# *U*niversità degli *S*tudi di *U*rbino

**Facolta' Di Scienze Matematiche Fisiche e Naturali**
**C. D. L. In informatica applicata**

Corso di *Sistemi di Comunicazione Multimediali*

# Localizzazione di una stazione mobile wireless basata sulla vicinanza a stazioni base

## ***Progetto e Relazione***


Studente
Elvis Ciotti

Ing.
Andrea Acquaviva


5 / 4 / 2005

**Tipologia del progetto:** simulazione wireless in ns2 della localizzazione approssimativa di una stazione (nodo) mobile basata sul rilevamento per vicinanza rispetto a stazioni posizionate in coordinate note (stazioni base).

**Descrizione:**
Nello scenario wireless, dopo aver impostato una griglia di stazioni base, la stazione mobile si muove in più posizioni casuali e manda in broadcast la richiesta alle stazioni base di rispondere con la propria posizione.
In base ai parametri impostati per la densità della griglia e alla potenza limitata sui nodi (e quindi distanza di rilevamento), possono rispondere 0,1 o più nodi alla richiesta. In tal modo di riesce a rilevare la posizione approssimativa della stazione mobile.
Per problematiche relative al simulatore[1] non è stato possibile implementare un algoritmo di localizzazione, basato sul tempo di volo della risposta e successiva triangolazione.

Note tecniche:
Il simulatore utilizzato è ns[2], versione2.27 su Mandrake Linux (Kernel v2.6)

# IMPLEMENTAZIONE[3]

## Impostazione del layout

Variabili da settare per la costruzione della griglia:

| | |
|---|---|
| val(nnx) | Numero di nodi nell'asse X |
| val(nny) | Numero di nodi nell'asse Y |
| val(gridx) | Distanza orizzontale fra i nodi della griglia |
| val(gridy) | Distanza verticale fra i nodi della griglia |

Variabili dipendenti:

| | |
|---|---|
| val(nn) | Numero di nodi totali (ottenute dal prodotto delle precedenti) |
| val(x) | Ottenuta in base al numero di nodi orizzontali e distanza fra loro,con cornice esterna di 5 |
| val(x) | Ottenuta in base al numero di nodi verticali e distanza fra loro,con cornice esterna di 5 |

Le variabili dipendenti sono settate nel seguente modo

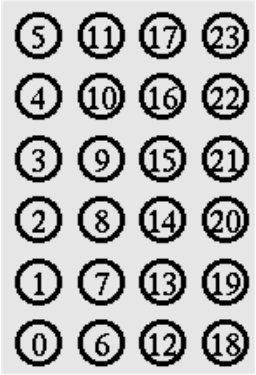
```
set val(nn)             [ expr ( $val(nnx) * $val(nny) )]
set val(x)              [ expr ( $val(nnx) * $val(gridx) ) + 10 ]
set val(y)              [ expr ( $val(nny) * $val(gridy) ) + 10 ]
```

Con un ciclo, viene creata la griglia :

```
set i 0
for {set x 0} {$x < $val(nnx) } {incr x} {
  for {set y 0} {$y < $val(nny) } {incr y} {

     # setto nodo e dimensione
     set node_($i) [$ns_ node]

     # setto coordinate a griglia
     $node_($i) random-motion 0          ;# disable random motion
     $node_($i) set X_ [ expr (5 + ($x * $val(gridx))) ]
     $node_($i) set Y_ [ expr (5 + ($y * $val(gridy))) ]
     $node_($i) set Z_ 0.0
     $ns_ initial_node_pos $node_($i) 10 ;# dopo settaggio coordinate, imposto dimensioni
                                            nodi nel simulatore NAM

     set i [expr $i + 1] ;# incrementa i (quindi va da 0 a [nnx * nny])
  }
}
```

[1] Sembra che nel simulatore non ci sia un modello preciso del ritardo, che viene mascherato dal rumore introdotto casualmente. Si ottengono quindi Rtt non proporzionati alla distanza fisica fra i nodi, rendendo impossibile implementare un calcolo della posizione.
[2] Sito ufficiale http://www.isi.edu/nsnam/ns
[3] Si riportano le porzioni di codice rilevante e relativa descrizione. Il codice completo è presente alla fine di questa relazione.

