CCN Lite – OMNet++

Computer Networks group

Contents

- Conceptual description of OMNet++ integration of CCN Lite
- UML class diagram of OMNet++ components
- Files in ccnlite/src/
  - code base that implements the CCN Lite – OMNet++ integration
- Eclipse bundle: what is where
- Configuring CCN experiments
- Adding your own code

Conceptual Diagram

Obvious Dependencies:

- OMNet++ (> v4.2.2) simulator
- INET framework (> v1.99.4.)
- Optionally MiXiM framework (> v2.2.1)
- ... everything else we provide ☺

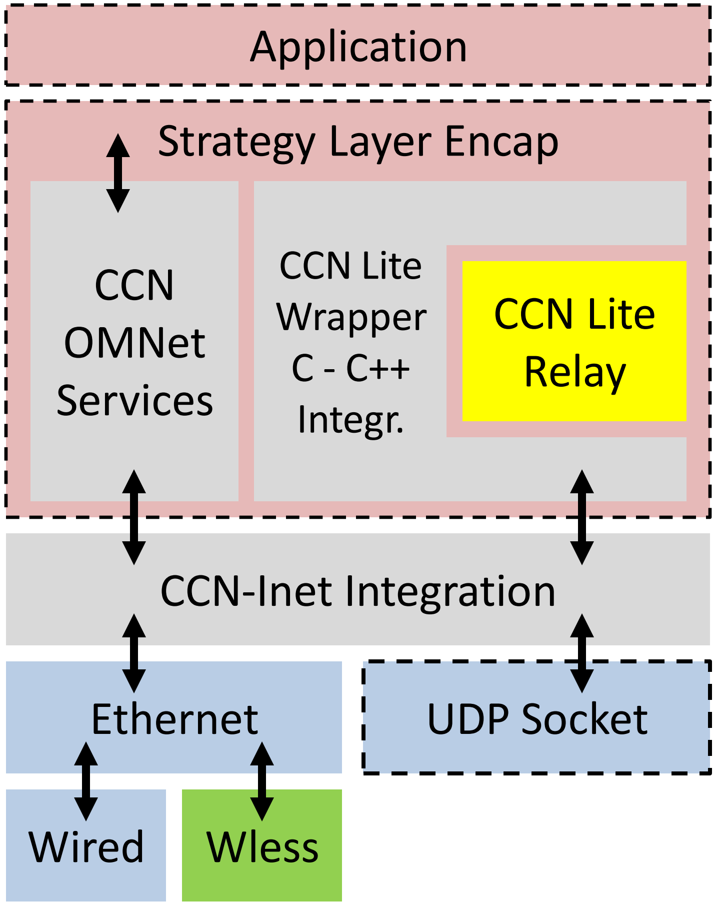
Not currently provided in suite

OMNet++ (CCNLite)

To be Developed by user

OMNet++ (INET)

OMNet++ (MiXiM)



Application

Strategy Layer Encap

CCN OMNet Services

CCN Lite Wrapper C - C++ Integr.

