MobiWan: A NS-2.1b6 simulation platform for Mobile IPv6 in Wide Area Networks

Thierry Ernst[‡†]
[‡]MOTOROLA LABS Paris
Espace Technologique St-Aubin
91193 Gif-sur-Yvette Cedex - France

[†]INRIA Rhône-Alpes
PLANETE Project
655, Avenue de l'Europe
38334 Saint-Ismier Cedex - France

E-mail:Thierry.Ernst@inrialpes.fr
Web:http://www.inrialpes.fr/planete/pub/mobiwan

June 1, 2001

Contents

Copyright notice

Chapter 1

Presentation

1.1 Introduction

Our research project is concerned about studying mobility of nodes in Wide-Area IPv6 Networks. Our main aim is to simulate local mobility (with a single administrative domain or site) and global mobility (across domain boundaries or sites) in potentially large Internet topologies. Particularly, we would like to simulate Mobile IPv6, Hierarchical Mobile IPv6, various micro-mobility protocols, etc. Our simulations may sometimes involve the manipulation of large Internet topologies (hundreds of nodes). Indeed, we need to simulate a representative large Internet topology and a mobility model which would allow mobility of a node in any part of the topology. This is shown on figure **??** where a mobile is moving between two distinct sites separated by a large internetwork - the cloud.

NS-2 modules simulating Mobile IPv4, Wireless communication, and Ad-hoc networks have already been contributed by SUN and CMU. They were first developed for NS-2.1b2 and were then incorporated in the NS-2.1b6 distribution.

However, those modules do not fulfill our needs and particularly not the movement of nodes in a WAN. First, the actual modules simulate movement of nodes within a bounded geographical grid (i.e. geographical movement) whereas WAN mobility is more concerned by topologically distant movements in a topology (i.e. topological movement). Second, Mobile IPv6 and Hierarchical Mobile IPv6 are not supported. And last, there is no easy means to configure and manipulate large topologies. This is why the NS extensions presented in this report were needed.

We are happy to contribute this code to the NS community because we think that it may be very useful to many of you. However, this code was developed for our own needs and may not always be adequate for everyone, We apologize for that and we also apologize for the bugs you may find in this code (surely quite a few !), and would welcome any remark, feedback and patch (especially if you find a bug !), in order to enhance it whenever possible.

Our code therefore comes with no guarantees and we are sorry that we may not be of any help to those of you who wish to use it, mainly because we are likely running out of time. Though, we will do our best to help you whenever possible and we will also think about porting our code to the ns-2.1b8 distribution suite. Any help would be appreciated for this task.

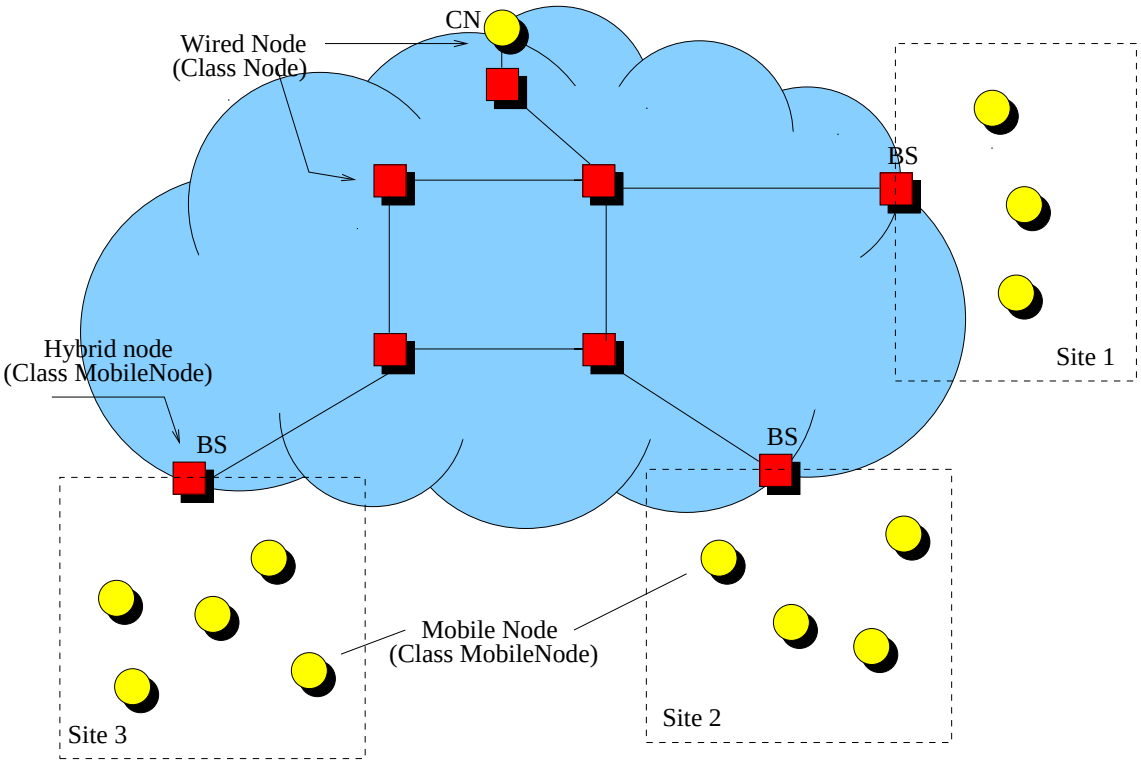

Figure 1.1: Local and global mobility in NS

1.2 Aim of our enhancements

1.2.1 Mobility in a WAN

In order to demonstrate Wide-Area Mobility, we need to differentiate local mobility (mobility within a site or domain of administration) from global mobility (mobility between sites or domains of administration).

A site is a network under a common administrative policy. All nodes in a site are identified by the same IP prefix. The length of the prefix determines the maximum size of the site. The site may also be defined by a bounded geographical area. Mobile nodes move freely within the geographical boundary of the site and attach to distinct routers within the site. A good instance of a site may be a university campus.

For WAN mobility, we need:

- Means to define the notion of sites in NS network topologies.
- Means for a mobile node to remain in a site bounded to a certain geographical area, and then move to another site.
- A mobility model which distinguishes local mobility (topologocally close movements) from global mobility (topologically distant movements).

Fig. 1.1 shows the distinction between local mobility (within a grid) and global mobility (from grid square to another).