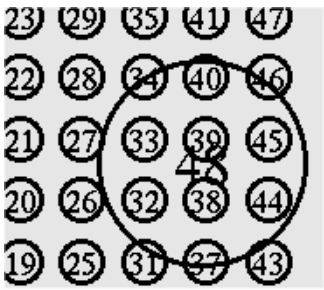
Esempio di griglia con 4 x 6 nodi visualizzata nel tool grafico NAM




| | |
|---|---|
| val(nnx) | 4 |
| val(nny) | 6 |
| val(nn) | 4 x 6 = 24 |
| val(gridx) | 15 |
| val(gridy) | 15 |
| val(x) | 4 x 15 + 10 = 70 |
| val(y) | 6 x 15 + 10 = 100 |

Il settaggio del nodo mobile avviene impostando coordinate a zero e dimensione iniziale di 50 per distinguerlo nella griglia (vedi fig. a destra)

```
set nodo_mobile [$ns_ node]
#
$nodo_mobile random-motion 0
$nodo_mobile set X_ 0.0
$nodo_mobile set X_ 0.0
$nodo_mobile set Z_ 0.0
#
$ns_ initial_node_pos $nodo_mobile 50
```




## Configurazione dei nodi

```
set val(chan)           Channel/WirelessChannel    ;# Tipo canale
set val(prop)           Propagation/TwoRayGround   ;# Modello propagazione
set val(netif)          Phy/WirelessPhy            ;# tipo interfaccia di rete
set val(mac)            Mac/802_11                 ;# MAC
set val(ifq)            Queue/DropTail    ;# tipo di coda
set val(ll)             LL                         ;# tipo di link layer
set val(ant)            Antenna/OmniAntenna        ;# modello di antenna
set val(ifqlen)         2
set val(rp)             DSDV              ;# protocollo di routing

$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON \
-channel $chan_1_ \
```

La limitazione della potenza del segnale fino a 10 mt si ottiene impostando `Pt_` dell'interfaccia di rete circa al valore sottostante:

```
Phy/WirelessPhy set Pt_ 0.0000536
```

## Creazione dell'agente e applicazione ai nodi

Il funzionamento dell'agente è simile a quello Ping[4] e proprio dalla modifica di questo si è ottenuto il nuovo agente utilizzato. Vengono di seguito riportate solo le modifiche fatte all'agente e non il codice completo dell'agente Ping.

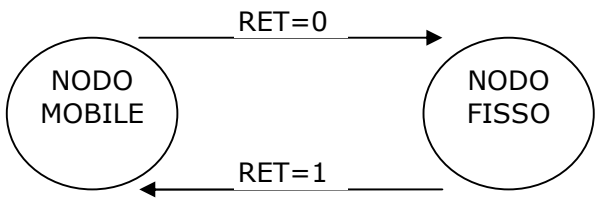
- Rispetto all'agente Ping Il pacchetto conterrà in più le informazioni relative alle coordinate che la stazione base inserisce.

```
struct hdr_ping {
      char ret;
      double send_time;
      double rcv_time;
      int seq;
      double cx, cy;
            …
}
```

- Il comando di invio (dal nodo mobile) crea il pacchetto, setta `ret` a zero ( segnala il pacchetto come inviato dal nodo mobile, in modo che le stazioni fisse capiscano che in tal caso lo devono riempire con le loro coordinate) e setta inizialmente le coordinate a zero.

```
int PingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the Ping header for the new packet:
      hdr_ping* hdr = hdr_ping::access(pkt);
      // Set the 'ret' field to 0, so the receiving node
      // knows that it has to generate an echo packet
      hdr->ret = 0;
      hdr->cx = 0;
      hdr->cy = 0;
      hdr->seq = seq++;
      hdr->send_time = Scheduler::instance().clock();
      send(pkt, 0);
      return (TCL_OK);

    }
     …
```

- Quando le stazioni fisse ricevono il pacchetto con `ret`=0, ricreano un nuovo pacchetto contenente coordinate e tempo di ritorno[5], poi settano il `ret` a 1 (a indicare il pacchetto di ritorno alla stazione mobile legge) e rinviano il pacchetto.




