Simple Ant Routing Algorithm - SARA
NS2 install procedures

Fernando Jorge Ribeiro Correia
fcorreia@tagus.inesc-id.pt

January 5, 2012

1 Introduction

The *Simple Ant Routing Algorithm* (SARA) offers a low overhead solution, by optimizing the routing process. Three complementary strategies were used in our approach: during the route discovery we have used a new broadcast mechanism, called the *Controlled Neighbor Broadcast* (CNB), in which each node broadcasts a control message (FANT) to its neighbors, but only one of them broadcast this message again. During the route maintenance phase, we further reduce the overhead, by only using data packets to refresh the paths of active sessions. Finally, the route repair phase is also enhanced, by using a deep search procedure as a way of restricting the number of nodes used to recover a route. Thus, instead of discovering a new path from the source to the destination, we start by trying the discovery of a new path between the two end-nodes of the broken link. A broadest search is only executed when the deeper one fails to succeed.

SARA can be tested with different types of traffic. However, due the route refresh requirements, in which the pheromones are re-enforced by the packets that goes through the links between nodes, when is used traffic with asymmetric behavior, i.e., the packet flow rate is different in both directions (from source to destination and from destination to source), the pheromone value can supply a wrong information about the link quality. This situation can be experimented when is simulated Constant Bit Rate traffic (CBR). To overcame this situation, it was created a simple data packet class called TX_CBR. The TX_CBR is a CBR service in which when a packet arrives at the destination node is re-transmitted to the source node. This way is ensured a symmetric kind of data traffic. This TX_CBR class can also support connection to multiple sources.

The next section, the user will be guided through the procedures to install SARA and TX_CBR in NS2. This two classes must be installed.

2 Install procedure

For SARA works properly in NS2 it must expanded in your NS2 working directory the following files:

- sara.Vrs2.2.8.A.tar.gz

- tx_cbr.V1.tar.gz
- aux_obj.tar.gz

These three files must create three folders named as **sara**, **tx-cbr** and **ara**. In these folder will be placed the sara core files, the tx_cbr core files and the auxiliary classes required by sara. The named **ara** is related to an early sara developments. This files are responsible to do the following procedures:

Files	**Description**
sara/sara.cc	main class which includes all procedures to discover, select, maintain, and repair the routes.
sara/sara_ngh.cc	creates an object for each node in the neighborhood. Has ara_base as root object.
sara/sara_rt.cc	creates an object for each route found by each network node. sara_rt has a pointer to a list of the sara_rotas object. Has ara_base as root object.
sara/sara_rotas.cc	for each known route, sara_rotas object indicate which is the link for the next hop. Has ara_base as root object.
sara/sara_seq_num.cc	creates an object for each route discovery procedures. This object will keep track of the FANT agents sent to the network. Has ara_base as root object.
sara/sara_session.cc	it is used identify which are the route repair procedure active. Has ara_base as root object.
sara/sara_pkt.h	definitions used in the SARA control agents - FANT, BANT, RFANT, RBANT, RRERROR
ara/ara_base.cc	this is the top class for this architecture. It works with ara_vector, in which is implemented a list of ara_base object.
ara/ara_vector.cc	this class that implements a list of ara_base objects, i.e., it can hold different list, one for each sara class.
ara/ara_alg.cc	this class has several general purpose function.
ara/ara_rqueue.cc	this class implements a temporay queue to keep the data packets transported by TCP while SARA is discovering / repairing the route
ara/ara_path.cc	this is used to record information about the path used just for statistical information. Has ara_base as root object.
tx_cbr/tx_cbr.cc	the tx_cbr class allows SARA to implement CBR traffic in both directions. The tx_cbr works directly over IP. It allows to connect multiple sources to one destination node on a wireless environment.

3 Things to change

This code was developed to work with NS2.31. There are some changes we need to do in order to integrate this code inside simulator.

3.1 Packet type declaration

The packet type declaration is defined inside the file `common/packet.h`. In here we must include the name of our packet type. Find the `packet_t` enumeration and add the following lines (2 and 3):

common/packet.h

```
        // insert new packet types here
        PT_SARA,
        PT_TXCBR,
        PT_NTYPE // This MUST be the LAST one
};
```

Just below in same file, there is the definition of `p_info` class. Inside the constructor we will provide a textual name for these packet type (lines 4 and 5).

common/packet.h

```
1  class p_info {
2    public:
3          p_info(){
4                name_[PT_TXCBR]= strdup("tx_cbr");
5                name_[PT_SARA]= strdup("sara");
6                name_[PT_TCP]= strdup("tcp");
7                name_[PT_UDP]= strdup("udp");
```

Also in file `common/packet.h` is required to add the following lines to `hdr_cmn`. Look for `struct hdr_cmn` and insert:

common/packet.h

```
struct hdr_cmn {
    enum dir_t { DOWN= -1, NONE= 0, UP= 1 };
    packet_t ptype_;    // packet type (see above)
    // change by: Fernando Correia
    int     pid_;        // path ID
    int     psn_;        // packet serial number
    int     pts_;        // packet time stamp
    inline  int& pid()  return (pid_);
    // change by: Fernando Correia
    int    size_;        // simulated packet size
    int    uid_;         // unique id
    int    error_;       // error flag
    ...
}
```

These variables don't change SARA's behaviour, it is just control information for my simulations. You can check the proposed changes in `sara.cc`.