CCN Lite Relay

CCN-Inet Integration

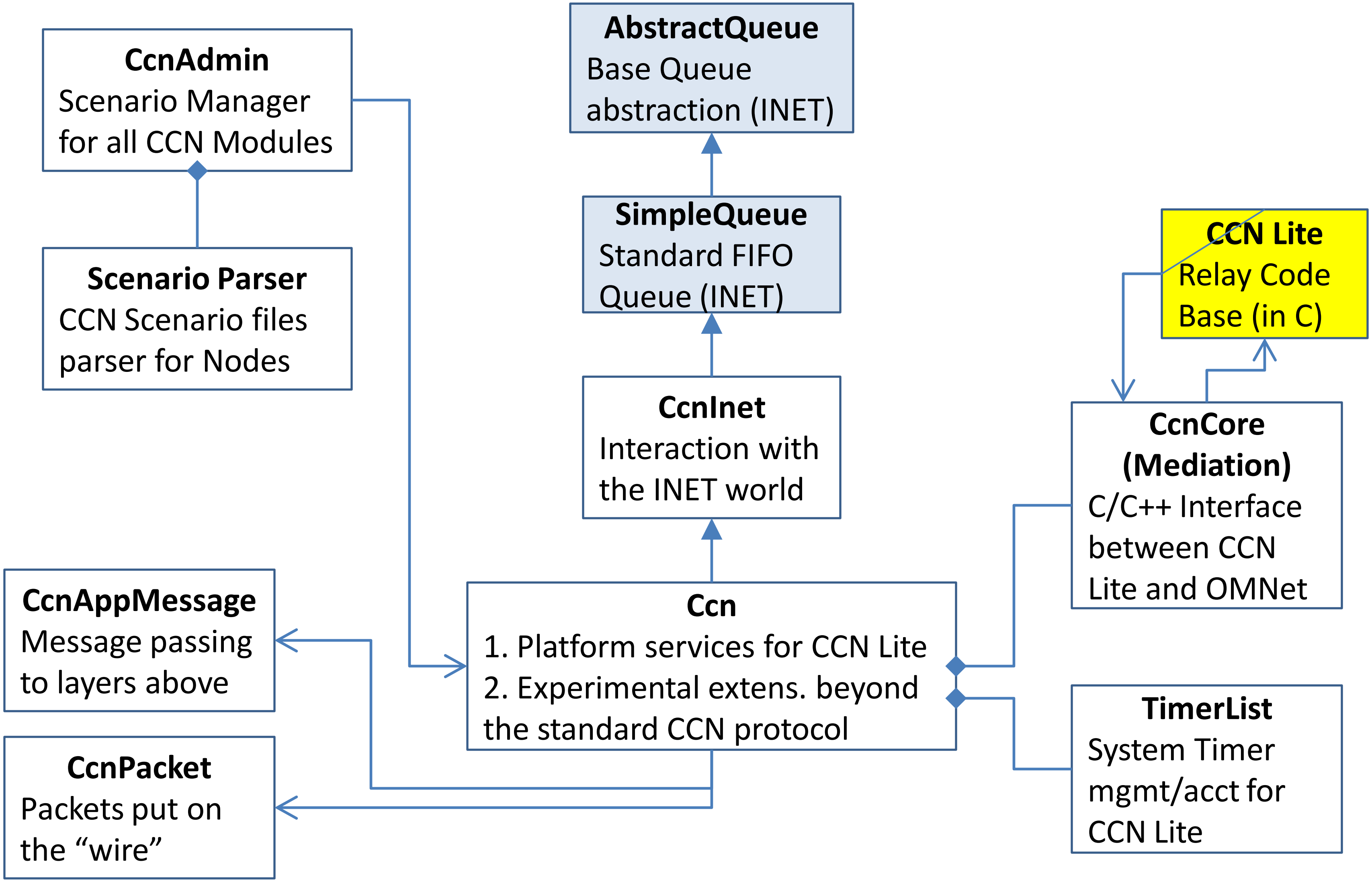
Ethernet

UDP Socket

Wired

Wless

UML Class Structure



CcnAdmin
Scenario Manager for all CCN Modules

Scenario Parser
CCN Scenario files parser for Nodes

AbstractQueue
Base Queue abstraction (INET)

SimpleQueue
Standard FIFO Queue (INET)

CcnInet
Interaction with the INET world

CCN Lite
Relay Code Base (in C)

CcnCore (Mediation)
C/C++ Interface between CCN Lite and OMNet

CcnAppMessage
Message passing to layers above

Ccn
1. Platform services for CCN Lite
2. Experimental extens. beyond the standard CCN protocol

TimerList
System Timer mgmt/acct for CCN Lite

CcnPacket
Packets put on the “wire”

Files (components)

- **ccnl/** - Actual CCN Lite Relay implementation
- **CcnCore.{cc,h}** - CCN Lite Integration (C - C++)
- **Ccn.{cc,h,ned}** - OMNet++ services
- **CcnInet.{cc,h,ned}** - OMNet++ INET Framework Integration
- **CcnAdmin.{cc,h,ned}** - Scenario Administrator (God)
- **Parser.{cc,h}** – Scenario parser (By Thomas Meyer)
- **CcnPacket_m.{cc,h}** – Extensible container for CCN packets exchanged via CCN nodes in OMNet++
- **CcnAppMessage_m.{cc/h/msg}** – Message passing that serves the communication between the CCN layer and layers above