La coordinate del nodo corrente si calcolano tramite l'indirizzo (valore intero 0,1…) del nodo nel simulatore (dall'array `NodeShift_[]`), che passato alla funzione `get_node_by_address`, permette di utilizzare la funzione `getLoc` ed ottenere le coordinate del nodo. Le coordinate vengono scritte nella variabili *cx* e *cy.*

```
if (hdr->ret == 0) {
    double stime = hdr->send_time;
    int rcv_seq = hdr->seq;


    Packet::free(pkt);
    Packet* pktret = allocpkt();
```

[4] implementazione già presente nel simulatore , contenuta nei files `~/apps/ping.cc` e `~/apps/ping.h`). Da ricompilare una volta modificati con i comandi *make depend* e *make*, dalla root di ns.

[5] non rilevante in questa tipologia di progetto, ma mantenuto per eventuali future modifiche in quanto già presente nell'agente Ping

```
    hdr_ping* hdrret = hdr_ping::access(pktret);
    hdrret->ret = 1;

    double x,y,z;
    int nodocorr = hdrip->src_.addr_ >> Address::instance().NodeShift_[1];
    nsaddr_t t;
    MobileNode *test = (MobileNode *) (Node::get_node_by_address(nodocorr));

    test->getLoc(&x,&y,&z);

    hdrret->cx = x;
    hdrret->cy = y;

    hdrret->send_time = stime;
    hdrret->rcv_time = Scheduler::instance().clock();
    hdrret->seq = rcv_seq;
    send(pktret, 0);
  } else { … }
```

Nel caso in cui `ret` non è uguale a zero, cioè il pacchetto ritorna al nodo mobile, si deve stampare a video:

- da che nodo proviene la risposta ( `hdrip->src_.addr_ >> Address::instance().NodeShift_[1]` )
- quali sono le sue coordinate ( `hdr->cx, hdr->cy` )
- tempo di ritorno ( `Scheduler::instance().clock()-hdr->send_time) * 1000` )

Le informazioni sono passate a Tcl tramite la `tcl.eval()`

```
if (hdr->ret == 0) {
    . . .
  } else {
    char out[100];
    [ . . ]
          sprintf(out, "%s recv %d %3.1f %.0lf %.0lf", name(),
          hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
          (Scheduler::instance().clock()-hdr->send_time) * 1000, hdr->cx, hdr->cy);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Discard the packet
    Packet::free(pkt);
  }
```

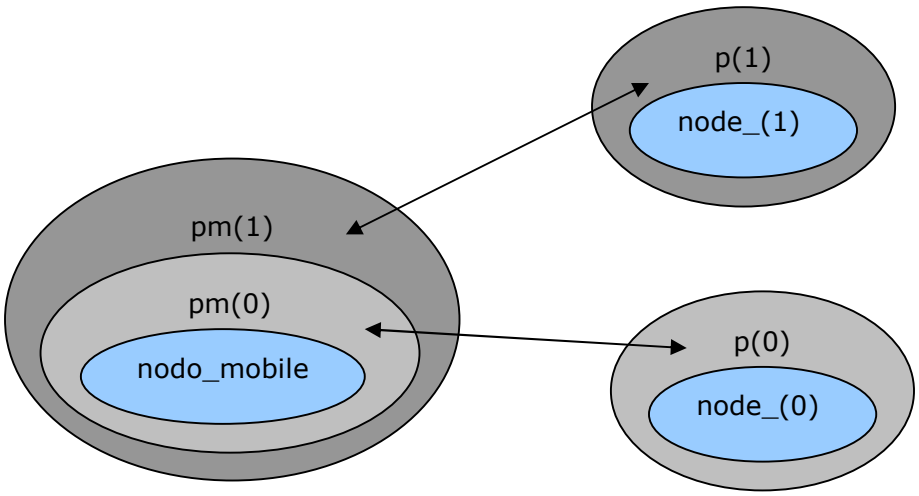
La seguente importazione dell'agente in Tcl permette di leggere il nodo da cui è provenuta la risposta ($from), le sue coordinate ($cx e $cy) e il tempo di risposta.

```
Agent/Ping instproc recv {from rtt cx cy} {
       $self instvar node_
       puts "  Nelle vicinanze rilevo la stazione $from ($cx,$cy). RTT $rtt ms"
}
```

