Extensions to *ns-2*
Notes and Documentation

Laurent Paquereau, ⟨laurent.paquereau@q2s.ntnu.no⟩

Centre for Quantifiable Quality of Service in Communication Systems[1],
Norwegian University of Science and Technology, Trondheim, Norway

October 29, 2009

[1] "Centre for Quantifiable Quality of Service in Communication Systems, Centre of Excellence" appointed by The Research Council of Norway, funded by the Research Council, NTNU, UNINETT and Telenor. http://www.q2s.ntnu.no

Contents

Chapter 1

Introduction

This document details extensions to *ns-2* distributed as a patch[1] for ns-2.34 under the terms of the GNU General Public License version 2 as published by the Free Software Foundation[2]. Rationales for these extensions are given in [3][3] and [2].

In short, these extensions aim to provide a more flexible and better integrated support for wireless and mobile networks and to support *multi-** networks, that is networks where nodes may have multiple interfaces, possibly of different types, where nodes may communicate over multiple channels, where multiple routes to a destination may be available, where nodes may be interested in reaching a given resource/function/service rather than a specific node. Examples of such networks include wireless mesh and sensors networks.

These extensions provide a generic network layer architecture and a generic framework for implementing routing and forwarding protocols over one or more interfaces and regardless of their type. All the new objects presented in the following chapters are integrated in the standard *ns-2* `Node`, see [4]. A unique object and layout is used regardless of the type of interfaces and the routing/forwarding protocols running on the node. Capabilities (e.g. mobility) are added to the standard *ns-2* `Node` by means of modules. The new layout of a `Node` is shown in Figure 1.1. Dashed lines indicate objects and paths existing in the standard *ns-2* `Node`.

A complement to this documentation is the doxygen[4] documentation that can be generated by running `doxygen` in the q2s_doc directory once the patch has been applied.

[1] Available on-line at `http://www.q2s.ntnu.no/~paquerea/ns`
[2] `http://www.gnu.org/licenses/gpl.txt`
[3] Note that the design and the implementation have been refined and extended since the publication of [3].
[4] `http://www.stack.nl/~dimitri/doxygen/`

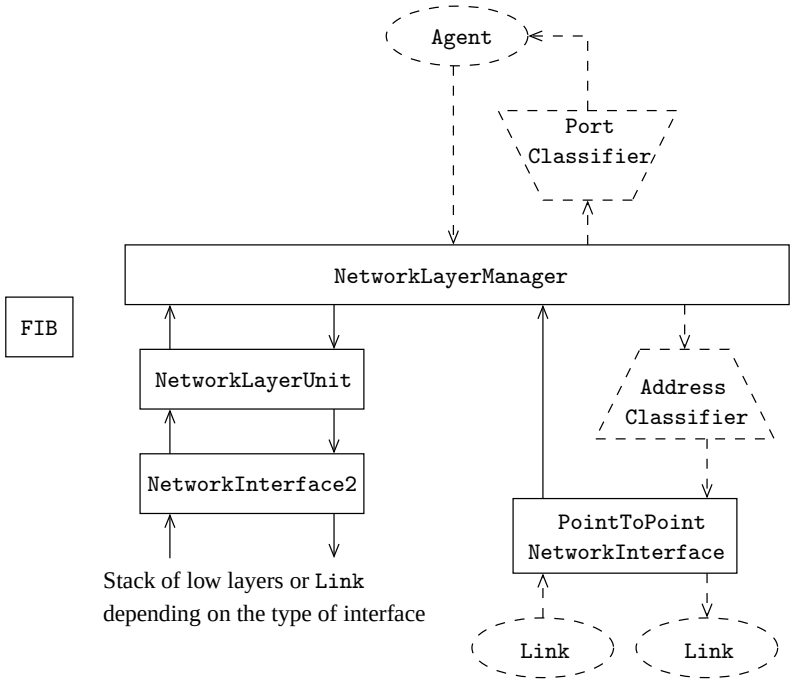

Figure 1.1: Node layout

Chapter 2

Modules

Directory: q2s_modules

The idea of a *module-based* node [3] is to provide *capabilities* (e.g. mobility) to a standard *ns-2* `Node` by means of *modules*[1,2] rather than deriving new node objects inheriting from `Node` (e.g. `MobileNode`) as it has been done so far. This is to provide a generic and flexible design and to make it easier to support new features.

Modules are enabled/disabled when configuring `Nodes`. A static list of enabled modules is kept in the OTcl `Node` class. Whenever a `Node` is created, enabled modules are instantiated and registered for this particular `Node`. A `Node` keeps a pointer on each module registered.

Modules are enabled/disabled using the `node-config` procedure. For instance, to enable/disable the position module presented below:
`<Simulator instance> node-config -position on/off`

2.1 Base class

Files: ns-module.{cc,h}

`NsModule` is a base class for modules. An `NsModule` has a name which is used to maintain a list of registered modules on a `Node` in OTcl.

2.1.1 Commands at a glance

Following is a list of internal commands:

To get the module name:
`$module module-name`
called by `Node::register-module/unregister-module` (OTcl).

[1] By *module* we refer to specific types of objects in *ns-2*, namely `RtModule` and our generic `NsModule` object. Note however that the term "module" has sometimes been used as a generic term (wireless/mobile support as for example been referred as "module").

[2] Until now only routing modules had been defined (`RtModule/`) so implicit naming was used (e.g. `Manual` for `RtModule/Manual`). It is not the case anymore, hence explicit names must be used (return of `module-name` Tcl command).

The two following commands must be overridden by sub-classes to correctly register the module in Node (C++) and check for any module dependency.

To register a module on a Node:
`$module register <Node>`
called by `Node::register-module` (OTcl).

To unregister a module:
`$module unregister`
called by `Node::unregister-module` (OTcl).

2.2 PositionModule

Files: position-module.{cc,h}, q2s_tcl/position.tcl

A PositionModule is used to set and store the coordinates of a Node on a Topography [m]. Position information is required when dealing with wireless communication since the communication properties (e.g. propagation delay, received signal strength, modulation used, fading, etc.) depend on the distance between the transmitter and the receiver.

The position of a Node can only be set at its creation. During a simulation, the position of a Node may changed if mobility is enabled for this Node, see 2.3.

2.2.1 Commands at a glance

Following is a list of commands used in simulation scripts:

Positioning procedures `set-x/y/z/position/position-logger` are defined for a Node to avoid the need for explicit references to position modules in scenario scripts. Calls to these procedures are simply forwarded to the `Module/Position` commands of same name.

To set the position of a Node to $(x,\ y,\ z)$:
`$node set-position <x> <y> <z>`
It is also possible to set each of the coordinates individually using `set-x`, `set-y` and `set-z`. The position must always be within the boundaries of the topography. Coordinates are in meters.

Following is a list of internal commands:

To disable the `set-` commands once the position of a Node is set:
`$position_module is-set`

To register a module on a Node:
`$position_module register <Node>`
called by `Node::register-module` (OTcl).

To unregister a module:
`$position_module unregister`
called by `Node::unregister-module` (OTcl).

To get a reference on the position module (`Module/Position`) registered on a `Node` if any:
`$node position-module`

To get a string containing the position of a `Node` to write in a NAM trace file:
`$position_module str-nam-position`
called in `Simulator::dump-namnodes` (OTcl).