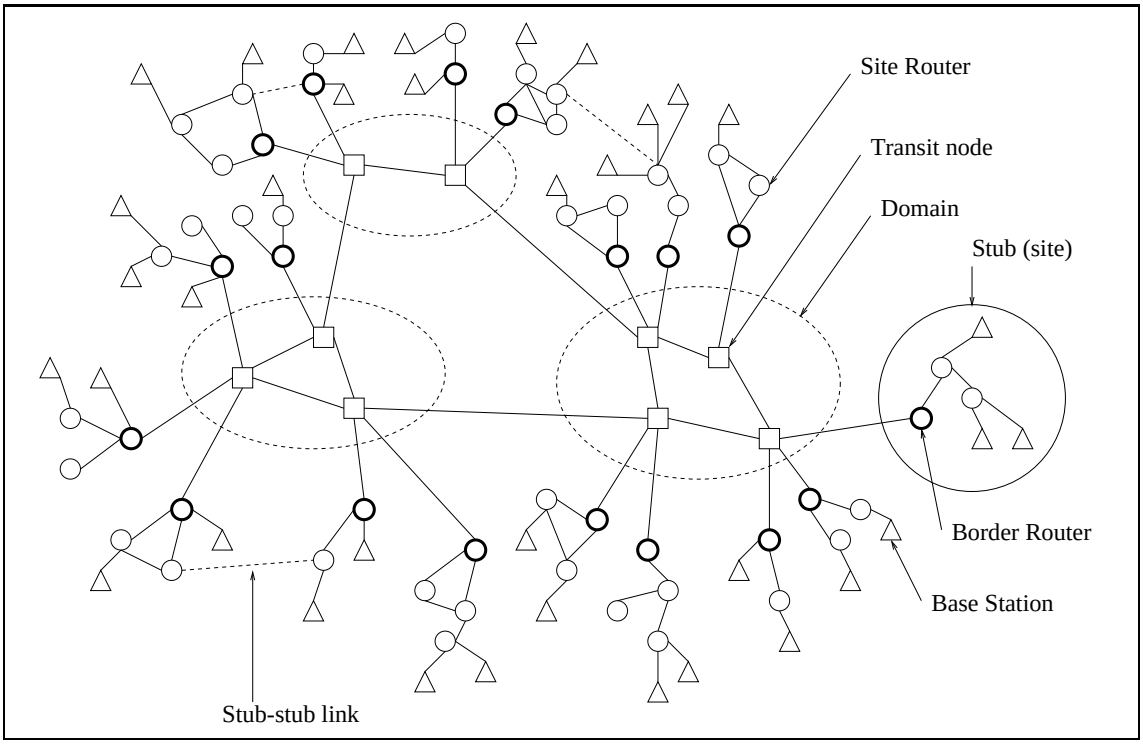

Figure 1.2: Expected topology

1.2.2 Topology Generation and Manipulation

Our simulation project involves large Internet topologies. In order to configure and manipulate very large network topologies, it may be very usefulin an easy manner, i.e. automatically. Particularly, we want nodes with distinct position in the topology hierarchy to perform distinct functions.

As we may notice, topologies generated by GT-ITM are hierarchical and already provide all the information we need, but most of the SGB information is lost by the existing NS translators. The TCL output doesn't distinguish Stub Nodes from Transit Nodes anymore, which would be very useful.

In order to conduct large simulations, we need:

- Automatic generation of large and hierarchical topologies wich are an actual representation of the Internet.
- Hierarchical topologies that exhibit the notion of domains, sites, and subnetworks.
- Easy configuration and manipulation of the topologies in order to identify the position thereof the function of the node in the hierarchy, and to configure the node with the necessary features according to its function. For instance, we may want to configure all Border Routers with HMIPv6 and all Base Stations with a Home Agent function.
- Easy configuration of the simulation scenario, for instance select all the Site Routers in a particular site as the mobile node's correspondent nodes.

Fig 1.2 shows the kind of topology we would like to manipulate directly in NS.

1.2.3 IPv6 and Mobile IPv6

Since our aim is to study mobility in Wide-Area IPv6 networks, we need modules that simulate the IPv6, Mobile IPv6 and Hierarchical Mobile IPv6 protocols.

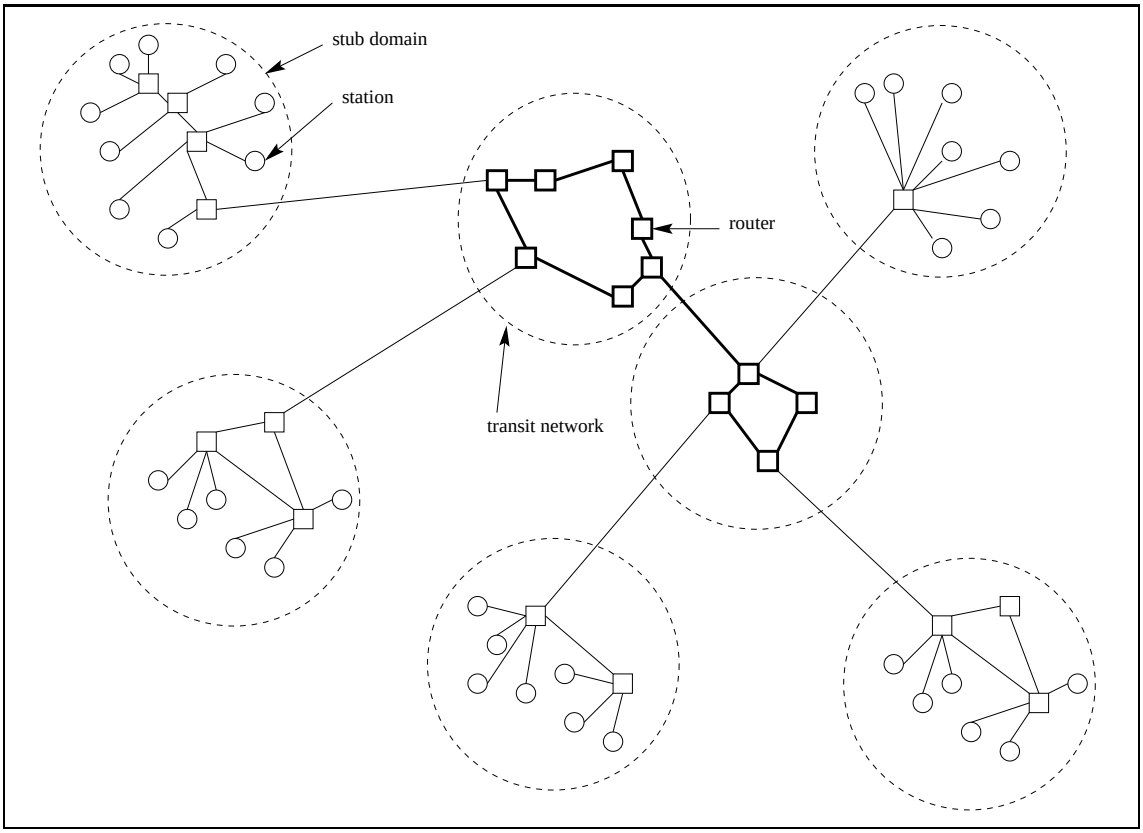

Figure 1.3: Transit stubs topology by gt-itm

1.3 Existing Facilities in NS

1.3.1 Modeling Internet Topologies

NS-2 does actually provide means to produce topologies which actually are a good representation of the Internet. For this purpose, we may make use of GT-ITM [2], a topology generator, and translates its output in NS-2 procedure calls.

The following is an extract from *Modeling Internet Topology* [1]. This is the model used by *Tiers*, a topology generator.

Each routing domain in the Internet can be classified as either a stub domain or a transit domain. A stub domain carries only traffic that originates or terminates in the domain. Transit domains do not have this restriction. The purpose of transit domains is to interconnect stub domains efficiently; without them, every pair of stub domains would need to be directly connected to each other. Stub domains generally correspond to campus networks or other collection of interconnected LANs, while transit domains are almost always wide- or metropolitan-area networks (WANs or MANs).

A transit domain consists of a set of backbone nodes. In a transit domain each backbone node may also connect to a number of stub domains, via gateway nodes in the stubs. Some backbone nodes also connect to other transit domains. Stub domains can be further classified as single- or multihomed. Multi-homed stub domains have connections to more than one transit domain, single-homed stubs connect to only one transit domain. Some stubs domains may have links to other stubs. Transit domains may themselves be organized in hierarchies, e.g. MAN s connect mainly to stubs domains and WANs.

Gt-itm is a topology generator that uses different kinds of generation models. One of this models is the transit stub model [5], as it is shown on fig. 1.3. This model produces hierarchised topologies in SGB format.

However, the translator from SGB format to NS-2 procedure calls looses most of the information, particularly the hierarchy of nodes. Although the translator produces NS-2 hierarchical addresses, the user in unable to determine if a node is a transit node or a stub node. This information may be very useful.