- L'agente creato viene poi applicato ai nodi. In alternativa alla modifica delle impostazioni dei pacchetti in volo in broadcast, al nodo mobile viene attaccato un agente per ogni altro nodo presente sulla griglia[6]. In questo modo il nodo mobile può comunicare con tutti i nodi della griglia (se a distanza di rilevamento). Se al nodo fisso node_(i) è attaccato l'agente p(i), al mobile è attaccato il corrispondente agente pm(i), p(i) e node_(i) vengono collegati, in modo che il nodo mobile e il nodo n(i) possano comunicare.

```
set i 0
for {set x 0} {$x < $val(nnx) } {incr x} {
  for {set y 0} {$y < $val(nny) } {incr y} {

       # agenti sui nodi fissi
      set p($i) [new Agent/Ping]
      $ns_ attach-agent $node_($i) $p($i)
      #
      # creo agente corrispondente sul nodo mobile
      set pm($i) [new Agent/Ping]
      $ns_ attach-agent $nodo_mobile $pm($i)
      #
      #attacco i corrispondenti
      $ns_ connect $p($i) $pm($i)
      #
      set i [expr $i + 1] ;# incrementa i (quindi va da 0 a [nnx * nny])
  }
}
```

Esempio di applicazione di agenti in un layout con due nodi




[6] Suggerimento proveniente dalla mailing list ufficiale di ns

## Posizionamenti del nodo mobile

Variabili da settare:

| | |
|---|---|
| val(pos) | n. di posizionamenti e richieste di posizione fatte dal nodo mobile |
| val(intervallo) | intervallo temporale fra i posizionamenti successivi |

Variabili dipendenti:

| | |
|---|---|
| val(stop) | tempo di stop per la simulazione, in base alle due variabili precedenti |

Impostazione variabili random[7] per posizionamenti all'interno della griglia

```
global defaultRNG
$defaultRNG seed 0
#
set rngx [new RNG]
set RVx [new RandomVariable/Uniform]
$RVx set min_  1
$RVx set max_  $val(x)-1
$RVx use-rng $rngx
#
set rngy [new RNG]
set RVy [new RandomVariable/Uniform]
$RVy set min_  1
$RVy set max_  $val(y)-1
$RVy use-rng $rngx
```

La serie di posizionamenti avviene ad intervalli e numero di volte impostati, posizionando ogni volta il nodo alle coordinate casuali ($px, $py)[8] e stampandole a video. A questo punto il nodo mobile invia a tutti la richiesta, ovvero tutti gli agenti applicati al nodo mobile inviano ai rispettivi agenti il pacchetto sopraindicato.

```
for {set t 1} {$t <= $val(pos) } {incr t} {

   set px [expr round([$RVx value])]
   set py [expr round([$RVy value])]

   # mi sposto e invio (cioè tutti gli agenti attaccati sopra il mobile
   #  inviano a tutti  gli altri corrispettivi)
   set tempo  [ expr ( $val(intervallo) * $t ) ]
   set tempo1 [ expr ( $tempo + 1 ) ]
   set tempo2 [ expr ( $tempo + 2 ) ]

   $ns_ at $tempo    "puts \"\nt=$tempo: Vado nelle coordinate ($px $py)\""
   $ns_ at $tempo.1  "$nodo_mobile setdest $px $py 9999.0"

   #tutti gli agenti del mobile fanno richiesta posizione
   for {set i 0} {$i < $val(nn) } {incr i} {
      $ns_ at $tempo2 "$pm($i) send"
   }
}
```

La simulazione termina alla fine dei posizionamenti e invio delle richieste.

```
set val(stop)           [expr ( $val(pos)+1) * $val(intervallo) ]
```