2.3 MobilityModule

Files: mobility-module.{cc,h}, q2s_tcl/mobility.tcl

A `MobilityModule` provides mobility capability to a `Node`. That is, it keeps track of the (2D-)destination and speed of the node [m.s^{-1}]. A `PositionModule` is required in order to register a `MobilityModule` on a `Node`.

2.3.1 Module configuration

The configuration of a `MobilityModule` is done using the `mobility-config` procedure. The usage of this procedure is similar to that of `node-config`, that is:
`<Simulator instance> mobility-config -<option> <value>`

The available options are given in Table 2.1.

option	description
update-interval	position update interval used by `WirelessChannel2` [s] (default is 1)
trace	mobility tracing on/off

Table 2.1: Available options for `mobility-config`

`update-interval` sets a static member of `MobilityModule` (`updateInterval_`). When `WirelessChannel2` is used, `updateInterval_` is used to decide how often the position of a node far away from any communication area ("far away" depends on `WirelessChannel2::distCST_` which itself depends on the propagation model used) is updated. It affects the uncertainty range[3].The larger `updateInterval_` the larger the uncertainty range. Any node within the uncertainty range of a transmitting node must receive a copy of the packet transmitted to ensure that any node that actually detects the transmission (i.e. actually within carrier sensing range i.e., as it is implemented, within a square of sidelength twice the uncertainty range and centered at the sending node, or, in other words, distant of at most two times the uncertainty range using Manhattan distance) receives a copy. The position of such a node is updated at the time of transmission regardless the update interval because it needs to be exact to calculate the propagation delay. On the other hand, the position of a node beyond the uncertainty range of any transmitting node is only updated every `updateInterval_`. If `updateInterval_` is 0.0, the position of every node is updated each time a packet is transmitted and thus a copy is forwarded only to the nodes that actually are within the carrier sensing range. This may not be efficient since updating the position of a node may be demanding (read the list of wireless interfaces at each node, read and update the lists of interfaces for all the channels...). Therefore this parameter must be chosen as a trade-off between the number of unnecessary packet copies and the processing time due to position updates. It should be adjusted depending on the level of mobility. This is similar to what is done by a `WirelessChannel` with `MobileNode`. `updateInterval_`corresponds to `XLIST_POSITION_UPDATE_INTERVAL`. The "safety" of 5 used in `WirelessChannel::sendUp()` corresponds to the uncertainty range (it is nothing more than

[3]Because the position of a node may not be up to date due to `updateInterval_` a node at a distance $d \in (\text{range}, \text{range} + \texttt{maxSpeed\_} \times \texttt{updateInterval\_})$, where `maxSpeed_` is the largest speed observed, may actually be at a distance $d' \leqslant$ range. The uncertainty range $\mathrm{UR} = \text{range} + \texttt{maxSpeed\_} \times \texttt{updateInterval\_}$ is the the minimal distance such that $d > \mathrm{UR}$ implies $d' >$ range.

XLIST_POSITION_UPDATE_INTERVAL×MAX_SPEED). However all of this is far from being explicitly commented either in the source code or in [4]. The assumptions made on the speed presented in Section 8 in [1] still apply.

When mobility tracing is turned on, mobility lines are written to the trace file. Those lines correspond to the 'M' lines found in standard trace files and produced by MobileNode::log_movement(), see Chapter 16 in [4]. The fields are given below.

Event fields

M mobility event

-t time

Node fields

-Na node address

-Nx current x coordinate [m]

-Ny current y coordinate [m]

-Nz current z coordinate [m]

Mobility fields (only if the node is not at destination)

-Mx destination x coordinate [m]

-My destination y coordinate [m]

-Ms speed [m.s^{-1}]

It is possible to log the mobility of one or more nodes using a MobilityLogger. In this case, lines are written to a log file. See Chapter 5 for the difference between trace files and log files.

2.3.2 Commands at a glance

Following is a list of commands used in simulation scripts:

Similarly to positioning procedures, mobility procedures are defined for a Node to avoid the need for explicit references to mobility modules in scenario scripts. Calls to these procedures are simply forwarded to the Module/Mobility commands of same name.

To set the destination of a Node to $(x,\ y)$ (in meters):
`$node set-destination <x> <y>`

To set the speed [m.s^{-1}] of the a Node:
`$node set-speed <speed>`

To set both destination (coordinates in meters) and speed [m.s^{-1}] at once:
`$node set-mobility <x> <y> <speed>`

Note that the node will actually move only if the destination differs from the current position and the current speed is strictly positive.

To set a MobilityLogger:
`$node set-mobility-logger <MobilityLogger> [<rate>]`
If a rate parameter is given, a new line will be written at this rate [s^{-1}] to the log file. For this a LogMobilityTimer is used. This corresponds to the LogTimer OTcl class presented in Chapter 16 in [4].

Following is a list of internal commands:

To register a module on a `Node`:
`$mobility_module register <Node>`
called by `Node::register-module` (OTcl).

To unregister a module:
`$mobility_module unregister`
called by `Node::unregister-module` (OTcl).

To get a reference on the mobility module (`Module/Mobility`) registered on a `Node` if any:
`$node mobility-module`

To set the trace object:
`$mobility_module set-trace <BaseTrace>`
called by `Module/Mobility::add-trace` (OTcl)

To log mobility when destination and/or speed change:
`$mobility_module log-mobility`

To trace when destination and/or speed change:
`$mobility_module trace-mobility`
This commands corresponds to `Node/MobileNode::log-movement`, see Chapter 16 in [4].

To NAM-trace when destination and/or speed change:
`$mobility_module nam-trace-mobility`

2.4 EnergyModule

Files: energy-module.{cc,h}, q2s_tcl/energy.tcl

An `EnergyModule` provides energy-support capability to a `Node` using an `EnergyModel`. The energy level decreases with time and packet transmissions/receptions. See Chapter 19 in [4].

Disclaimer. The `EnergyModule` is *not functional* even if some code already refers to methods defined in this class. It must be considered as a bare attempt to provide a framework in line with position and mobility modules. Further modifications (of the module itself, of `EnergyModel` and at the `Mac` and `WirelessPhy` layers) are required. In particular, the energy and properties of a node and the state and properties of a specific interface should be dissociated. Currently, an error is raised in OTcl to prevent the use of an energy module (see q2s_tcl/energy.tcl).

2.4.1 Module configuration

The configuration of a `EnergyModule` is done using the `energy-config` procedure. The usage of this procedure is similar to that of `node-config`, that is:
`<Simulator instance> energy-config -<options> <value>`

The available options are given in Table 2.2.

option	description
energy-model	`EnergyModel` type
initial-energy	[J]
level1	[J]
level2	[J]
rx-power	[W]
tx-power	[W]
idle-power	[W]
sleep-power	[W]
transition-power	[W]
transition-time	[s]
trace	node energy tracing `on/off`

Table 2.2: Available options for `energy-config`

`energy-model`, `initial-energy`, `level1`, `level2` and `energy-trace` are per node parameters. `rx-power`, `tx-power`, `idle-power`, `sleep-power`, `transition-power` and `transition-time` are per physical layer parameters.