Such a topology is a good skeleton for the topology we need for the simulation. The site concept doesn't appear so it has to be added, this will be the main work of the transformation of this topology. A site will be a stub domain of the previous topology to which we will add base stations. A site is a stub domain and a set of base stations linked to nodes of this stub but it also needs properties that make a node enter or leave this site, remember that the principe of the simulation is mobile nodes entering and leaving sites.

1.3.2 Different kinds of nodes in NS

Two kinds of nodes are of primary concern to us. In figure 1.1, wired nodes *class Node* are in the cloud, while nodes in grids are instances of *Class MobileNode*. Nodes at the edge of the cloud are Base Stations *Class MobileNode* with no motion and are used to connect the mobile nodes to the wired nodes.

Class *Node* Class *Node* is designed for simulating wired nodes. Nodes of this class connect to one another by the means of point-to-point *Links*, which have specified delays, bandwidths and loss probability. As this was first proposed by SUN enhancements to simulate]it Mobile IP (v4), this class *Nodes* may be used for simulating mobiles nodes. This could be done by specifying that links become up and $down$ at a specified time. However, this is not a common behavior and this is not efficient in terms of simulation processing nor size of simulation since this implies to set up links between a mobile node and all its possible points of attachment. This is only acceptable for simulations of a very limited size (limited number of mobiles and points of attachment). We can therefore not rely on this for our simulations.

Class *MobileNode* Mobile nodes and wireless facilities have been added in NS in order to simulate ad-hoc networks. The aim is to evaluate ad-hoc routing protocols. To the contrary of wired nodes of class *Nodes*, mobile nodes of class *MobileNode* connect to one-another by means of a *Channels* (a kind of wireless broadcast medium). Nodes of class *MobileNode* are moving in a 2-dimensional grid of a specified size according to some parameters. Co-ordinates are adjusted as the node moves within the grid. Mobile nodes may communicate with a wired node through a Base Stations. Base Stations are an hybrid of *Class Node* and *Class MobileNode* also instances of *Class MobileNode*, but with a null motion.

Fig. 1.4 shows the structure of the Mobile IPv4 node.

1.3.3 Existing Mobility Model

The NS distribution includes a mobility model used to create mobility scenarios. It is used for nodes of class *MobileNode*. Movements of nodes *MobileNode* in the grid may be specified by two means:

- Setting explicitly starting position of the node and its future destination. The node has a speed, and moves towards specified co-ordinates. The co-ordinates of a new destination and a new speed may be given at any time.
- Random movement of the node. The node starts its movement with a random speed and direction, specified as co-ordinates within the grid. A new speed and direction is chosen at a random time.

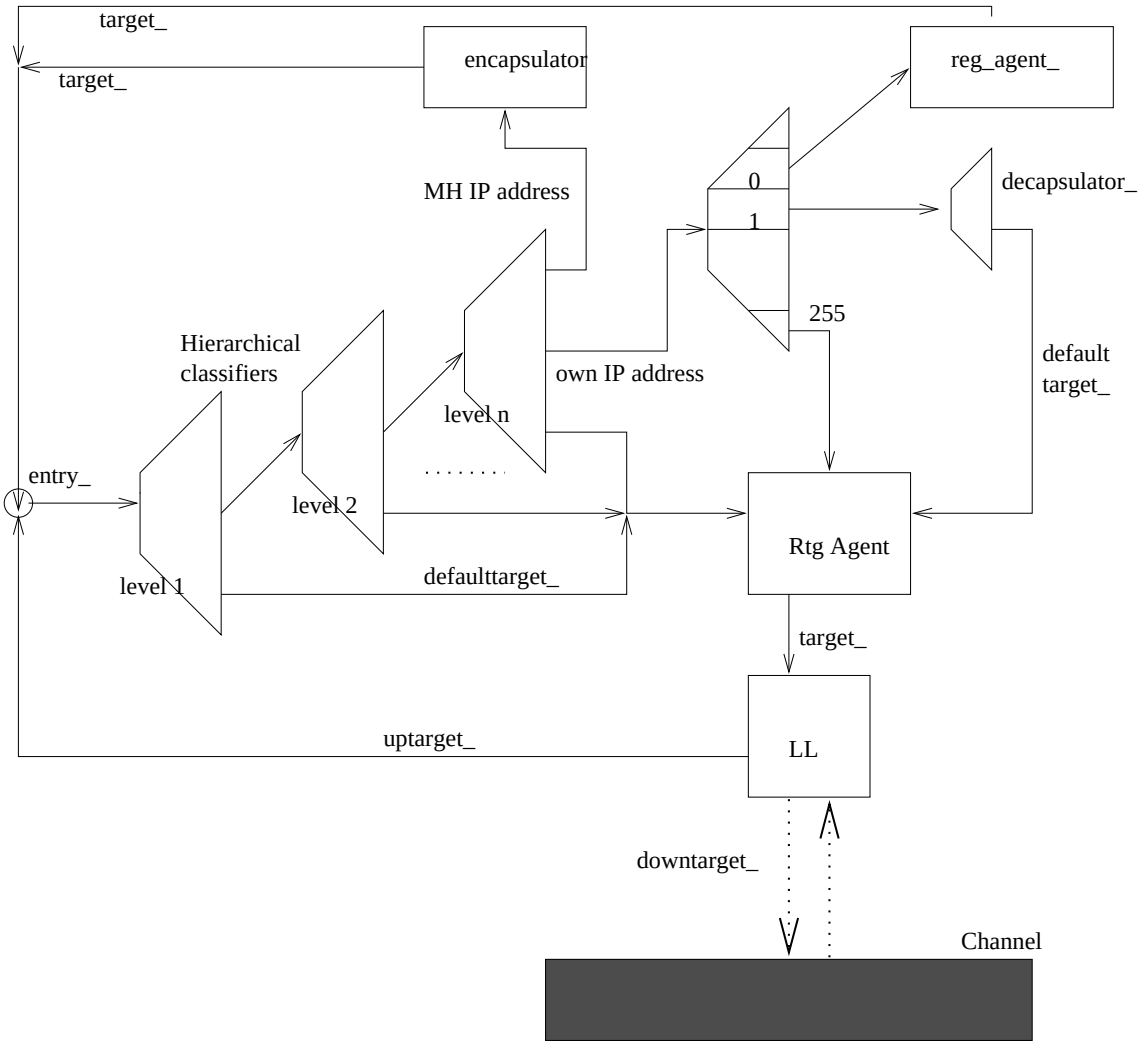

Figure 1.4: Mobile IPv4 in NS

Chapter 2

NS-2.1b6 Enhancements

2.1 Summary of New Features

Our simulation platform is built on NS-2 (ns-2.1.b6). It allows to simulate Wide-Area mobility of nodes in a large and hierarchical internetwork (simulations of up to 3000 nodes have been conducted so far). The platform is mainly used to simulate MobileIPv6 and Hierarchical Mobile IPv6 /footnoteThe HMIPv6 code is not yet in the current code release but it is not limited to simulate those protocols only. Particularly, the TOPOGEN and TOPOMAN extensions may be used for any simulation involving large networks, with or without mobile nodes.

We have enhanced NS-2 with the following items:

TOPOGEN: GT-ITM to NS translator A new tool to translate a SGB Topology to a NS topology. In order to allow easy configuration and manipulation, nodes are classified according to their position (which transit domain in the topology, which site in the transit domain, and which subnet in the site) and their function.

TOPOMAN Library A library of Tcl/OTcl procedures to manipulate and configure the generated topology. For example, the library can return the set of nodes in a site, the set of border nodes, the set of Base Stations, etc.

SCEN TOOLS Configuration and mobility scenario generation tool which makes use of TOPOMAN.