3.2 Tracing support

The simulations aim is to get a trace file describing what happended during execution. To write information about the protocol state it is necessary to use the `Trace` object. To log information regarding our packet type we implement the format `protoname()` function inside the CMUTrace class. For do that we must edit the `trace/cmu-trace.h/cc` and add the proper code. First we change the `trace/cmu-trace.h` by adding lines 6 and 7.

trace/cmu-trace.h

```
1 class CMUTrace : public Trace {
2 public:
3         CMUTrace(const char *s, char t);
4         /* ... */
5         void  format_aodv(Packet *p, int offset);
6         void  format_txcbr(Packet *p, int offset);
7         void  format_sara(Packet *p, int offset);
8 };
```

and in the `trace/cmu-trace.cc`, we must add lines above:

trace/cmu-trace.cc

```
1 #include <sara/sara_pkt.h>
2 #include <tx_cbr/tx_cbr.h>
3 /* ... */
4 void
5 CMUTrace::format_txcbr(Packet *p, int offset)
6 {
7         struct hdr_cbr* cbrh = HDR_CBR(p);

8         sprintf(pt_->buffer() + offset, "-P TxCBR SRC=%d DST=%d SN=%d LEN=%d NHOPS=%d",
9                 cbrh->saddr, cbrh->daddr, cbrh->id, cbrh->len, cbrh->nhop);
10 }

11 void
12 CMUTrace::format_sara(Packet *p, int offset)
13 {
14         struct hdr_sara* ah = HDR_SARA(p);

15         if(pt_->tagged()){
16                 sprintf(pt_->buffer() + offset,
17                         "-sara:s %d -sara:d %d -sara:t %d -sara:l %d ",
18                         ah->sara_src(),
```

```
19                          ah->sara_dst(),
20                          ah->sara_type(),
21                          ah->seq_num);
22          }
23          else{
24                  switch(ah->sara_type()){
25                  case SARA_HELLO:
26                          sprintf(pt_->buffer() + offset,
27                                  "-P sara_hello -s %d -d BROADCAST",
28                                  ah->sara_prev());
29                          break;
30                  }
31          }
32 }
33 /* ... */
```

the trace information that was add is very simple, but it is always possible to increase the trace information. To call these functions, we also must add the lines 6-11 in `format()` function in `trace/cmu-trace.cc` file.

trace/cmu-trace.cc

```
1 void
2 CMUTrace::format(Packet* p, const char *why)
3 {
4     /* ... */
5     case PT_PING:
6     case PT_SARA:
7          format_sara(p, offset);
8          break;
9     case PT_TXCBR:
10         format_txcbr(p, offset);
11         break;
12    /* ... */
13    default:
14    /* ... */
15 }
```

3.3 Tcl library

To configure SARA and TX_CBR working parameters we need to do some changes in tcl files. In `tcl/lib/ns-packet.tcl` is located the next code that we must change (lines 3 and 4):

tcl/lib/ns-packet.tcl

```
foreach prot {
# ...
        SARA
        TXCBR
# ...
} {
        add-packet-header $prot
}
```

Default values for binded attributes have to be given inside `tcl/lib/ns-default.tcl`. We must go to the end of the file and write the next code:

tcl/lib/ns-default.tcl

```
Agent/SARA set F_ 1;                 #convergence factor
Agent/SARA set ph_valid_ 1.0;        #pheromone time life
Agent/SARA set rand_seed_ 0;         #random seed value - '0'-random
Agent/SARA set bcast_limit_ 2;       #DSA broadcast limit
Agent/SARA set hello_int_ 5.0;       #HELLO msg interval
Agent/SARA set rd_int_ 0.5;          #FANT generation interval - T0
Agent/SARA set rr_int_ 1;            #route repair timer
Agent/SARA set sd_int_ 0.2;          #FANT confirmation interval - T1
Agent/SARA set retry_ 5;             #max tx atemps
Agent/SARA set ph_max_level_ 50; #default value for max value allowed for pheromon level
Agent/SARA set metrica_ 0;           #metric to use: 0-dist/pheromone (only)
Agent/SARA set gps_ 0;               #tx GPS msg: '0'-no; '1'-yes
Agent/SARA set delta_ 0.1;           #link recuperation index
Agent/SARA set log_cbr_ 0.5;         #
Agent/SARA set ph_mod_ 3;            #pheromone management model
Agent/SARA set taxa_evaporacao_ 0.5; # pheromone evaporation rate for AS model

Agent/TxCBR set packet_size_ 1000;   #packet lebght
Agent/TxCBR set txInt_ 0.16;         #160ms each 1000B = 50kbps
```