Eclipse project bundle description

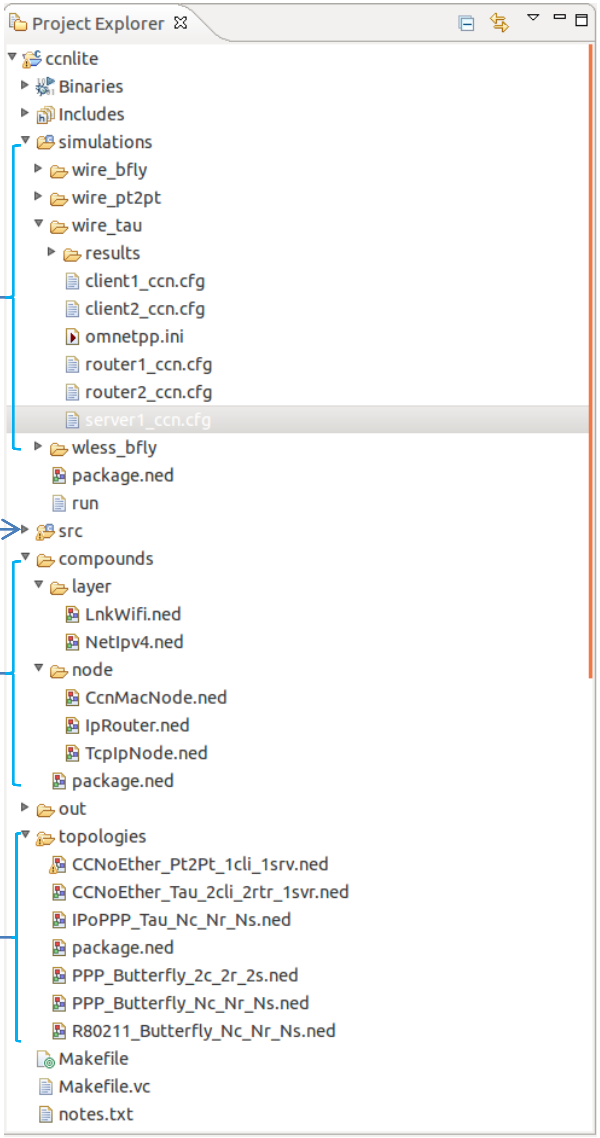
What is Where

Simulation tests for different set-ups and topologies (INI files as well as CCN node scenario configuration files are here)

Source code and NED definitions of *simple modules* and message types

NED configuration files for *compound modules*
- node definitions
- layer definitions

NED configuration files for different topologies



```
ccnlite
  Binaries
  Includes
  simulations
    wire_bfly
    wire_pt2pt
    wire_tau
      results
      client1_ccn.cfg
      client2_ccn.cfg
      omnetpp.ini
      router1_ccn.cfg
      router2_ccn.cfg
      server1_ccn.cfg
    wless_bfly
    package.ned
    run
  src
  compounds
    layer
      LnkWifi.ned
      NetIpv4.ned
    node
      CcnMacNode.ned
      IpRouter.ned
      TcpIpNode.ned
    package.ned
  out
  topologies
    CCNoEther_Pt2Pt_1cli_1srv.ned
    CCNoEther_Tau_2cli_2rtr_1svr.ned
    IPoPPP_Tau_Nc_Nr_Ns.ned
    package.ned
    PPP_Butterfly_2c_2r_2s.ned
    PPP_Butterfly_Nc_Nr_Ns.ned
    R80211_Butterfly_Nc_Nr_Ns.ned
  Makefile
  Makefile.vc
  notes.txt
```

Experiment Configuration

- OMNet++ .INI files
  - Experiment parameters
  - Module options
- OMNet++ NED files
  - Topology
  - Node component composition
- Node configuration files
  - CCN scenario