Memory consumption optimization We have brought NS addressing from 3 levels to 4 levels. This hack was necessary to minimize routing table computation for large topologies. Moreover, allowing a 4th layer in the hierarchy makes configuration of the network easier. The NS team should soon bring ns-2 to a n-level hierarching addressing scheme, but we couldn't wait for it. Fig. 2.1 shows the new structure of NS nodes with 4 levels of hierarchy.

Wide-area Mobility Intra-Site mobility is mobility within a single domain of administration or mobility within a site (a campus, an organization). Intra-site mobility is performed by means of the existing NS-2 features contributed by CMU. Inter-Site mobility means crossing domain boundaries, i.e. mobility between two domains of administration or between sites. Inter-Site mobility is performed by means of new features that allow the mobile node to move from one site to another. Each site is associated with a distinct air channel. All Base Stations in the same site listen on the same air channel and are associated with the same geographical grid. The Mobile Node changes its geographical location within the grid . A mobile crossing a site boundary is therefore changing the current channel it is listening to. To keep things simple, the Mobile Node does not change its geographical coordinates when it changes from one site to another. This allows us to keep using the existing CMU mobility model, particularly random movements. Indeed, the channel on which the Mobile Node is listening to is transparent to the mobility model.

IPv6 and Mobile IPv6 protocols We have developped a set of NS Agents that simulate the Mobile IPv6 and IPv6 protocols. This is going to be extended to support Hierarchical Mobile IPv6. We haven't implemented all the proper IPv6 features as this was not necessary for our simulations (e.g. DHCPv6, Neighbor Discovery, etc). Our only concern was to modify the header size, to add Router Advertisements and Solications between BSs and MNs, to add encapsulation and decapsulation at all nodes, and to add the routing extension header processing.

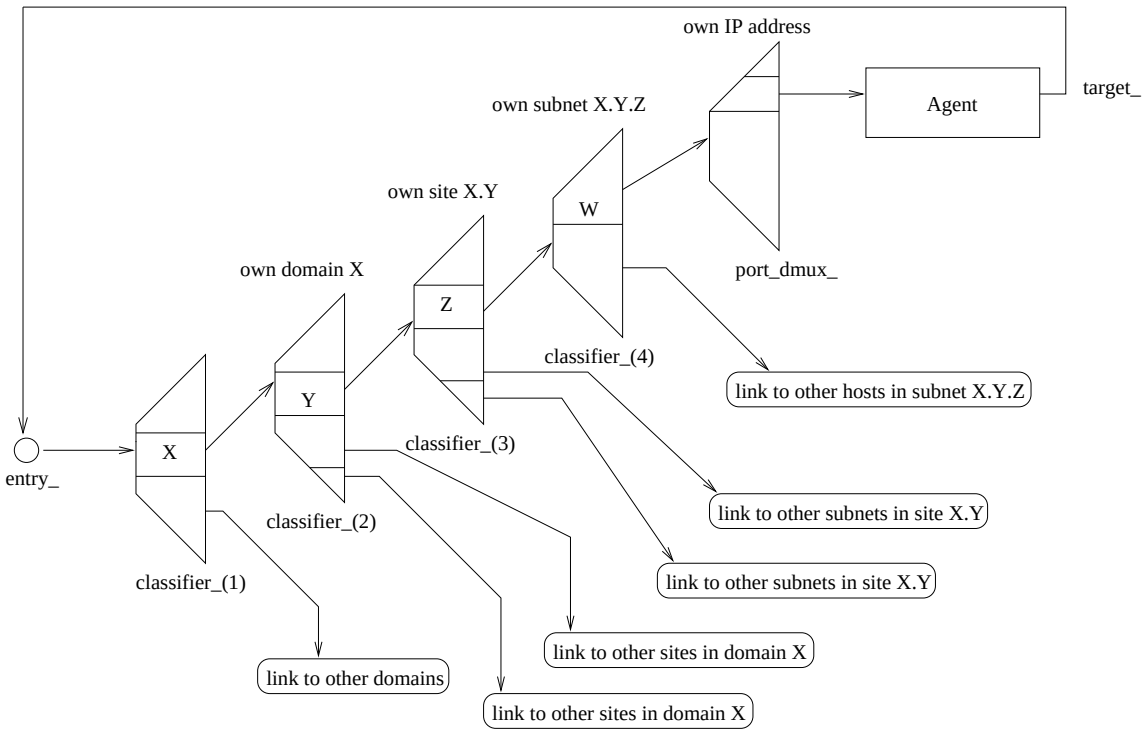

Figure 2.1: 4 levels of hierarchy for NS addresses

Mobility of a multicast source We have hacked some minor parts of the code in order to support Multicast for Mobile Nodes and Base Stations (the existing NS doesn't allow those to send or receive multicast and it was quite easy to add this). We have only added features allowing a Mobile Node to send to a group, not to receive from a group (indeed it may also work, but not tested !)

2.2 Topology generation and manipulation

MORE DETAILED DOCUMENTATION LATER IN SUMMER

2.2.1 TOPOGEN

TOPOGEN translates the GT-ITM output (SGB format) to a NS-like format suitable for TOPOMAN. GT-ITM is left unchanged.

TOPOGEN classifiies nodes according to their position in the topology (which transit domain in the topology, which site in the transit domain, and which subnet in the site) and their function. As an output, TOPOGEN first tells the number of administrative domains in the topology. Then, it adds transit nodes to a specified domain and for each transit node, it determines the number of sites. For each site, it adds a border node, and the other nodes. Of course, it also outputs the list of links between the nodes.

Here follows a TOPOGEN output. This is a 53-nodes topology divided in 2 administrative domains comprising a total of 8 sites. There are 4 Base Stations in each site.

The GT-ITM parameters were:

```
# <method keyword> <number of graphs> [<initial seed>]
# A: <# stubs/trans node> <#rand. t-s edges> <#rand. s-s edges>
# B: <# transit domain>
# C: <# nodes/transit domain>
# D: <# nodes/stub domain>
# <n> <scale> <edgemethod> <alpha> [<beta>] [<gamma>]
#
# number of nodes = BxCx(1+AxD)
# nb sites (stubs) = AxBxC
# number of nodes = 2x2x(1+2x2) = 20
ts 1 47
2 0 0
2 20 3 0.5 1.0
2 20 3 0.6 1.0
2 10 3 0.42 1.0
```

This topology is then created with `topogen.sh micro-test-20`. `topogen.sh` is a script which calls GT-ITM and then TOPOGEN:

```
#!/bin/sh


GT_ITM_CMD=itm
SGB2NS=sgb2ns_sites

GT_ITM_FILE=$1

NB_OUT_FILES=‘grep ts $GT_ITM_FILE | cut -d’ ’ -f2‘

echo "$GT_ITM_CMD $GT_ITM_FILE" ;
$GT_ITM_CMD $GT_ITM_FILE
```

```
COUNTER=0
while [ $COUNTER -lt $NB_OUT_FILES ] ; do
   echo "$SGB2NS $GT_ITM_FILE-$COUNTER.gb" ;
   $SGB2NS $GT_ITM_FILE-$COUNTER.gb ;
   COUNTER=‘expr $COUNTER + 1‘ ;
done
```