[7] Manuale cap. 22

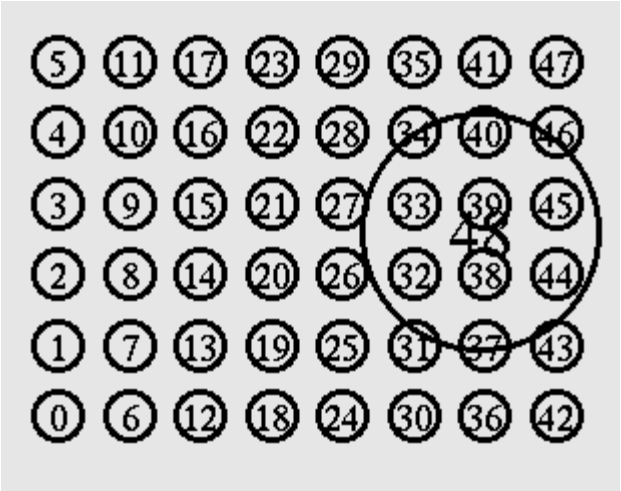
[8] Non essendo la velocità  rilevante, è impostata ad un valore relativamente alto.

**Esempio di output**

**SHELL**

```
*******************************************
   Layout di 130 x 100 - 8 x 6 nodi - griglia
di 15 x 15
*******************************************
num_nodes is set 49
INITIALIZE THE LIST xListHead
Starting Simulation...
channel.cc:sendUp - Calc highestAntennaZ_ and
distCST_
highestAntennaZ_ = 10.0,  distCST_ = 430.6
SORTING LISTS ...DONE!

t=50: Vado nelle coordinate (125 81)

t=100: Vado nelle coordinate (96 31)
  Nelle vicinanze rilevo la stazione 38 (95,35)

t=150: Vado nelle coordinate (14 35)
  Nelle vicinanze rilevo la stazione 8 (20,35)
  Nelle vicinanze rilevo la stazione 2 (5,35)

t=200: Vado nelle coordinate (120 35)

t=250: Vado nelle coordinate (115 41)
  Nelle vicinanze rilevo la stazione 44
(110,35)
t=300: STOP
```