OMNet++ .INI files

Which topology to use

Console debug output level
.. Can be adjusted per node

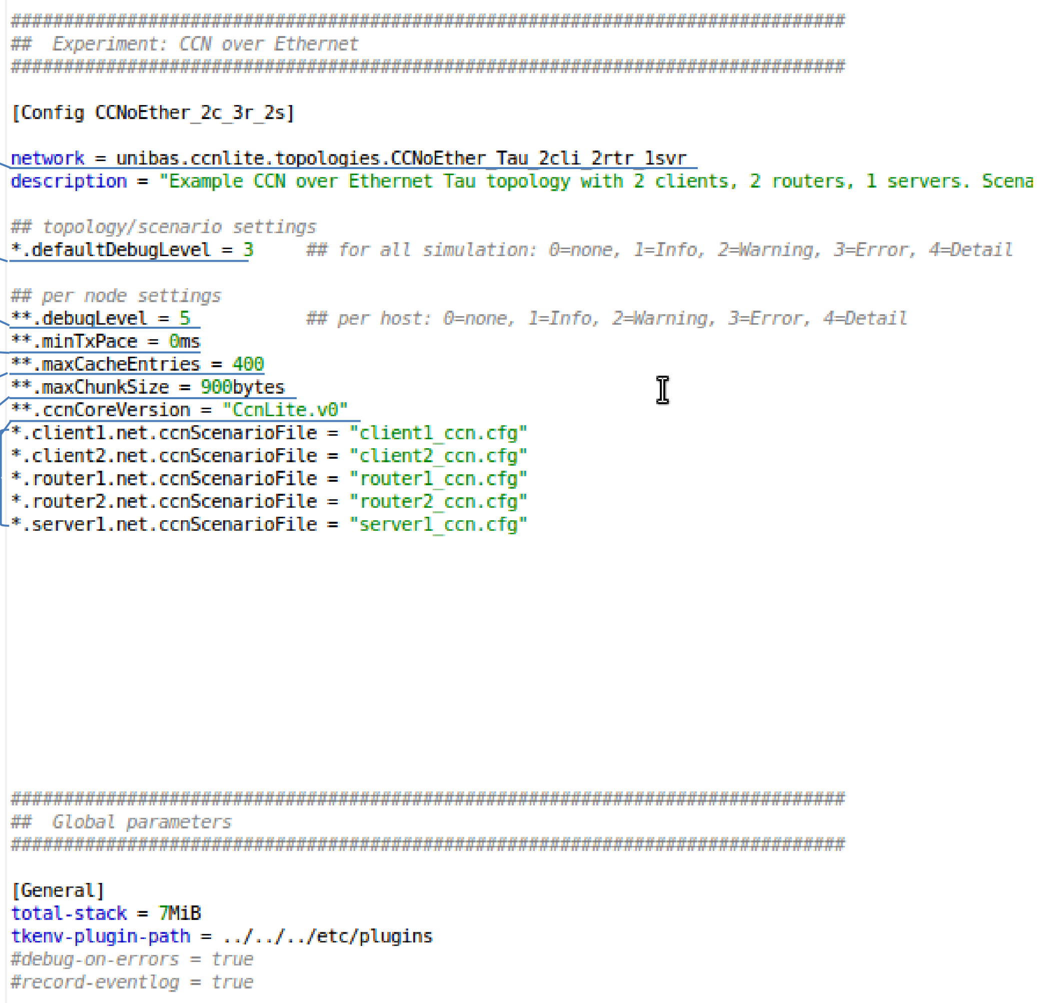
Pace transmissions of CCN pkts
(at each CCN relay)

Cache storage
(for each node)

Size of named content chunks
(used with dummy traffic)

Version of CCN Lite code to use
(for each node)