And now, the TOPOGEN output:

```
set tm_all_nodes(z) 0 ; #  global variable
proc tm_build_links {} {
  global tm_all_nodes ns

   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(11) 100Mb 1.80ms DropTail
   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(10) 100Mb 2.00ms DropTail
   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(5) 50Mb 1.10ms DropTail
   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(4) 50Mb 2.00ms DropTail
   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(2) 50Mb 3.40ms DropTail
   $ns duplex-link $tm_all_nodes(0) $tm_all_nodes(1) 100Mb 0.90ms DropTail
   $ns duplex-link $tm_all_nodes(1) $tm_all_nodes(9) 50Mb 3.10ms DropTail
   $ns duplex-link $tm_all_nodes(2) $tm_all_nodes(3) 10Mb 2.90ms DropTail
   $ns duplex-link $tm_all_nodes(5) $tm_all_nodes(6) 10Mb 1.20ms DropTail
   $ns duplex-link $tm_all_nodes(5) $tm_all_nodes(7) 10Mb 1.30ms DropTail
   $ns duplex-link $tm_all_nodes(8) $tm_all_nodes(9) 10Mb 1.60ms DropTail
   $ns duplex-link $tm_all_nodes(10) $tm_all_nodes(16) 50Mb 0.60ms DropTail
   $ns duplex-link $tm_all_nodes(10) $tm_all_nodes(14) 50Mb 0.70ms DropTail
   $ns duplex-link $tm_all_nodes(10) $tm_all_nodes(12) 50Mb 1.90ms DropTail
   $ns duplex-link $tm_all_nodes(10) $tm_all_nodes(11) 100Mb 3.20ms DropTail
   $ns duplex-link $tm_all_nodes(11) $tm_all_nodes(18) 50Mb 2.90ms DropTail
   $ns duplex-link $tm_all_nodes(13) $tm_all_nodes(14) 10Mb 3.60ms DropTail
   $ns duplex-link $tm_all_nodes(15) $tm_all_nodes(17) 10Mb 0.90ms DropTail
   $ns duplex-link $tm_all_nodes(16) $tm_all_nodes(17) 10Mb 2.50ms DropTail
   $ns duplex-link $tm_all_nodes(18) $tm_all_nodes(19) 10Mb 2.10ms DropTail
}

proc tm_load_nodes {} {
  global TOPOM

  $TOPOM tm_add_nb_domains 2

  $TOPOM tm_add_tn_to_domain 0 0
  $TOPOM tm_add_nb_sites_to_tn_id 0 3
  $TOPOM tm_add_tn_to_domain 1 0
  $TOPOM tm_add_nb_sites_to_tn_id 1 1
  $TOPOM tm_add_border_to_site 2 1
  $TOPOM tm_add_node_to_site 3 1
  $TOPOM tm_add_border_to_site 4 2
  $TOPOM tm_add_border_to_site 5 3
  $TOPOM tm_add_node_to_site 6 3
  $TOPOM tm_add_node_to_site 7 3
  $TOPOM tm_add_node_to_site 8 4
  $TOPOM tm_add_border_to_site 9 4
  $TOPOM tm_add_tn_to_domain 10 1
  $TOPOM tm_add_nb_sites_to_tn_id 10 3
```

```
  $TOPOM tm_add_tn_to_domain 11 1
  $TOPOM tm_add_nb_sites_to_tn_id 11 1
  $TOPOM tm_add_border_to_site 12 5
  $TOPOM tm_add_node_to_site 13 6
  $TOPOM tm_add_border_to_site 14 6
  $TOPOM tm_add_node_to_site 15 7
  $TOPOM tm_add_border_to_site 16 7
  $TOPOM tm_add_node_to_site 17 7
  $TOPOM tm_add_border_to_site 18 8
  $TOPOM tm_add_node_to_site 19 8
}
```

2.2.2 TOPOMAN

TOPOMAN automatically builds the NS topology. It loads the TOPOGEN topology, defines the NS addressing automatically (what a headache this was for large topologies !), and create nodes and links.

TOPOMAN does not only allow to create topologies and to configure them automatically, but it also allow to display and play with the topology before it is effectively created. This is very useful to configure large topologies, since the user doesn't need NS to create all the objects and build the routing table before displaying the topolog: you don't need to run NS (`$ns run` to display the topology.

Loading the topology

NS topologies may be loaded by hand with TOPOMAN instructions, or using the TOPOGEN translator directly. By hand, the user just needs to write the procedures `proc tm_build_links` and `proc tm_load_nodes` as in the example above.

TOPOMAN allows to classify nodes according to their function in the topology. By now, we have:

- Transit Nodes (Backbone nodes)
- Border Routers (nodes connecting a site to a transit domain)
- Site Routers (a router in a site other than the Border Router)
- Base Stations (router connecting Mobile Nodes to a wired node)
- Mobile Nodes
- ... add more if you need ...

The user may add new types of nodes at will, still using the TOPOMAN calls. TOPOMAN keeps detailed information about the topology. For instances, TOPOMAN knows the number of domains, sites, nodes in the site, etc. TOPOMAN may be extended to return any kind of information needed by the user. By now, TOPOMAN may return the list of Border Nodes in the topology, the address prefix for nodes in the topology, etc.

Addressing

Each transit node corresponds to a sub-domain. An administrative domain is therefore an aggregation of one or several transit nodes. Each border node is the gateway between a site and the sub-domain. All routers in

the same site (Border Routers, Site Routers, Base Stations) have the same prefix, i.e. this corresponds to the prefix of the site. Each router in the site has also its own subnetwork prefix. This means that end-systems on the same subnetwork share the same subnetwork prefix. A mobile node is such an end-system.

TOPOMAN arranges NS addresses according to this information. You don't need to care about NS addressing anymore. If you want to know which site a node belongs to, you may use the TOPOMAN procedures because TOPOMAN keeps all the information internally. (see file ns-topoman.tcl and the demo scripts - more detailed information about available procedures will be provided later)

The NS address is divided in 3 levels by default. It was technically impossible to show the domain and the sub-domain in the NS address, this information is kept internally to TOPOMAN.

If you want mobiles nodes in the topology, NS addresses will be organized like this:

- First level identifies the site. Transit node have their own site (technical reason). All nodes with the same site have the same site id.
- Second level identifies the subnet within the site. This gives the subnet prefix. All border routers, site routers, and base stations have the same a distinct subnet id.
- Third level identifies the host within the subnet, i.e. the end-system. Routers have $hostid = 0$; mobile nodes have $hostid <> 0$.

Effective node creation

For each kind of node, there is an associated procedure `create-<type_of_node>` which is used to effectively create the NS object(s). You may change this procedure if you want a Base Station to be configured slightly differently from the default one (see file proc-mipv6-config.tcl). For instance, if you want Border Routers to be configured with a specific protocol, you may do this in the procedure `proc create-border-router`.

How to use TOPOMAN

```
global TOPOM

 # Create the TOPOMAN instance
 set TOPOM [new Topoman]

 # Load the main topology
 source <topo_file.tcl>
 tm_load_nodes

 # Load additional nodes (Base Stations, Mobile Nodes ...)

 # Load Base Stations
 $TOPOM tm_add_bs_to_node <attach_router_id> <X> <Y> <Z> [<args>]
 $TOPOM tm_add_bs_to_node <attach_router_id> <X> <Y> <Z> [<args>]

 # Load Mobile Nodes
 $TOPOM tm_add_mn_to_bs <ha_node_id> <X> <Y> <Z> <random_on_off> [<args>]

 # Create and define topography
```

```
set topo        [new Topography]

$topo load_flatgrid <X_length> <Y_length>

# god is a necessary object when wireless is used
create-god 1

# Effectively create Nodes, Links and NS addressing
$TOPOM tm_create_topo
```

If you are creating topologies without mobile nodes, you can save one extra level of hierarchy that could be used for domains. In this case, create TOPOMAN with `set TOPOM [new Topoman WIREDONLY]`