**NAM**



# CODICE TCL COMPLETO

```
# ======================================================================
# Define options
# ======================================================================
set val(chan)           Channel/WirelessChannel    ;# channel type
set val(prop)         Propagation/TwoRayGround    ;# Propagation/TwoRayGround  |
Propagation/RFMGroundProp |  Propagation/SimpleProp

set val(netif)          Phy/WirelessPhy            ;# network interface type
set val(mac)            Mac/802_11                 ;# MAC type  Mac/802_11
set val(ifq)            Queue/DropTail    ;# interface queue type
set val(ll)             LL                         ;# link layer type
set val(ant)            Antenna/OmniAntenna        ;# antenna model
set val(ifqlen)         2                         ;# max packet in ifq
set val(rp)             DSDV                   ;# routing protocol default DSDV|DSR|AODV

set val(nnx)             8          ;# number of mobilenodes (asse x)
set val(nny)             6          ;# number of mobilenodes (asse x)

set val(nn)            [expr ( $val(nnx) * $val(nny) ) ]          ;# number of mobilenodes (asse x)


set val(gridx)            15           ;# distanza fra nodi griglia in orizzonatale
set val(gridy)            15

set val(x)           [ expr ( $val(nnx) * $val(gridx) ) + 10 ]     ;# X dimention of topo
set val(y)           [ expr ( $val(nny) * $val(gridy) ) + 10 ]     ;# X dimention of topo

set val(pos)        5  ;#n. di posizionamenti random e invio richiesta posizione
set val(intervallo)     50 ;#secondi di intervallo fra gli spostamenti
set val(stop)           [expr ( $val(pos)+1) * $val(intervallo) ]           ;# simulation time

puts "************************************************************"
puts "   Layout di $val(x) x $val(y) - $val(nnx) x $val(nny) nodi - griglia di $val(gridx) x
$val(gridy)   "
puts "************************************************************"

Phy/WirelessPhy set Pt_ 0.0000536
#0.0000535     non prende a 10 m
#0.00005355         prende a 10 m


Antenna/OmniAntenna set X_ 0
Antenna/OmniAntenna set Y_ 0
Antenna/OmniAntenna set Z_ 10

#Queue/DropTail/PriQueue set Prefer_Routing_Protocols    1


# create simulator instance

set ns_             [new Simulator]

# create trace object for ns and nam

set tracefd  [open out.tr w]
$ns_ trace-all $tracefd

set namtrace    [open out.nam w]
$ns_ namtrace-all-wireless $namtrace $val(x) $val(y)

# setup topography object
set topo     [new Topography]

# define topology
$topo load_flatgrid $val(x) $val(y)

#god per nodi e mobile
create-god [expr ($val(nn)+1)]

set chan_1_ [new $val(chan)]

#router trace deve essere ON oppure non monitorizza risposte ping
$ns_ node-config -adhocRouting $val(rp) \
-llType $val(ll) \
-macType $val(mac) \
```

```
-ifqType $val(ifq) \
-ifqLen $val(ifqlen) \
-antType $val(ant) \
-propType $val(prop) \
-phyType $val(netif) \
-topoInstance $topo \
-agentTrace ON \
-routerTrace ON \
-macTrace OFF \
-movementTrace ON \
-channel $chan_1_ \

#Define a 'recv' function for the class 'Agent/Ping'
Agent/Ping instproc recv {from rtt cx cy} {
       $self instvar node_
       #puts "Da $from a [$node_ id] <---> $rtt ms (Coord  $cx , $cy )"