When energy tracing is turned on, energy lines are written to the trace file. Those lines correspond to the 'E' lines found in standard trace files and produced by `MobileNode::log_energy()`, see Chapter 16 in [4]. The fields are given below.

Event fields

E energy event

-t time

Node fields

-Na node address

-Ne energy level

It is possible to log the energy of one or more nodes using an `EnergyLogger`. In this case, lines are written to a log file. See Chapter 5 for the difference between trace files and log files.

2.4.2 Commands at a glance

Following is a list of commands used in simulation scripts:

To set an `EnergyLogger`:
`$node set-energy-logger <EnergyLogger> [<rate>]`
If a `rate` parameter is given, a new line will be written at this rate [s^{-1}] to the log file. For this a `LogEnergyTimer` is used. This corresponds to the `LogTimer` OTcl class presented in Chapter 16 in [4].

Following is a list of internal commands:

To register a module on a `Node`:
`$energy_module register <Node>`
called by `Node::register-module` (OTcl).

To unregister a module:
`$energy_module unregister`
called by `Node::unregister-module` (OTcl).

To get a reference on the energy module (`Module/Energy`) registered on a `Node` if any:
`$node energy-module`

To set the trace object:
`$energy_module set-trace <BaseTrace>`
called by `Module/Energy::add-trace` (OTcl)

To trace the energy level of a `Node`:
`$energy_module trace-energy`
called by `Module/Energy::init` to trace the initial energy level of a `Node`.

NB. Other commands are defined in `EnergyModule` but need to be modified when the energy and properties of a node (defined in an `EnergyModule`) and the state and properties of a specific interface are dissociated. See disclamer above.

Chapter 3

Network interfaces

Directory: q2s_network-interfaces

To send packets to other nodes in a network, a node uses one or more network interfaces, possibly of different types. This chapter presents network interface objects and how to configure them. How to add an interface to a node is described in Chapter 6.

3.1 State of the art (ns-2.34)

Protocols for wired networks use `Link` objects and address classifiers use pointers onto the first object (`NsObject`) of the link. Routing agents for the `MobileNode` use a pointer onto a link layer object. Satellite nodes use `NetworkInterface` and `LinkHead` objects for incoming and outgoing packets, respectively. A `LinkHead` contains pointers onto all the objects that form the interface stack and a `Node` has a list of `LinkHead` and a list of `NetworkInterface` attached to it.

There is no generic network interface object and no record of all the interfaces attached to a node. Therefore there is no generic way of interacting or addressing any network interface. Addresses are assigned to nodes, not to interfaces. Identifiers are assigned to `NetworkInterface` but are only used to tag packets.

3.2 Base objects

Files: network-interface.{cc,h}

3.2.1 `NetworkInterface2`

`NetworkInterface2`[1] is a generic network interface object. Each `NetworkInterface2` attached to a `Node` is assigned a unique identifier. All packets sent or received on this interface are tagged with the interface identifier (`iface_` field in the packet common header).

[1] As a convention we use 2 for a class that aims to replace the existing class of same name.

A list of all the interfaces attached to a Node is maintained by the NetworkLayerManager, see Chapter 4. There is no a priori limitations on the number or the type of NetworkInterface2 attached to a Node.

A NetworkInterface2 is either up (working) or down (failed or stopped). When bringing an interface down, all the components of the stack are reinitialized, all the packets in the stack are discarded and all the routes on this interface in the FIB, see 4.3.1, are deleted. To support this functionality, every layer object constituting a network interface must implement a reinit method and care must be taken to delete any scheduled packets at any layer of the interface stack.

Commands at a glance

Following is a list of commands used in simulation scripts:

To bring the interface up or down, respectively:
`$interface up/ $interface down`

To reset the interface:
`$interface reset`

To get the NetworkLayerUnit the interface is attached to, if any:
`$interface get-network-layer-unit`

To get the Node the interface is attached to:
`$interface get-node`

To get/set the preference value of the interface:
`$interface get-preference/$interface set-preference <value>`

For debugging purposes, to print out information about all the interfaces attached to a node:
`$node dump-interfaces`

Following is a list of internal commands:

To attach the interface to the given NetworkLayerUnit::
`$interface set-network-layer-unit <NetworkLayerUnit object>`
Called in the Node::add-interface (OTcl), see 6.1.1.

3.2.2 FullStackNetworkInterface

FullStackNetworkInterface is also a generic object. It inherits from NetworkInterface2 but in addition keeps a pointer onto all the objects which compose the network interface stack (LL and below). All the elements in the stack get pointers to each other through the FullStackNetworkInterface and do not contain pointers to specific elements (see inline functions in .h files for the different components). The objects constituting an interface protocol stack except the channel cannot be replaced once set (changing the channel implies re-initializing the interface). NetworkInterface2 does not have pointers to the different components of the stack. This is because when using Link objects, see Chapter 6 in [4], to model a point-to-point link there is no protocol stack as such associated with the interface.

3.2.3 Commands at a glance

Following is a list of commands used in simulation scripts:

To get the x={`channel`, `phy`, `mac`, `queue`, `ll`, `arp-table`, `in-error`, `out-error`} object of the interface, if any:
`$interface get-`x

Following is a list of internal commands:

To set the x={`channel`, `phy`, `mac`, `queue`, `ll`, `arp-table`, `in-error`, `out-error`} object of the interface:
`$interface set-`x
Called in the `Node::add-interface` (OTcl), see 6.1.1.

3.3 PointToPointInterface

Files: point-to-point-interface.{cc,h}

A `PointToPointInterface` is the interface (extra layer) between a `Node` and a `Link`. It is either the `head_` or the `tail_` of a `SimpleLink`. It updates the direction field in the common header. The direction of outgoing packets is DOWN, the direction of incoming packets is UP and the direction of packets on the `Link` is NONE.

`PointToPointInterface` inherits from `NetworkInterface2` since no protocol stack as such is present when using a point-to-point `Link`. The characteristics of the `Link` the interface is attached to (cost, delay, queue length etc.) maintained in OTcl (ns-link.tcl) are accessible through the interface object.

Bringing down/up a `PointToPointInterface` is equivalent to bringing down/up the link to which it is attached (the neighbouring interface is therefore brought down/up at the same time).

3.3.1 Commands at a glance

Following is a list of internal commands:

sets the x={`link`, `queue`, `dynamics`} object of the interface:
`$interface set-`x
Called in `SimpleLink::connect` (OTcl) itself called by `Simulator::add-simplex- link` (OTcl), see Section 6.2.1.

To set the link cost:
`$interface set-link-cost`
Called in `Link::cost` (OTcl). Remember that the cost of a duplex link is to be set separately for each direction of a the link.

3.4 WirelessInterface

Files: wireless-interface.{cc,h}

WirelessInterface[2] inherits from FullStackNetworkInterface. A WirelessInterface is attached to a Wireless Channel2 and must be attached to a Node with a PositionModule since the position of nodes is used to decide which nodes are able to communicate with each other and the properties of the link e.g. delay, received signal strength etc.