Displaying topologies

Here follows an instance of what TOPOMAN may display (same topology as before).

```
>--------------------- NS Topology ---------------------<

>----------------- Transit Domain 0 --------------------<
Transit Node 0: 0.0.0 (_o20) [Hier] :
  Site 1:
      Border Routers
          - 2: 1.0.0  (_o46) [Hier]
      Site Routers
          - 3: 1.1.0  (_o59) [Hier]
      Base Stations
          - 20: 1.2.0  (_o283) [BS]
          - 28: 1.3.0  (_o650) [BS]
          - 36: 1.4.0  (_o1010) [BS]
          - 44: 1.5.0  (_o1370) [BS]
  Site 2:
      Border Routers
          - 4: 2.0.0  (_o72) [Hier]
      Site Routers
      Base Stations
          - 21: 2.1.0  (_o329) [BS]
          - 29: 2.2.0  (_o695) [BS]
          - 37: 2.3.0  (_o1055) [BS]
          - 45: 2.4.0  (_o1415) [BS]
  Site 3:
      Border Routers
          - 5: 3.0.0  (_o85) [Hier]
      Site Routers
          - 6: 3.1.0  (_o98) [Hier]
          - 7: 3.2.0  (_o111) [Hier]
      Base Stations
          - 22: 3.3.0  (_o375) [BS]
          - 30: 3.4.0  (_o740) [BS]
          - 38: 3.5.0  (_o1100) [BS]
          - 46: 3.6.0  (_o1460) [BS]
Transit Node 1: 4.0.0 (_o33) [Hier] :
```

```
  Site 5:
      Border Routers
          - 9: 5.1.0  (_o137) [Hier]
      Site Routers
          - 8: 5.0.0  (_o124) [Hier]
      Base Stations
          - 23: 5.2.0  (_o421) [BS]
         - 31: 5.3.0  (_o785) [BS]
          - 39: 5.4.0  (_o1145) [BS]
          - 47: 5.5.0  (_o1505) [BS]
>-------------------------------------------------------<

>----------------- Transit Domain 1 --------------------<
Transit Node 10: 6.0.0 (_o150) [Hier] :
  Site 7:
      Border Routers
          - 12: 7.0.0  (_o176) [Hier]
      Site Routers
      Base Stations
          - 24: 7.1.0  (_o467) [BS]
          - 32: 7.2.0  (_o830) [BS]
          - 40: 7.3.0  (_o1190) [BS]
          - 48: 7.4.0  (_o1550) [BS]
  Site 8:
      Border Routers
          - 14: 8.1.0  (_o202) [Hier]
      Site Routers
          - 13: 8.0.0  (_o189) [Hier]
      Base Stations
          - 25: 8.2.0  (_o513) [BS]
          - 33: 8.3.0  (_o875) [BS]
          - 41: 8.4.0  (_o1235) [BS]
          - 49: 8.5.0  (_o1595) [BS]
  Site 9:
      Border Routers
          - 16: 9.1.0  (_o228) [Hier]
      Site Routers
          - 15: 9.0.0  (_o215) [Hier]
          - 17: 9.2.0  (_o241) [Hier]
      Base Stations
          - 26: 9.3.0  (_o559) [BS]
          - 34: 9.4.0  (_o920) [BS]
          - 42: 9.5.0  (_o1280) [BS]
          - 50: 9.6.0  (_o1640) [BS]
Transit Node 11: 10.0.0 (_o163) [Hier] :
  Site 11:
      Border Routers
          - 18: 11.0.0  (_o254) [Hier]
      Site Routers
          - 19: 11.1.0  (_o267) [Hier]
      Base Stations
          - 27: 11.2.0  (_o605) [BS]
          - 35: 11.3.0  (_o965) [BS]
          - 43: 11.4.0  (_o1325) [BS]
          - 51: 11.5.0  (_o1685) [BS]
>-------------------------------------------------------<
```

```
>-------------------- Mobile Nodes ---------------------<
1 mobile nodes:
Mobile node 52: 1.2.1 (_o1730) [MN]
>-------------------------------------------------------<

>------------------ Topology Summary -------------------<
2 administrative domains
8 total number of sites

52 total number of wired nodes
1 mobile nodes
53 total number of nodes
>-------------------------------------------------------<
```

NS addressing with 4 levels of hierarchy

TOPOMAN may alternatively produce NS addresses with 3 levels of hierarchy, or 4 levels. This reduces the memory comsumption of NS once it is instructed to compute the routing tables. As an instance, a simulation with a 500-nodes topology and the same parameters will take 37 minutes instead of 70, and will consume 100 MB of memory instead of 150.

If you want 4 levels of hierarchy, create TOPOMAN with `set TOPOM [new Topoman ENHANCED]`. The NS addresses are divided into 4 levels:

- First level identifies the sub-domain with the topology (i.e. the transit node). This gives the sub-domain prefix (the domain prefix is recorded internally)
- Second level identifies the site in the sub-domain (i.e. the stub). Transit nodes are in the $stubid0$. This gives the site prefix.
- Third level identifies the subnet within the site. This gives the subnet prefix. All border routers have $stubid <> 0$.
- Fourth level identifies the host within the subnet, i.e. the end-system. Routers have $hostid = 0$; mobile nodes have $hostid <> 0$.

You also need to compile NS with the enhanced route.cc and route.h files. You can not use 4 levels and 3 levels with the same binary (this would have required to change more code in TCL, then I just replaced the files). If you don't run simulations with mobile nodes, you don't need 4 levels of hierarchy.

2.3 Mobile IPv6 and IPv6

In this section, we describe our NS enhancements to support Mobile IPv6 [4].

As for the IPv6 protocol suite itself, we haven't implemented all the proper IPv6 features as this was not really necessary to simulate Mobile IPv6 (e.g. DHCPv6, Neighbor Discovery, etc). We only modified the header size, and we added Router Advertisements and Solicitations between BSs and MNs, encapsulation and decapsulation at all nodes, and processing of the routing extension header.

Mobile IPv6 processing is implemented as a set of new NS Agents and a set of existing NS Classifiers. Nodes that need Mobile IPv6 capabilities also comprise separate objects for encapsulation, decapsulation, routing header processing, etc. This obviously required changes to the way NS Nodes are configured.

As for the MN and the BS or HA, we rely on `Class MobileNode` as contributed by CMU. The ad-hoc routing Agent is replaced by the `Network Agent`. As for CNs, we make use of `Class Node`. We don't make use of class HierNode anymore. The structure of the Mobile IPv6 nodes is shown in fig. 2.2 and fig. 2.3.

Basically, the Mobile IPv6 protocol aims to advertise the current location in the topology to all its correspondent nodes. By default, packets are sent to the home network where they are intercepted by the Home Agent and encapsulated to the primary care-of address of the MN. If a CN has a care-of address for the MN, it sends the packet directly to the MN using a Routing Extension Header. All those features are implemented in NS. This requires encapsulation, decapsulation, routing extension header processing at all nodes by means of a set of `Classifiers`. If there is an entry in the *Binding Cache*, the source of the packet is checked. It its the local node, a routing extension header is inserted; if the source is not the local node, then the packet is encapsulated. In addition, the MN needs to decapsulate the packets and it also needs to determine what is its current access router (Base Station) by means of *Router Advertisements and Solicitations*.

2.3.1 Network Agent

This agent is used by BSs and MNs and replaces the ad-hoc routing Agent in `Class MobileNode`. It mainly instantiates the next hop variable in the NS packet header, verifies the TTL, and transmit broadcast packets (i.e. Router advertisements and Solicitations) to the MIPv6Agent.[1].

As for the MN, agent `NetworkMN` allows packets sent to the careof address to be forwarded to the port demux. It also monitors incoming packets and informs `MNAgent` about the source of the packet by making a call to the `MNAgent` (which results in a new entry in the Correspondent List, i.e. *the Binding Update List*).