CCN Scenario files that setup
the behavior of each node
throughout the experiment



```
################################################################################
##  Experiment: CCN over Ethernet
################################################################################

[Config CCNoEther_2c_3r_2s]

network = unibas.ccnlite.topologies.CCNoEther_Tau_2cli_2rtr_1svr
description = "Example CCN over Ethernet Tau topology with 2 clients, 2 routers, 1 servers. Scena

## topology/scenario settings
*.defaultDebugLevel = 3     ## for all simulation: 0=none, 1=Info, 2=Warning, 3=Error, 4=Detail

## per node settings
**.debugLevel = 5           ## per host: 0=none, 1=Info, 2=Warning, 3=Error, 4=Detail
**.minTxPace = 0ms
**.maxCacheEntries = 400
**.maxChunkSize = 900bytes
**.ccnCoreVersion = "CcnLite.v0"
*.client1.net.ccnScenarioFile = "client1_ccn.cfg"
*.client2.net.ccnScenarioFile = "client2_ccn.cfg"
*.router1.net.ccnScenarioFile = "router1_ccn.cfg"
*.router2.net.ccnScenarioFile = "router2_ccn.cfg"
*.server1.net.ccnScenarioFile = "server1_ccn.cfg"









################################################################################
##  Global parameters
################################################################################

[General]
total-stack = 7MiB
tkenv-plugin-path = ../../../etc/plugins
#debug-on-errors = true
#record-eventlog = true
```

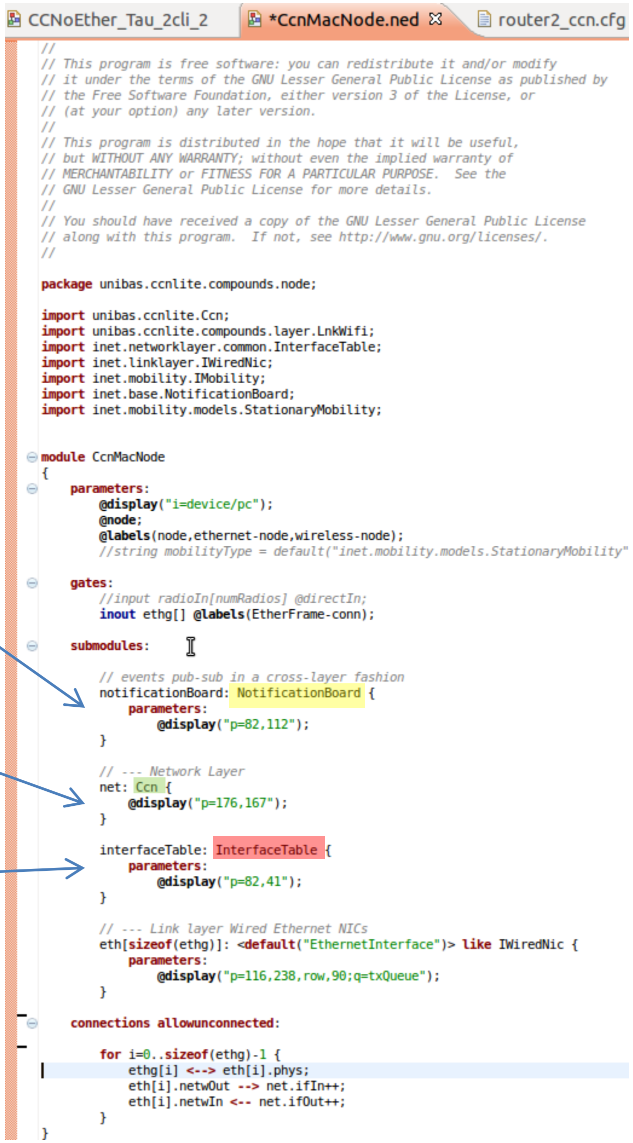
OMNet++ NED files: CCN Node

Eg. Configuration of CCN node over Ethernet

The notification board is useful for pub/sub of events (intended to use in the future)