WirelessChannel2 (Channel/Wireless) basically copies the functionality of WirelessChannel (Channel/Wireless Channel) (see channel.h). Differences include:

- WirelessChannel2 uses a list of WirelessInterfaces instead of a list of (Mobile)Nodes.
- The list is still sorted according to the position (x-coordinate) of the node. However, nextX_/prevX_ are not used and the list is not sorted if distCST_ is maximal (e.g. when the propagation model used is shadowing).
- distCST_ is defined as a non-static member so that it may be different for different channels (if the maximal transmission power is different).

3.5 Interface configuration

File: q2s_tcl/interface.tcl

Interfaces are configured using the interface-config procedure of the Simulator class. The syntax is, similarly to node-config, <Simulator instance> interface-config -<option> <value>. The available options are given in Table 3.1.

option	description
phy	physical layer type e.g. WirelessPhy
mac	mac layer type e.g. Mac/802_11
ifq	interface queue type e.g. Queue/DropTail/PriQueue
ifq-length	interface queue length e.g. 50
ll	link layer type e.g. LL
antenna	antenna type e.g. Antenna/OmniAntenna
channel	Channel/Wireless object
propagation	Propagation object
phy-trace	physical layer tracing on/off
mac-trace	mac layer tracing on/off
eot-trace	mac end of transmission tracing on/off
ifq-trace	interface queue level tracing on/off
net-trace	network layer tracing on/off
in-error	ErrorModel object
out-error	ErrorModel object

Table 3.1: Available options for interface-config

[2] formerly called WirelessStackInterface in [3].

Chapter 4

Network layer

The network layer architecture presented in this chapter is composed of two levels: a NetworkLayerManager, see Section 4.1 on top of one or more NetworkLayerUnits, see Section 4.2. These new objects are integrated in the Node layout as shown in Figure 1.1. Only one NetworkLayerUnit is shown and only one NetworkInterface2 is attached to this NetworkLayerUnit. However, there is no restriction either on the number of NetworkLayerUnits on a Node, or on the number or type of interfaces attached to such unit.

In addition, a forwarding information base (FIB) is used on each node to store routes. See Figure 1.1. A generic object (NetworkInformationBase) for storing routing/forwarding information is described in Section 4.3. A similar structure for storing information about neighbouring nodes (NeighborInformationBase) is presented in Section 4.4.

This architecture aims at providing a generic support for *multi-** networks [2] and a generic framework for implementing routing and forwarding protocols over one or more interfaces and regardless of their type.

4.1 NetworkLayerManager

Files: network-layer-manager.{cc,h}

The purpose of the NetworkLayerManager is twofold:

- to maintain a list of NetworkInterface2 and a list of NetworkLayerUnits. When a NetworkInterface2 is added to a Node, the NetworkLayerUnit it is attached to must also be specified. The NetworkLayerManager keeps a list of all the NetworkInterface2 attached to the Node and assigns a unique identifier to each of them. The NetworkLayerManager also maintains a mapping between interface identifiers and the NetworkLayerUnit the interface is attached to[1].
- to handle data packets generated locally and received from neighbour nodes. Upon reception of a packet, the Network LayerManager decides what to do with the packet. If the packet is at destination, the packet is simply forwarded upwards to the port classifier. If the packet is not at destination, it is either forwarded or discarded. Which route/interface the packet should be forwarded on is determined by the ManagerForwardingPolicy attached to the NetworkLayer Manager, if any. If the ManagerForwardingPolicy is not able to select any route to send the packet on or if no ManagerForwardingPolicy is associated to the NetworkLayerManager, the packet is forwarded to the Network

[1] In reality, downTarget_ inherited from NDownBiConnector is used for this. See q2s_common/ndown-biconnector.h. NDownBiConnector is a bi-directional connector with one up-target and N down-targets. It may be seen as an extended version of BiConnector or Classifier.

LayerManager's default target if any, otherwise it is discarded. If tunneling to gateways is turned on and the route returned by the ManagerForwardingPolicy is a route to a gateway, the packet is encapsulated[2] and sent to the gateway.

A NetworkLayerManager is attached on a Node at the Node's creation when the network-layer-manager option has been turned on using the node-config command (default is off, i.e. legacy behaviour).

4.1.1 ManagerForwardingPolicy

The purpose of a ManagerForwardingPolicy is to select which route should be used to forward a packet. This decision is primarily based on the information stored in the FIB. If more than one entry are found for the packet source and destination, different criteria may be used to decide which route should be used. For instance, using the LowestMetricAndPreference Policy the route with the lowest metric is used. In case of a tie, the most preferred route is chosen. If there is still a tie, the route on the most preferred interface is chosen. If there is still a tie, the most recent route is used. Another implemented policy is PreferenceAndLowestMetricPolicy. In this case, the preference value of a route is used before the route metric.

4.1.2 NetworkLayerManager configuration

The configuration of the NetworkLayerManager is done using the node-config command of Simulator. The available options for configuring the NetworkLayerManager are given in Table 4.1. They will have an effect only if the network-layer-manager option is also turned on.

option	description
manager-forwarding-policy	ManagerForwardingPolicy/ subclass name []
tunnel-to-gateway	tunneling to gateways on/off [off]

Table 4.1: Available options for node-config for configuring the NetworkLayerManager

4.1.3 Commands at a glance

Following is a list of commands used in simulation scripts:

To set the manager's default target:
$network_layer_manager default-target <NsObject>

Following is a list of internal commands:

To set the forwarding policy:
$network_layer_manager forwarding-policy <ManagerForwardingPolicy>
called by Node::init (OTcl) using the node-config value for the corresponding option.

To turn on/off tunneling to gateway nodes:
$network_layer_manager tunnel-to-gateway <on/off>
called by Node::init (OTcl) using the node-config value for the corresponding option.

[2] See EncapsulatedPacket in q2s_common/encapsulated-packet.h.

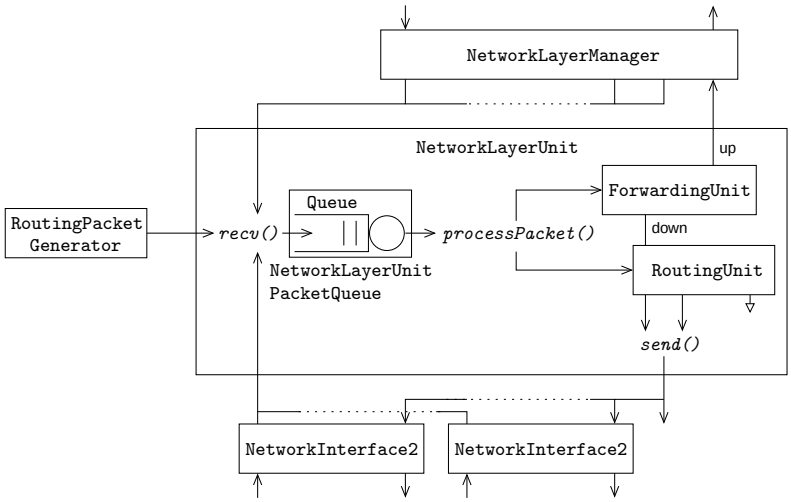

Figure 4.1: Detailed structure of a `NetworkLayerUnit`

To add an interface to the network-layer-manager:
`$network_layer_manager add-interface <NetworkInterface2> <NetworkLayerUnit>`
called by `Node::add-interface` (OTcl). The second parameter to this procedure is the `NetworkLayerUnit` the interface is attached to. For backward compatibility, the node's entry point may be given as a target instead of the `NetworkLayerUnit` the interface is attached to.