2.3.2 Mobile IPv6 Agents

The Mobile IPv6 protocol itself is implemented as a set of NS Agents. The base class is `Class MIPv6Agent`, and derived classes are `Class MNAgent, Class CNAgent, and Class BSAgent`. The MIPv6 Agents make use of the existing features:

Router Advertisements and Solicitations Those are presently processed directly in the MIPv6 Agent (`MNAgent and BSAGent`. See section 2.3.1 and below.

[1] we are going to move the processing of Router Advertisements and Solicitations from the `MIPv6Agent` to the `Network Agent` when a proper Neighbor Discovery protocol is implemented

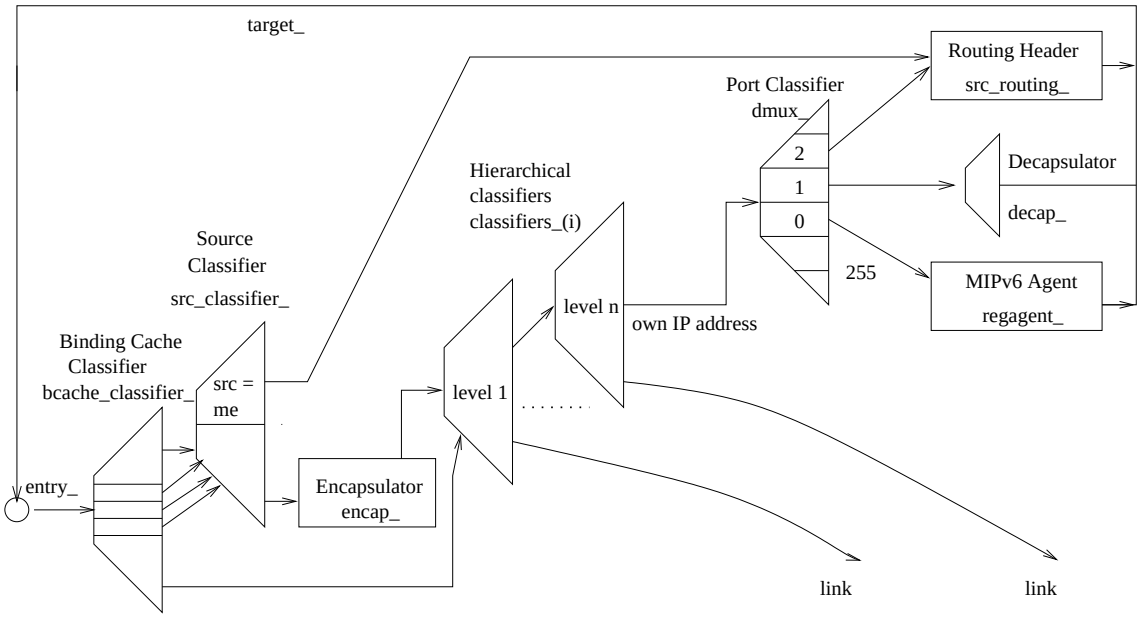

Figure 2.2: Wired Node Mobile IPv6 enable (Class Node)

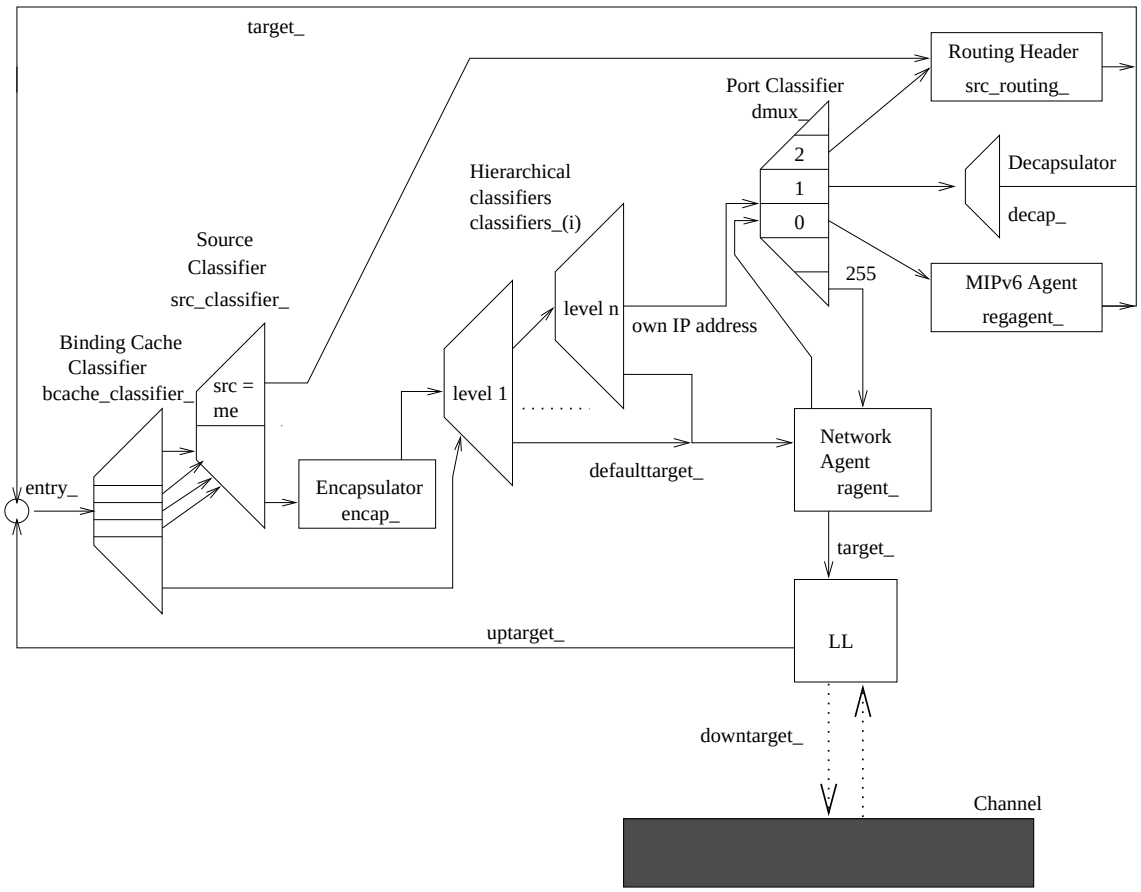

Figure 2.3: Wireless Node Mobile IPv6 enabled (Class MobileNode)

Base Station list This list is maintained by the MN thanks to Router Advertisements sent by the BSs. Entries are removed when the lifetime expires. The MN listens to Router Advertisements sent from Base Stations. When it receives a Router Advertisement from an unknown BS, it is recorded in the Base Station list for a $< lifetime >$ period. MN then gets a new care-of address and uses this BS as the default. The previous BS is added in the *Binding Update List*.

Getting a Care-of Address Addresses are composed by two components: the `network_id` which identifies the subnet and the location in the topology, and the `node_id` which identifies the node. The `network_id` of the careof address is therefore the prefix of the BS's address and identifies the visited subnet, whereas the `node_id` is the `node_id of the MN's home address % 128`. Modulo 128 allows for 128 simultaneous mobiles in the simulation (not tested). The ARP object at the MN is modified in order to reply to requests for the primary careof address of the MN.

Note: with this method of address allocation, there may be a conflict between one MN's home address and another mobile 's care-of address. This may be solved at the configuration set up by allocating a home address in a site where there are no mobiles.

Binding Update List This list maintained by the MN has two purposes: maintaining the list of active correspondent of the MN, and maintaining information about the BUs it sent. sing a single list avoids to manage two distinct and complementary lists since the common purpose if to know who should be sent a BU and which one was effectively sent.