in `tcl/lib` files, we also need to perform a few changes in `ns-lib.tcl`. In this file we must add the procedures to create the SARA route agents. To do that, first change `ns-lib.tcl`:

tcl/lib/ns-lib.tcl

```
Simulator instproc create-wireless-node args {
# ...
        switch -exact $routingAgent_ {
                SARA {
```

```
5                       set ragent [$self create-sara-agent $node]
6               }
7               # ...
8       }
9       # ...
10}
```

then create the agent as coded below

tcl/lib/ns-lib.tcl

```
1 Simulator instproc create-sara-agent { node } {
2         # Create a sara routing agent for this node
3         set ragent [new Agent/SARA [$node node-addr]]
4         $self at 0.001 "$ragent start"
5         $node set ragent_ $ragent
6         return $ragent
7         }
```

3.4 Priority queue

Each routing protocol message must have a priority level. It was choose to use `PriQueue` to treat the control packets. In `PriQueue` we must add line 11:

queue/priqueue.cc

```
1 void
2 PriQueue::recv(Packet *p, Handler *h)
3 {
4         struct hdr_cmn *ch = HDR_CMN(p);

5         if(Prefer_Routing_Protocols) {
6                 switch(ch->ptype()) {
7                     case PT_DSR:
8                     case PT_MESSAGE:
9                     case PT_TORA:
10                    case PT_AODV:
11                    case PT_SARA:
12                        recvHighPriority(p, h);
13                        break;

14                    default:
15                        Queue::recv(p, h);
16                }
```

```
17          }
18          else {
19                  Queue::recv(p, h);
20          }
21  }
```

3.5 GOD class

SARA is an inter-layer communication protocol. It requires information about the neighbor nodes that can be sent by MAC layer. In NS2 I simulate this functionality by changing GOD class behavior. Do the following changes in `mobile/god.h|cc`:

mobile/god.h

```
1  class God : public BiConnector {
2  public:
3          God();
4  ...
5          int getMyTopGrid(double x, double y);
6          int getMyBottomGrid(double x, double y);

7          vector getNodeCoord(int id);   // add this line

8          inline int getMyGridSize() {
9          return gridsize_;
10     }

11    // -----------------------
12 private:
13 ...
14 }
```

mobile/god.cc

```
1  vector
2  God::getNodeCoord(int id)
3  {
4          assert(id < num_nodes);
5          mb_node[id]->update_position();
6          vector a(mb_node[id]->X(), mb_node[id]->Y(), mb_node[id]->Z());

7          return a;
```

```
8 }



9 bool God::IsNeighbor(int i, int j)
10 {
11   assert(i<num_nodes && j<num_nodes);

12   //printf("i=%d, j=%d                          n", i,j);
13   /* comment the following lines
14   if (mb_node[i]->energy_model()->node_on() == false ||
15       mb_node[j]->energy_model()->node_on() == false ||
16       mb_node[i]->energy_model()->energy() <= 0.0 ||
17       mb_node[j]->energy_model()->energy() <= 0.0 ) {
18     return false;
19   }
20   */

21   mb_node[i]->update_position();
22   mb_node[j]->update_position();

23   vector a(mb_node[i]->X(), mb_node[i]->Y(), mb_node[i]->Z());
24   vector b(mb_node[j]->X(), mb_node[j]->Y(), mb_node[j]->Z());
25   vector d = a - b;

26   if (d.length() < RANGE)
27     return true;
28   else
29     return false;
30 }
```

3.6 Makefile

Now everyting is implemented and we only need to compile it. To do so we will edit Makefile file by adding our **SARA** and **TX_CBR** files inside `OBJ_CC` variable as in following code (lines 4).

Makefile

```
1 OBJ_CC = \
2         sara/sara_session.o sara/sara_seqnum.o sara/sara_rt.o \
3         sara/sara_rotas.o sara/sara_ngh.o sara/sara.o \
```

```
4         ara/ara_alg.o ara/ara_base.o ara/ara_path.o \
5         ara/ara_vector.o ara/ara_rqueue.o \
6         tx_cbr/tx_cbr.o \
7 # ...
```

to compile the code there are only two things that remain. First it is necessary to do:

```
[ns-2.31]$ touch common/packet.cc
```

and then:

```
[ns-2.31]$ make
```

this should do the work and the code must compile.