4.2 `NetworkLayerUnit`

Files: network-layer-unit.{cc,h}

A `NetworkLayerUnit` is an instance of a routing/forwarding protocol operating on a `Node` and running over one or more `NetworkInterface2`. It may contain a `ForwardingUnit` and a `RoutingUnit`.

A `NetworkLayerUnit` receives both incoming and outgoing packets on any interface attached to it. Packets are then dispatched internally. The `ForwardingUnit` handles data packets, the `RoutingUnit`, if any, routing packets[3]. A `Queue` may be used to enable simulation of processing time. Note that forwarded packets (not generated by the local node) are only queued on their way up the stack.

Details of the structure of the `NetworkLayerUnit` are shown in Figure 4.1.

`NetworkLayerUnits` are to be created in simulation scripts. `ForwardingUnits` and `RoutingUnits` on the other hand will usually be created in the (OTcl or C++) constructor of a `NetworkLayerUnit`.

Commands at a glance

Following is a list of commands used in simulation scripts:

To create a `RoutingPacketGenerator` and attach the generator to the `NetworkLayerUnit`:

[3] more precisely, the packets of type equals to the type of packets associated to the `NetworkLayerUnit` the `RoutingUnit` is attached to and not packets which type is classified as `ROUTING`.

`$network-layer-unit create-generator <RoutingPacketGenerator/ subclass name>`

To set the preference value:
`$network-layer-unit set-preference <int>`
This preference value may be used for routes via this `NetworkLayerUnit` and/or to decide which `NetworkLayerUnit` to forward a packet to when no route is found in the `FIB` for a packet.

To get a reference on the `RIB`:
`$network-layer-unit get-rib`

To set a `Logger`, see Section 5.2:
`$network-layer-unit set-logger <Logger>`

To use of LL/MAC feedback (callback):
`$network-layer-unit use-llfeedback`
`NetworkLayerUnit::onLLFeedback` is then called back on LL/MAC notification.

To decide whether a `Queue` should be used or not and, in the case it should, to configure the `Queue` change NetworkLayerUnit class defaults (see q2s_tcl/defaults.tcl). Configuration variables are: `queue-type_`, `queue-limit_`, `service-time-data_` and `service-time-routing_`. If `queue-type_` is set to `""` no queue is used. `NetworkLayerUnitPacketQueue` processes only one packet at a time and service times are deterministic.

Following is a list of internal commands:

To attach a `NetworkInterface2` to the `NetworkLayerUnit`:
`$network-layer-unit add-interface <NetworkInterface2>`
called by `Node::add-interface` (OTcl)

4.2.1 ForwardingUnit

Files: forwarding-unit.{cc,h}

A `ForwardingUnit` handles data packets in a `NetworkLayerUnit`. Incoming packets are passed upwards to the `Network LayerManager`, outgoing packets should be passed to the output interface chosen based on the information set by the `NetworkLayerManager` or/and contained in the local `RIB`, if any, and the forwarding protocol implemented. In the base class, outgoing unicast packets for which the output interface is unknown are simply discarded. This behaviour should be changed appropriately in the case for instance of a reactive routing protocol, where the packet should be buffered and a route discovery procedure triggered.

If no `ForwardingUnit` has been created and data packets are to be handled, a base `ForwardingUnit` is created by default.

4.2.2 RoutingUnit

File: routing-unit.h

A `RoutingUnit` handles the routing packets (see Footnote 3) and performs route computation. Routing packets are not generated by a `RoutingUnit` but by one or more `RoutingPacketGenerators` attached to the `NetworkLayerUnit`. Computed routes are stored in the `RIB` of the `NetworkLayerUnit`. The `RoutingUnit` is responsible for populating the `FIB` of the node with stable routes and removing routes when they become stale or unavailable.

Note that, contrary to data packets, routing packets at intermediate nodes do not pass through the NetworkLayerManager. Hence, IP fields such that the TTL must be updated by the RoutingUnit.

4.2.3 RoutingPacketGenerator

Files: routing-packet-generator.{cc,h}

A RoutingPacketGenerator is attached to a NetworkLayerUnit and is used to generate routing (control) packets (in reality, packets of type equal to the type of packets associated with the NetworkLayerUnit, i.e. packets that will be handled internally by the RoutingUnit). The NetworkLayerUnit keeps a list of all the generators attached to it.

RoutingPacketGenerator is a generic class that extends TimerHandler. Subclasses should implement the expire method and packets should be generated when the timer expires. Reactive packet generation can therefore be obtained by rescheduling the timer to expire immediately.

Commands at a glance

Following is a list of commands used in simulation scripts:

To start/stop the timer:
`$gen start/stop`

4.3 NetworkInformationBase

Files: network-information-base.{cc,h}

A NetworkInformationBase is a generic structure for storing routing/forwarding information at the network layer. A NetworkInformationBaseEntry contains information about a route to a destination. In a *multi-** context [2], there may be multiple routes to a destination, using different interfaces, different channels and different next-hops. Hence, there may be more than one NetworkInformationBaseEntry for a single destination.

For a given destination, there is a unique NetworkInformation BaseEntry for a tuple <interface, channel, next-hop, source, type> (DestinationIBKey). Wildcards are ANY_IFACE.value(), ANY_ADDRESS, ANY_CHANNEL, ANY_ADDRESS, ANY_TYPE, respectively. They can be used to look up all the entries that match a subset of criteria; for instance all the entries using a given interface regardless the channel, the next-hop, the source and the type.

A base NetworkInformationBaseEntry includes:

- destination: for instance a node/sub-net/domain address or “default”.
- interface: the output interface to use (required if the NetworkInformationBaseEntry is not a route to a gateway).
- channel: the channel to use if it applies, UNKN_CHANNEL otherwise.
- next-hop: the next-hop towards the destination. If no next-hop is provided and the next-hop cannot be guessed, the destination address is used (it must then be a node address). If the NetworkInformationBaseEntry is a route to a gateway, next-hop must be the address of the gateway.
- source: the source of the packets that may use this route; ANY_ADDRESS should be used for source-independent routes.

- type: the type of the route (if several types of routes are to coexist); UNKN_TYPE should be used for routes with unspecified type.
- metric: measure of the quality of the route, e.g. the cost to the destination in number of hops.
- preference: the preference value [0] may be used to decide which route to use when more than one are available.

4.3.1 FIB

If a NetworkLayerManager is created, a forwarding information base (FIB) is also created and associated to the Node. It is a NetworkInformationBase in which routes can be added using OTcl add and delete commands, see below. These commands are used indirectly when using Node::add-route and Node::delete-route (OTcl), respectively; either manually or by detailed dynamic routing protocols implemented in OTcl (for instance DV and LS).

A FIB contains routes computed by any routing protocol either global[4] (for instance Static or Session) or running on the node[5], or manually added to the FIB, see Section 4.3.3.