An entry indicates the address of the destination, a flag specifying if this node should be sent a BU or not (not used in the implementation in your hands), the careof-address, time, sequence number, and lifetime of the BU last sent, plus additional information (type of node to distinguish between BS, HA, and CN).

As a list of correspondent, it lists the potential nodes to which a BU may be sent. Those nodes are whether automatically inserted by the `Network Agent`, or explicitly by the user from the OTcl interpreter, or automatically from the `MNAgent` itself:

- Automatically from the `Network Agent`. The `Network Agent` monitors all incoming packets and inserts an entry in the *MN's Binding Update List*. Deciding who should receive a BU is the MN's responsibility. Entries are kept in the list until explicitly removed, however, the entry is de-activated if no packets are received from the CN for a `MNAgent::reglftm_` seconds period of time. No BU is sent if the entry is "de-activated". The entry is re-activated when the MN receives a new packet from this destination.
- Explicitly from the OTcl interpreter. This concerns both CNs and HAs. Those entries are kept permanently, unless explicitly removed from the OTcl interpreter.
- Automatically from the Mobile IPv6 Agent itself. This concerns the previous BS only. When the MN registers with a new BS, it adds the previous BS in the list in order to establish forwarding. The entry is activated for `MNAgent::reglftm_` seconds, then it is automatically removed.

As a list of binding updates, it indicates the nodes to which Binding Updates where effectively sent. Entries in the list may be activated or not. If activated, and if *Routing Optimization* and *Forwarding from the previous BS* are set ON (default), BUs will be sent respectively to *CNs* and the *previous access router* in addition to the Home Agent. An entry gives information about the last sent BU to this particular node.

Routing Optimization `Agent/MN set rt_opti_ 0` switches off routing optimization between CN and MN. If flag is on, BUs are automatically sent to the CNs.

Forwarding from the previous BS `Agent/MN set bs_forwarding_ 0` switches off forwarding from the previous access router. By default, the MN sends a BU to its previous access router. However, packets may enter a loop with some simulations: this may happen when MN attaches back to a BS it was already attached just before, i.e. after a time no longer than the lifetime of the BU it sends to the previous BS to establish forwarding. We leave the solution of this "Mobile IPv6" issue to the user since the Mobile IPv6 only suggests forwarding from the previous access router, but does not have a word about how to de-register this registration. Then, if you don't want to drop packets, set this flag ON, and add the necessary features to de-register the previous binding, otherwise some packets may become bigger and bigger. IMHO, this is not a bug of this implementation, this is a bug of the specification.

Sending Binding Updates The MN sends BUs when it obtains a new CoA and when the periodic timer has expired. Currently, BUs are sent to everyone at the same time. However, any user may decide to send BU at CNs and HA at different rates by rewriting the procedure `MNAgent::send_standard_bu`.

The MN sends a BU to all nodes registered in its *Binding Update List* for which the entry is activated. All entries have an activation flag that says if the destination should be sent a BU (we can therefore interrupt sending BUs to a particular node for a certain amount of time - this feature is possible, but no procedure is available yet). If the entry is activated, a BU is sent and the entry is updated to monitor what was the careof address used, at what time it was sent, and what lifetime it was given (likely `MNAgent::reglftm_` seconds).

Binding Cache When a node receives a Binding Update, an entry is added in the Binding Cache. As a result, the "routing table" must be updated in order to redirect packets to the specified care-of address (using a Routing Extension Header or Encapsulation). In NS, the routing table is inserted as a set of `Classifiers`. Then, we added a set of new classifiers to determine if the packet should be redirected and how.

History List Only useful to control the behavior of Mobile IP - this could be removed easily if needed. For long run simulations, this list may become large and consume memory. Usually, an entry which is removed from the Binding Update List is inserted in the History List. This mainly concerns all the BSs to which a MN was registered, and the CN removed from the OTCL interpreter.

Monitoring the behavior of Mobile IPv6 The user can dump the content of the BS, BU and the History lists. This is very useful to collect statistics and to monitor the behavior of Mobile IP. As an additional feature, the emission and reception of Mobile IPv6 signaling packets can be traced on the default output by setting the `MIPv6Agent print_info_ variable to 1`.

2.3.3 OTCL implementation

All nodes in the simulation may be Mobile IPv6 enabled using `proc node-config -mipv6` (see file proc-mipv6-config.tcl). This set the flag `mipv6_` that indicates Mobile IPv6 capabilities will be added when the node instance is created. For each node, you also need to specify what kind of Mobile IPv6 Agent you want, i.e. if it's a CN, a BS, or a MN (MAP for HMIPv6 - later). For a CN, `$ns node-config -mipv6 -mipagent CN`

Procedures in file *proc-mipv6-config.tcl* are also used by TOPOMAN. Then, you may rewrite them at will to match your network configuration. Note that those procedures are currently defined for Wide-Area mobility. A distinct channel is created for each site. If you want BSs in distinct sites ("sites as defined by TOPOMAN) to listen on the same site, simply modify the procedures `create-base-station`.

2.3.4 Missing Pieces

- Neighbor Discovery
- DHCPv6
- Binding Update Piggybacking
- Binding Requests

2.4 Wide-Area Mobility

2.4.1 Local and Global Mobility

As for local mobility within a site, we make use of the existing NS mobile networking features. See the NS documentation for details.

As for global mobility between sites, all sites are associated with the same geographical grid but with a distinct channel. As a result, all BSs and MNs in the same site communicate on the same channel, and have the same address prefix. BSs are `Node/MobileNode` with no motion (speed is set to 0). A MN with no motion is a MN which doesn't move within the site.

2.4.2 Movements

A MN may move by two means: geographically, and topologically.

Geographical moves correspond to local mobility, i.e. mobility within a site covered by a geographical area. The MN moves within the grid. Those movements may be set randomly or using the *setdest* command. As a result of geographical moves, the MN may be in the coverage area of distinct Base Stations. As such, MN listens to Router Advertisements.

Topological moves correspond to global mobility, i.e. from one site to another. As a result of such a move, the mobile now listens on a new channel, the one corresponding to the site it enters. The MN does not actually changes its geographical co-ordinates as a result of such a move, but it changes its location in the topology. It therefore hears Router Advertisements sent by Base Stations in the new site, i.e. with a different address prefix.

2.4.3 Important note

Global mobility was not tested with more than one MN. Actually, at least one change is required if you want more than one mobile: in order to limit the number of events scheduled, only Base Stations in the visited site do emit Router Advertisements. BSs in the previous site are instructed to stop Router Advertisements. Then, you may change the procedure `proc enter-site` if you want more than one MN.

Bibliography

[1] Ken Calvert, Matt Doar, and Ellen W. Zegura. Modeling Internet Topology. *IEEE Communications Magazine*, 35(6):160–163, June 1997.

[2] Ken Calvert and Ellen Zeggura. GT-ITM: Georgia Tech Internetwork Topology Models. Technical report, Georgia Institute of Technology, College of Computing, 1996. http://www.cc.gatech.edu/projects/gtitm/.

[3] Kevin Fall and Kannan Varadhan. NS notes and documentation. Technical report, The Vint Project, UC Berkeley, LBL, USC/ISI, Xerox PARC, 2000. "http://www.isi.edu/nsnam/ns/index.html".

[4] David B. Johnson and C. Perkins. Mobility Support in IPv6. Internet Draft draft-ietf-mobileip-ipv6-13.txt, Internet Engineering Task Force (IETF), November 2000. Work in progress.

[5] Ellen W. Zegura, Ken Calvert, and Samrat Bhattacharjee. How to Model an Internetwork. In *Proceedings of IEEE Infocom*, San Francisco, CA, 1996.