       puts "  Nelle vicinanze rilevo la stazione $from ($cx,$cy). RTT $rtt ms"
}

# settaggio griglia nodi
set i 0 ;#contatore nodo da 0 a (nnx)x(nny)
for {set x 0} {$x < $val(nnx) } {incr x} {
  for {set y 0} {$y < $val(nny) } {incr y} {

     # setto nodo e dimensione
     set node_($i) [$ns_ node]

     # setto coordinate a griglia
     $node_($i) random-motion 0        ;# disable random motion
     $node_($i) set X_ [ expr (5 + ($x * $val(gridx))) ]
     $node_($i) set Y_ [ expr (5 + ($y * $val(gridy))) ]
     $node_($i) set Z_ 0.0
     $ns_ initial_node_pos $node_($i) 10 ;# dopo settaggio coordinate, imposto dimensioni nodi

     set i [expr $i + 1] ;# incrementa i (quindi va da 0 a [nnx * nny])
  }
}

# setto nodo mobile e suo agente
set nodo_mobile [$ns_ node]
#
$nodo_mobile random-motion 0           ;# disable random moti
$nodo_mobile set X_ 0.0
$nodo_mobile set X_ 0.0
$nodo_mobile set Z_ 0.0
#
$ns_ initial_node_pos $nodo_mobile 50


# attacco gli agenti
# Il nodo mobile ha attaccato un agente per ogni nodo presente per poter comunicare
set i 0
for {set x 0} {$x < $val(nnx) } {incr x} {
  for {set y 0} {$y < $val(nny) } {incr y} {

      # agenti sui nodi fissi
     set p($i) [new Agent/Ping]
     $ns_ attach-agent $node_($i) $p($i)
     #
     # creo agente corrispondente sul nodo mobile p0n-p0
     set pm($i) [new Agent/Ping]
     $ns_ attach-agent $nodo_mobile $pm($i)
     #
     #attacco i corrispondenti
     $ns_ connect $p($i) $pm($i)
     #
     set i [expr $i + 1] ;# incrementa i (quindi va da 0 a [nnx * nny])
  }
}


#imposto parametri per coordinate random x e y di posizionamento del nodo mobile
global defaultRNG
$defaultRNG seed 0
#
set rngx [new RNG]
set RVx [new RandomVariable/Uniform]
$RVx set min_  1
$RVx set max_  $val(x)-1
$RVx use-rng $rngx
#
set rngy [new RNG]
set RVy [new RandomVariable/Uniform]
```

```
$RVy set min_  1
$RVy set max_  $val(y)-1
$RVy use-rng $rngx


# ciclo di posizionamento random nodo del mobile e sua richiesta di posizione
for {set t 1} {$t <= $val(pos) } {incr t} {

   #set px [format "%d" [expr round([$RVx value])] ]
   #set py [format "%d" [expr round([$RVy value])] ]

   set px [expr round([$RVx value])]
   set py [expr round([$RVy value])]


  # puts "Vado nelle coordinate ($px $py)"

   # mi sposto e invio (cioè tutti gli agenti attaccati sopra il mobile inviano a tutti gli altri
corrispettivi)
   set tempo  [ expr ( $val(intervallo) * $t ) ]
   set tempo1 [ expr ( $tempo + 1 ) ]
   set tempo2 [ expr ( $tempo + 2 ) ]

   $ns_ at $tempo    "puts \"\nt=$tempo: Vado nelle coordinate ($px $py)\""
   $ns_ at $tempo.1  "$nodo_mobile setdest $px $py 9999.0"

   #tutti gli agenti del mobile fanno richiesta posizione
   for {set i 0} {$i < $val(nn) } {incr i} {
      $ns_ at $tempo2 "$pm($i) send"
   }
}


# reset alla fine
for {set i 0} {$i < $val(nn) } {incr i} {
    $ns_ at $val(stop) "$node_($i) reset";
}
$ns_ at $val(stop) "$nodo_mobile reset";

proc stop {} {
    global ns_ tracefd
    $ns_ flush-trace
    close $tracefd
    exec nam out.nam
    #exec gvim out.tr &
    puts "NS EXITING..."
    $ns_ halt

}

$ns_ at $val(stop) "puts \"t=$val(stop): STOP\""
$ns_ at $val(stop) "stop"

puts "Starting Simulation..."
$ns_ run
```