Commands at a glance

Following is a list of commands used in simulation scripts:

For debugging purposes, to print out the FIB of a node on the standard output (Unix route command-like format):
$node dump-fib

Following is a list of internal commands:

To add an entry to the FIB:
$fib add <dest> <iface> <next-hop> <channel> <source> <type> <gateway> <metric> <pref>
called by Node::add-route (OTcl). The types for the parameters are given below:
dest: char*
iface: NetworkInterface2* or -1 (UNKN_IFACE.value())
next-hop: Node or -2 (UNKN_ADDRESS)
channel: Channel* or -1 (UNKN_CHANNEL)
source: Node or -3 (ANY_ADDRESS)
type: int8_t $\geqslant 0$ or -1 (UNKN_TYPE)
gateway: true/false
metric: double $\geqslant 0$ or -1(UNKN_METRIC) pref: int

To delete an entry from the FIB:
$fib delete <destination> <interface> <next-hop> <channel> <source> <type>
called by Node::delete-route (OTcl) The types for the parameters are given below:
destination: char* (wildcard: "any")
interface: NetworkInterface2* or -2 (ANY_IFACE.value())
next-hop: Node or -3 (ANY_ADDRESS)
channel: Channel* or -2 (ANY_CHANNEL)
source: Node or -3 (ANY_ADDRESS)
type: int8_t $\geqslant 0$ or -2 (ANY_TYPE)

[4] in this case routes are added to/deleted from the FIB in Node::add_/delete_route (C++).
[5] in this case, routes are added to/deleted from the FIB in Node::add-/delete-route (OTcl) or directly by any NetworkLayerUnit, more precisely RoutingUnit.

4.3.2 RIB

A RIB is a specialized NetworkInformationBase that may be used by a NetworkLayerUnit to store routing information internally.

4.3.3 Support for existing (as in ns-2.34) routing/forwarding protocols

AddressClassifier

Figure 1.1 shows how the AddressClassifier, and by extension any legacy chain of Classifiers, is integrated in the new layout, thus enabling using existing protocols for wired networks without any change. However, a base NetworkLayerUnit with only a base ForwardingUnit can be used instead as well.

Protocols for wired networks

Routes computed by centralized routing protocols such as Static or Session are added to/deleted from the FIB in Node::add_route/delete_route. Routes computed by detailed dynamic protocols implemented in OTcl such that DV or LS are added to/deleted from the FIB in Node::add-route/delete-route.

Note that these protocols compute routes between any nodes connected by Links regardless which NetworkLayerUnit the PointToPointInterfaces may be attached to.

Note also that if several protocols are used, only a single route per protocol to a given destination is added to the FIB. Multiple routes are filtered out by the rtObject in OTcl.

Protocols for wireless networks

Routing protocols for wireless networks, e.g. AODV or DSR, are implemented as routing Agents and do not used the Node object but the MobileNode or a tailored version of it depending on the protocol. Such routing agents are not supported directly and need to be adapted (at least wrapped into a NetworkLayerUnit).

Manual routing

Manual routing, that is the manual configuration of the FIB in the simulation script, is done using the extended OTcl commands add-route and delete-route[6]. The usage of these commands is as follows:
$node add-/delete-route <destination> -<option> <value>

Available options are given in Tables 4.2 and 4.3, respectively.

Note To prevent routes from being added by default rtprotos Static and Direct, one may use Agent/rtProto/None:
<Simulator instance> rtproto None

[6]Backward compatibility with the original commands $ node add-/delete-route <destination> <target> is provided.

command	description
iface	`NetworkInterface2` or -1 (unspecified) [-1]
next-hop	`Node` or -2 (unspecified) [-2]
channel	`Channel` or -1 (unspecified) [-1]
src	`Node` or -3 (any) [-3]
type	route type or -1 (unspecified) [-1]
gw	next-hop is a gateway true or false, or gateway `Node` (overriding next-hop) [false]
metric	route metric or -1 (unspecified) [-1]
preference	route preference [0]

Table 4.2: Available options for `add-route`

command	description
iface	`NetworkInterface2` or -2 (any) [-2]
next-hop	`Node` or -3 (any) [-3]
channel	`Channel` or -2 (any) [-2]
src	`Node` or -3 (any) [-3]
type	route type or -2 (any) [-2]

Table 4.3: Available options for `delete-route`

4.4 NeighbourInformationBase

Files: neighbour-information-base.{cc,h}

A `NeighbourInformationBase` is a structure to store information about the neighbourhood of a node. A `NeighbourInformationBaseEntry` contains information about a neighbour: its address, the interface on which it can be reached and the channel to use to reach it, if applicable. These three elements are used as key (`NeighbourKey`) to distinguish between entries. Wildcards can be used to look up all the entries that match a subset of criteria; for instance all the entries to a given neighbour address regardless the interface and the channel used.

Chapter 5

Tracing and logging

Directory: q2s_trace

5.1 Tracer

Files: tracer.{cc,h}

Tracer objects are used to trace packets. A Tracer is associated to a NetworkInterface2 or a Node (a CMUTrace is associated to a source address[1] and a MobileNode) and is not attached to a specific destination (contrary to Trace). Tracers for different layers (Phy, Mac, Ifq, Net and E2E[2]) and types (Send, Recv, Drop) are implemented as different C++ objects (for CMUTrace there is only one C++ object, but several OTcl objects).

Tracer are attached to a trace file and/or a NAM trace file. Contrary to CMU traces, only Mac level events and Drop events are used for NAM tracing. With CMU traces, the same packet may be traced at several levels resulting for example in several broadcast circles for a single packet on the animation. In addition, Tracers are added to the global list of traces (alltrace_, see tcl/lib/ns-lib.tcl) so that the flush-trace command actually flushes the trace files[3]. Note that those files may also contain other information. For instance, the standard trace file is also used for mobility and energy tracing and the NAM trace file also contains extra information about the initial state of the nodes.

Finally, packet stamping, e.g. by a God object, is decoupled from tracing[4]. To enable stamping of packets dedicated objects are needed, for instance to tag packets with the optimal number of hops (God information) a GodStamp must be installed on the Node, see Section 5.1.4. Note that God must be modified to allow "active" God to be used.

5.1.1 Node layout

Tracers are inserted at the interface between layers, see q2s_tcl/interface.tcl and tcl/lib/ns-node.tcl. Figure 5.2 shows where Net and E2ETracers are inserted in the new Node layout. For comparison, Figure 5.1 recalls the positions of the corresponding CMUTrace objects (RTR and AGT) in the MobileNode. RtAgent denotes a routing Agent, for instance Agent/AODV. The

[1] Note that in tcl/lib/ns-mobilenode.tcl the id of the node is used to initialize src_ which is incorrect when hierarchical addresses are used.

[2] end-to-end; corresponds to traces at the agent level.

[3] $ns flush-trace is used in most (if not all) of the example scenarios using MobileNodes but does actually nothing since CMUtraces are not added to the global list of traces (see tcl/lib/ns-mobilenode.tcl).

[4] In CMUTrace, stamping is triggered only if AGT tracing is enabled.

positions of Tracers between lower layers, e.g. Mac, are unchanged.

Note the differences with respect to CMUTraces:

- Send/Recv for Tracers corresponds to the direction of the packet in the Node. This is not the case for CMUTrace/RTR where data packets generated at the MobileNode are "received" by the RtAgent.
- With the MobileNode, a data packet generated locally is traced twice at the RTR level; it is first received and then sent. On the other hand, a data packet that reaches its destination is not traced at the RTR level. With the new Node layout, a data packet generated locally is traced only once at its source at the NET level (Send). At the destination, the data packet is also traced once at the the NET level (Recv).