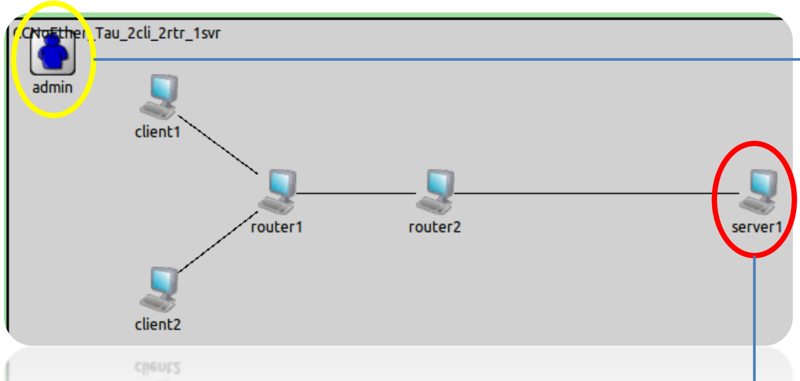
CCN layer functionality is provided by the Ccn module

As in the IPv4/6 network layer definition we reuse the Interface Table module (for the association of MAC addresses to NICs)



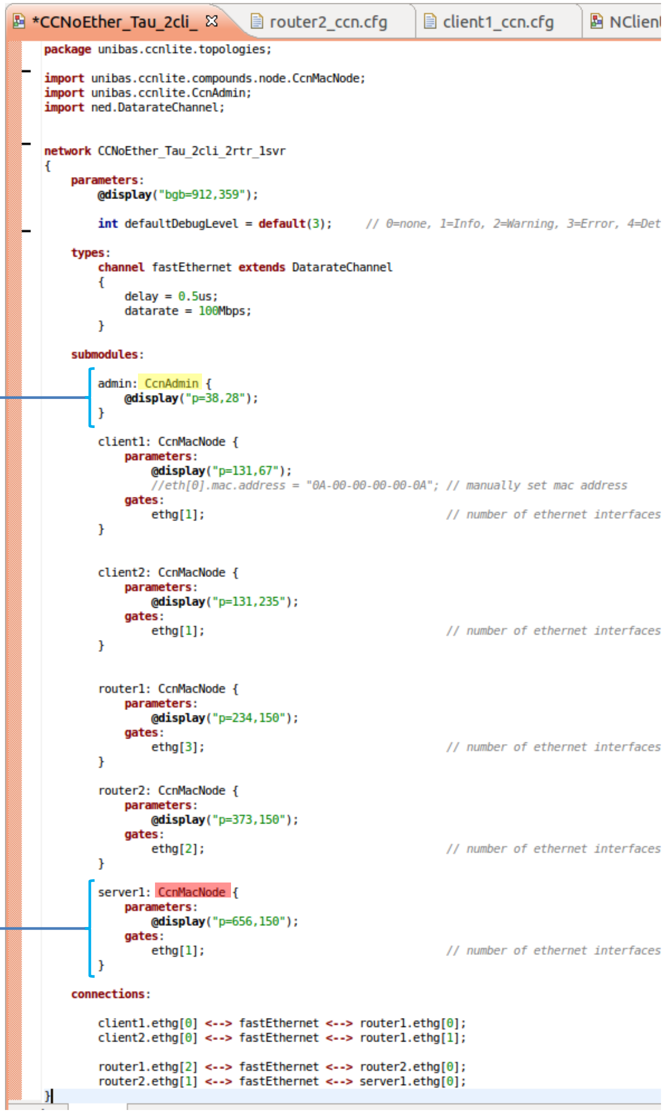
```
//
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU Lesser General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU Lesser General Public License for more details.
//
// You should have received a copy of the GNU Lesser General Public License
// along with this program.  If not, see http://www.gnu.org/licenses/.
//

package unibas.ccnlite.compounds.node;

import unibas.ccnlite.Ccn;
import unibas.ccnlite.compounds.layer.LnkWifi;
import inet.networklayer.common.InterfaceTable;
import inet.linklayer.IWiredNic;
import inet.mobility.IMobility;
import inet.base.NotificationBoard;
import inet.mobility.models.StationaryMobility;


module CcnMacNode
{
    parameters:
        @display("i=device/pc");
        @node;
        @labels(node,ethernet-node,wireless-node);
        //string mobilityType = default("inet.mobility.models.StationaryMobility"

    gates:
        //input radioIn[numRadios] @directIn;
        inout ethg[] @labels(EtherFrame-conn);

    submodules:

        // events pub-sub in a cross-layer fashion
        notificationBoard: NotificationBoard {
            parameters:
                @display("p=82,112");
        }

        // --- Network Layer
        net: Ccn {
            @display("p=176,167");
        }

        interfaceTable: InterfaceTable {
            parameters:
                @display("p=82,41");
        }

        // --- Link layer Wired Ethernet NICs
        eth[sizeof(ethg)]: <default("EthernetInterface")> like IWiredNic {
            parameters:
                @display("p=116,238,row,90;q=txQueue");
        }

    connections allowunconnected:

        for i=0..sizeof(ethg)-1 {
            ethg[i] <--> eth[i].phys;
            eth[i].netwOut --> net.ifIn++;
            eth[i].netwIn <-- net.ifOut++;
        }
}
```

