.4. This file will be different on each router, and can easiest be generated with a Perl script.

Set the following environment variable (example from the setting on node 1 in our demo):

```
$ifs = [ [ "vlan3", 26, 3, "10.1.5.1", "255.255.255.0", "00:0A:0A:01:05:01" ],
    [ "vlan4", 1, 2, "10.1.2.1", "255.255.255.0", "00:0A:0A:01:02:01" ],
    [ "vlan5", 1, 1, "10.1.3.1", "255.255.255.0", "00:0A:0A:01:03:01" ],
    [ "vlan6", 20, 0, "10.1.9.1", "255.255.255.0", "00:0A:0A:01:09:01" ],
   ];
```

This array contains four rows with the variables: interface name, cost value, physical port number, ip-address, netmask, mac-address.

Then run the script

```
# make-router.pl > ant1.click
```

The click file must be copied to the router. Generate one such file for each router.

G.5 Run Click

Log in to the router with ssh. Start Click like this:

```
# click ant1.click
```

References

[A1] OpenWRT main site `http://openwrt.org/`

[A2] OpenWRT Documentetion `http://wiki.openwrt.org/OpenWrtDocs`

[A3] OpenWRT forums `http://forum.openwrt.org/`

[A4] OpenWRT howto `http://wiki.openwrt.org/OpenWrtHowTo`

[A5] OpenWRT Faq `http://wiki.openwrt.org/Faq`

[A6] OpenWRT `http://www.cinnaman.info/mediawiki/index.php/OpenWRT`

[A7] robocfg `http://wiki.openwrt.org/OpenWrtDocs/Configuration#head-92940f6a6b5db4331a641f53a9f40ddbd3a2f505`