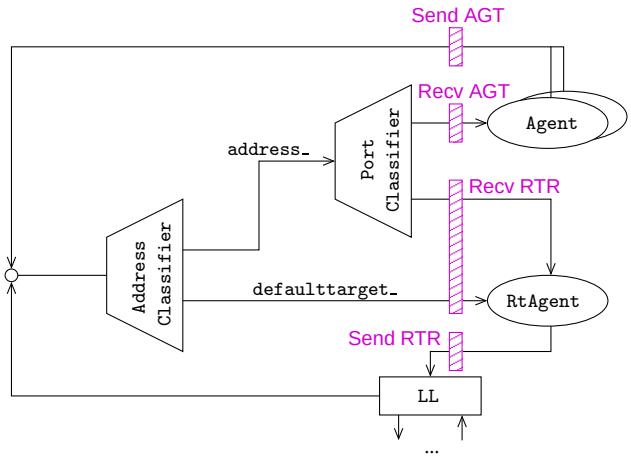

Figure 5.1:

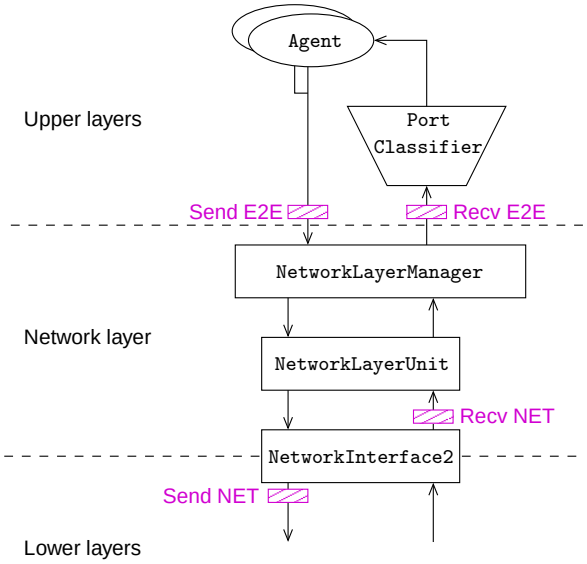

Figure 5.2:

5.1.2 Trace format

Tracer only provides a newtrace-like format, see Chapter 16 in [4]. Headers are printed only when relevant; for instance, IP header information does not apply for MAC layer control packets and, hop-by-hop information is not relevant at the E2E level.

In the list below, the corresponding field of the newtrace format is given in italic. Note that the order of the fields differs from the order of the fields of the newtrace format.

Event fields

event type

s (*s*) send event
r (*r*) receive event
d (*d*) drop event
f (*f*) forward event

-t (-*t*) time

-l (-*Nl*) layer (tracing level)

-w (-*Nw*) drop reason, see trace/drop.h ('---' for other types of events)

Node fields

-Na (-*Ni/-Hs*) local node address

-Ni local interface identifier (below Net level only)

If position:

-Nx (-*Nx*) x coordinate [m]

-Ny (-*Ny*) y coordinate [m]

-Nz (-*Nz*) z coordinate [m]

If energy:

-Ne (-*Ne*) energy level

Common header (metadata) fields

Channel information (if available):

-C channel index

Hop information:

-Hs sender address

-Hd (-*Hd*) destination address (* if broadcast)

Packet information:

-Pt (-*It*) packet type

-Pl (-*Il*) packet length

-Pi (-*Ii*) packet uid

-Pf (-*Pf*) hop count

-Po (-*Po*) optimal total number of hops (God)

Mac header fields (only for Mac level traces)

Example of the MAC 802.11 header:

-Mra (-*Md*) receiver MAC address

-Mta (-*Ms*) transmitter MAC address

-Mt (-*Mt*) frame type (hexadecimal)

-Mf frame type (human-readable)

-Md (-*Ma*) duration

IP header fields

-Is (-*Is, before last .*) source address

-Id (-*Id, before last .*) destination address

-If (-*If*) flow id

-It (-*Iv*) TTL

Layer 4 ports (information included in the IP header in *ns-2*)

-Ts (-*Is, after last .*) source port

-Td (-*Id, after last* .) destination port

Transport header remaining fields

Example of the TCP header:

-Tsn (-*Ps*) sequence number

-Tan (-*Pa*) acknowledgement number

5.1.3 Support for dynamic libraries

Application layer printers `PacketPrinter` (similar to `PacketTracer` in cmu-trace.cc/h) and MAC header printers can be added dynamically to support new packet types and MAC layers developed in dynamic libraries.

To add a new `PacketPrinter`:
`Tracer::addPacketPrinter(<packet_t>,<PacketPrinter*>);`

To add a new `MacHeaderPrinter`:
`MacTracer::addMacHeaderPrinter(<type()>,<MacHeaderPrinter*>);`

5.1.4 Configuration

Tracing at Phy, Mac, Ifq and Net levels is enabled on a per-interface basis using the corresponding options, see Table 3.1. E2E-tracing is enabled on a per-node basis using the `e2e-trace` option of the `node-config` command.

option	description
e2e-trace	E2E tracing `on/off [off]`

Table 5.1: Available option for `node-config` for enabling/disabling end-to-end tracing

Finally, as mentioned in the introduction to this section, tagging of packets is not triggered by `Tracer` ojects. To install a `GodStamp` object on a `Node`:
`$node_($i) install-agent-target [new GodStamp]`[5]

5.2 Logger

Files: logger.{cc,h}

A `Logger` is a generic object that can be used to log any type of information to a file.

For instance, a `MobilityLogger`[6] can be used to log the mobility of one or more `Nodes`.

[5]The new `install-agent-target` command of `Node` installs the (`BiConnector`) object passed as an argument between the `Agents` and the `NetworkLayerManager`. Used several times, it creates a chain of objects between the `Agents` and the `NetworkLayerManager`, e.g. a `GodStamp` and a `Tracer/E2E/Send`.

[6]defined in q2s_modules/mobility-module.h.

Chapter 6

Setting up nodes and networks

6.1 Creating a node

A Node is configured using the node-config command. New options are given in Table 4.1.

6.1.1 Adding a network interface to a node

NetworkInterface2 are created and added to a Node using the add-interface command. The syntax is the following:

```
$node add-interface <NetworkInterface2/[FullStack/] subclass name> [<NetworkLayerUnit>]
```

For instance 'PointToPoint' is to be used to create a PointToPointInterface (NetworkInterface2/PointToPoint), 'Wireless' is to be used to create a WirelessInterface (NetworkInterface2/FullStack/Wireless). The Network LayerUnit object is optional. If provided, the created interface is attached upwards to it.

If there is a NetworkLayerManager on the Node, the NetworkLayerManager is attached downwards to the NetworkLayer Unit the interface is attached to, if any. If no NetworkLayerUnit is provided, the NetworkLayerManager is attached downwards to the node entry point. See Figure 1.1.

If no NetworkLayerManager is present on the Node, the interface is attached upwards to the node entry point.