---

# PING.CC

```
#include "ping.h"
#include "mobilenode.h"
#include "address.h"
#include "node.h"

int hdr_ping::offset_;
static class PingHeaderClass : public PacketHeaderClass {
public:
       PingHeaderClass() : PacketHeaderClass("PacketHeader/Ping",
                                             sizeof(hdr_ping)) {
              bind_offset(&hdr_ping::offset_);
       }
} class_pinghdr;


static class PingClass : public TclClass {
public:
       PingClass() : TclClass("Agent/Ping") {}
       TclObject* create(int, const char*const*) {
              return (new PingAgent());
       }
```

```
} class_ping;


PingAgent::PingAgent() : Agent(PT_PING), seq(0), oneway(0)
{
      bind("packetSize_", &size_);
}

int PingAgent::command(int argc, const char*const* argv)
{
  if (argc == 2) {
    if (strcmp(argv[1], "send") == 0) {
      // Create a new packet
      Packet* pkt = allocpkt();
      // Access the Ping header for the new packet:
      hdr_ping* hdr = hdr_ping::access(pkt);
      // Set the 'ret' field to 0, so the receiving node
      // knows that it has to generate an echo packet
      hdr->ret = 0;
      hdr->cx = 100;
      hdr->cy = 150;
      hdr->seq = seq++;
      // Store the current time in the 'send_time' field
      hdr->send_time = Scheduler::instance().clock();
      // Send the packet
      send(pkt, 0);
      // return TCL_OK, so the calling function knows that
      // the command has been processed
      return (TCL_OK);

    }

    else if (strcmp(argv[1], "start-WL-brdcast") == 0) {
      Packet* pkt = allocpkt();

      hdr_ip* iph = HDR_IP(pkt);
      hdr_ping* ph = hdr_ping::access(pkt);

      iph->daddr() = IP_BROADCAST;
      iph->dport() = iph->sport();
      ph->ret = 0;
      send(pkt, (Handler*) 0);
      return (TCL_OK);
    }

    else if (strcmp(argv[1], "oneway") == 0) {
      oneway=1;
      return (TCL_OK);
    }
  }

  // If the command hasn't been processed by PingAgent()::command,
  // call the command() function for the base class
  return (Agent::command(argc, argv));
}


void PingAgent::recv(Packet* pkt, Handler*)
{
  // Access the IP header for the received packet:
  hdr_ip* hdrip = hdr_ip::access(pkt);

  // Access the Ping header for the received packet:
  hdr_ping* hdr = hdr_ping::access(pkt);


  // check if in brdcast mode
  if ((u_int32_t)hdrip->daddr() == IP_BROADCAST) {
    if (hdr->ret == 0) {

      printf("Recv BRDCAST Ping REQ : at %d.%d from %d.%d\n", here_.addr_, here_.port_, hdrip-
>saddr(), hdrip->sport());
      Packet::free(pkt);

      // create reply
      Packet* pktret = allocpkt();

      hdr_ping* hdrret = hdr_ping::access(pktret);
      hdr_cmn* ch = HDR_CMN(pktret);
      hdr_ip* ipret = hdr_ip::access(pktret);

      hdrret->ret = 1;
```

```
      // add brdcast address
      ipret->daddr() = IP_BROADCAST;
      ipret->dport() = ipret->sport();

      send(pktret, 0);

    } else {
      printf("Recv BRDCAST Ping REPLY : at %d.%d from %d.%d\n", here_.addr_, here_.port_, hdrip-
>saddr(), hdrip->sport());
      Packet::free(pkt);
    }
    return;
  }
  // Is the 'ret' field = 0 (i.e. the receiving node is being pinged)?
  if (hdr->ret == 0) {
    // Send an 'echo'. First save the old packet's send_time
    double stime = hdr->send_time;
    int rcv_seq = hdr->seq;


    // Discard the packet
    Packet::free(pkt);
    // Create a new packet
    Packet* pktret = allocpkt();
    // Access the Ping header for the new packet:
    hdr_ping* hdrret = hdr_ping::access(pktret);
    // Set the 'ret' field to 1, so the receiver won't send
    // another echo
    hdrret->ret = 1;

/* MOD - calcolo la posizione dei nodi */
    int nodocorr = hdrip->src_.addr_ >> Address::instance().NodeShift_[1];
      nsaddr_t t;
      //t=this->address();
      MobileNode *test = (MobileNode *) (Node::get_node_by_address(nodocorr));
      double x,y,z;
      test->getLoc(&x,&y,&z);

    hdrret->cx = x;
    hdrret->cy = y;
/* FINE MOD    */

    // Set the send_time field to the correct value
    hdrret->send_time = stime;
    // Added by Andrei Gurtov for one-way delay measurement.
    hdrret->rcv_time = Scheduler::instance().clock();
    hdrret->seq = rcv_seq;
    // Send the packet
    send(pktret, 0);
  } else {
    // A packet was received. Use tcl.eval to call the Tcl
    // interpreter with the ping results.
    // Note: In the Tcl code, a procedure
    // 'Agent/Ping recv {from rtt}' has to be defined which
    // allows the user to react to the ping result.
    char out[100];
    // Prepare the output to the Tcl interpreter. Calculate the
    // round trip time
    if (oneway) //AG
      sprintf(out, "%s recv %d %d %3.1f %3.1f", name(),
          hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
          hdr->seq, (hdr->rcv_time - hdr->send_time) * 1000,
          (Scheduler::instance().clock()-hdr->rcv_time) * 1000);

    /* MOD - qui il ping ritorna e cx,cy li leggo solamente */
    else sprintf(out, "%s recv %d %3.1f %.0lf %.0lf", name(),
          hdrip->src_.addr_ >> Address::instance().NodeShift_[1],
          (Scheduler::instance().clock()-hdr->send_time) * 1000, hdr->cx, hdr->cy);
    Tcl& tcl = Tcl::instance();
    tcl.eval(out);
    // Discard the packet
    Packet::free(pkt);
  }
}
```

**PING.H**

```
#ifndef ns_ping_h
#define ns_ping_h

#include "agent.h"
#include "tclcl.h"
#include "packet.h"
#include "address.h"
#include "ip.h"

struct hdr_ping {
	char ret;
	double send_time;
	double rcv_time;	// when ping arrived to receiver
	int seq;		// sequence number
	double cx, cy;


	// Header access methods
	static int offset_; // required by PacketHeaderManager
	inline static int& offset() { return offset_; }
	inline static hdr_ping* access(const Packet* p) {
		return (hdr_ping*) p->access(offset_);
	}
};

class PingAgent : public Agent {
public:
	PingAgent();
	int seq;	// a send sequence number like in real ping
	int oneway;	// enable seq number and one-way delay printouts
	virtual int command(int argc, const char*const* argv);
	virtual void recv(Packet*, Handler*);
};

#endif // ns_ping_h
```