6.2 Setting up a network

6.2.1 Connecting two nodes with a point-to-point link

When there is no NetworkLayerManager on a Node, the legacy behaviour is maintained. Links are created and attached to Nodes using the duplex/simplex-link commands.

When there is a NetworkLayerManager on a Node a Link is to be connected to but no NetworkLayerUnit is used, a Link is created and attached to the Node using the duplex/simplex-link command, as previously. However, the duplex/simplex-link command creates and attaches a PointToPointInterface to the Link.

Now, to connect a PointToPointInterface attached to a NetworkLayerUnit to a Link, the new add-duplex/simplex-link command is used. A PointToPointInterface is required on each of the Nodes to be connected with the Link. The syntax of add-duplex/simplex-link is similar to that of duplex/simplex-link, but nodes are replaced by interfaces. add-duplex/simplex-link creates a Link and attaches it to the interfaces. The code below illustrates this; two nodes are created and connected with a point-to-point duplex link and a NetworkLayerUnit is used on each of the Nodes.

```
set ns [new Simulator]

$ns node-config -network-layer-manager on

for {set i 0} {$i < 2} {incr i} {
    # Create a node
    set n($i) [$ns node]
    # Create a network layer unit for forwarding over the point-to-point link
    set nl($i) [new NetworkLayerUnit $n($i)]
}

# Create the point to point interfaces
set iface(0,1) [$n(0) add-interface PointToPoint $nl(0)]
set iface(1,0) [$n(1) add-interface PointToPoint $nl(1)]

# Create a duplex link between the nodes
$ns add-duplex-link $iface(0,1) $iface(1,0) 1Mb 10ms DropTail
```

Chapter 7

GNU Free Documentation License

Version 1.2, November 2002

51 Franklin St, Fifth Floor, Boston, MA 02110-1301 USA

Preamble

The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

1. APPLICABILITY AND DEFINITIONS

This License applies to any manual or other work, in any medium, that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. Such a notice grants a world-wide, royalty-free license, unlimited in duration, to use that work under the conditions stated herein. The "**Document**", below, refers to any such manual or work. Any member of the public is a license, and is addressed as "**you**". You accept the license if you copy, modify or distribute the work in a way requiring permission under copyright law.

A "**Modified Version**" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "**Secondary Section**" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (Thus, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "**Invariant Sections**" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License. If a section does not fit the above definition of Secondary then it is not allowed to be designated as Invariant. The Document may contain zero Invariant Sections. If the Document does not identify any Invariant Sections then there are none.

The "**Cover Texts**" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License. A Front-Cover Text may be at most 5 words, and a Back-Cover Text may be at most 25 words.

A "**Transparent**" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, that is suitable for revising the document straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup, or absence of markup, has been arranged to thwart or discourage subsequent modification by readers is not Transparent. An image format is not Transparent if used for any substantial amount of text. A copy that is not "Transparent" is called "**Opaque**".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML, PostScript or PDF designed for human modification. Examples of transparent image formats include PNG, XCF and JPG. Opaque formats include proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML, PostScript or PDF produced by some word processors for output purposes only.

The "**Title Page**" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

A section "**Entitled XYZ**" means a named subunit of the Document whose title either is precisely XYZ or contains XYZ in parentheses following text that translates XYZ in another language. (Here XYZ stands for a specific section name mentioned below, such as "**Acknowledgements**", "**Dedications**", "**Endorsements**", or "**History**".) To "**Preserve the Title**" of such a section when you modify the Document means that it remains a section "Entitled XYZ" according to this definition.

The Document may include Warranty Disclaimers next to the notice which states that this License applies to the Document. These Warranty Disclaimers are considered to be included by reference in this License, but only as regards disclaiming warranties: any other implication that these Warranty Disclaimers may have is void and has no effect on the meaning of this License.

2. VERBATIM COPYING

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

3. COPYING IN QUANTITY

If you publish printed copies (or copies in media that commonly have printed covers) of the Document, numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a computer-network location from which the general network-using public has access to download using public-standard network protocols a complete Transparent copy of the Document, free of added material. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

4. MODIFICATIONS

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

A. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.

B. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has fewer than five), unless they release you from this requirement.

C. State on the Title page the name of the publisher of the Modified Version, as the publisher.

D. Preserve all the copyright notices of the Document.

E. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.

F. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.

G. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.

H. Include an unaltered copy of this License.

I. Preserve the section Entitled “History”, Preserve its Title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section Entitled “History” in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.

J. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the “History” section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.

K. For any section Entitled “Acknowledgements” or “Dedications”, Preserve the Title of the section, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.

L. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.

M. Delete any section Entitled “Endorsements”. Such a section may not be included in the Modified Version.

N. Do not retitle any existing section to be Entitled “Endorsements” or to conflict in title with any Invariant Section.

O. Preserve any Warranty Disclaimers.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version’s license notice. These titles must be distinct from any other section titles.

You may add a section Entitled “Endorsements”, provided it contains nothing but endorsements of your Modified Version by various parties–for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

5. COMBINING DOCUMENTS

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice, and that you preserve all their Warranty Disclaimers.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections Entitled “History” in the various original documents, forming one section Entitled “History”; likewise combine any sections Entitled “Acknowledgements”, and any sections Entitled “Dedications”. You must delete all sections Entitled “Endorsements”.

6. COLLECTIONS OF DOCUMENTS

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

7. AGGREGATION WITH INDEPENDENT WORKS

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, is called an “aggregate” if the copyright resulting from the compilation is not used to limit the legal rights of the compilation’s users beyond what the individual works permit. When the Document is included in an aggregate, this License does not apply to the other works in the aggregate which are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one half of the entire aggregate, the Document’s Cover Texts may be placed on covers that bracket the Document within the aggregate, or the electronic equivalent of covers if the Document is in electronic form. Otherwise they must appear on printed covers that bracket the whole aggregate.

8. TRANSLATION

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License, and all the license notices in the Document, and any Warranty Disclaimers, provided that you also include the original English version of this License and the original versions of those notices and disclaimers. In case of a disagreement between the translation and the original version of this License or a notice or disclaimer, the original version will prevail.

If a section in the Document is Entitled “Acknowledgements”, “Dedications”, or “History”, the requirement (section 4) to Preserve its Title (section 1) will typically require changing the actual title.

9. TERMINATION

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

10. FUTURE REVISIONS OF THIS LICENSE

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See http://www.gnu.org/copyleft/.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

Bibliography

[1] CMU Monarch project, Computer Science Department, Canergie Mellon University, Pittsburgh. “The CMU Monarch project’s wireless and mobility extensions to ns”, August 1999.

[2] Laurent Paquereau and Bjarne E. Helvik. Simulation of Wireless Multi-* Networks in NS-2. In *Proceedings of the Second Workshop on ns-2: The IP network simulator (WNS2)*.

[3] Laurent Paquereau and Bjarne E. Helvik. A Module-based Wireless Node for NS-2. In *Proceedings of the First Workshop on ns-2: The IP network simulator (WNS2)*, Pisa, Italy, October 2006.

[4] The VINT Project, UC Berkeley, LBL, USC/ISI, and Xerox PARC. “The *ns* Manual”, Kevin Fall and Kannan Varadhan edition, 2008.