OMNet++ .NED files: Topologies



Every topology needs to have a CcnAdmin module in addition to the CCN nodes.

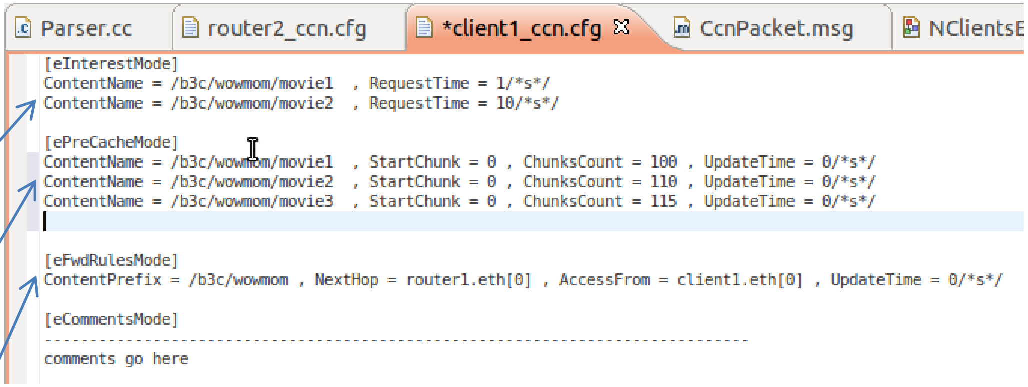
It also helps to set in advance the number of interfaces per node and explicitly define the connections (rather than use auto-vectors), since this will allow you to remember the interfaces when defining the forwarding rules in the scenario files (see next slide)



```
package unibas.ccnlite.topologies;

import unibas.ccnlite.compounds.node.CcnMacNode;
import unibas.ccnlite.CcnAdmin;
import ned.DatarateChannel;


network CCNoEther_Tau_2cli_2rtr_1svr
{
    parameters:
        @display("bgb=912,359");

        int defaultDebugLevel = default(3);     // 0=none, 1=Info, 2=Warning, 3=Error, 4=Det

    types:
        channel fastEthernet extends DatarateChannel
        {
            delay = 0.5us;
            datarate = 100Mbps;
        }

    submodules:

        admin: CcnAdmin {
            @display("p=38,28");
        }

        client1: CcnMacNode {
            parameters:
                @display("p=131,67");
                //eth[0].mac.address = "0A-00-00-00-00-0A"; // manually set mac address
            gates:
                ethg[1];                                     // number of ethernet interfaces
        }


        client2: CcnMacNode {
            parameters:
                @display("p=131,235");
            gates:
                ethg[1];                                     // number of ethernet interfaces
        }


        router1: CcnMacNode {
            parameters:
                @display("p=234,150");
            gates:
                ethg[3];                                     // number of ethernet interfaces
        }

        router2: CcnMacNode {
            parameters:
                @display("p=373,150");
            gates:
                ethg[2];                                     // number of ethernet interfaces
        }

        server1: CcnMacNode {
            parameters:
                @display("p=656,150");
            gates:
                ethg[1];                                     // number of ethernet interfaces
        }

    connections:

        client1.ethg[0] <--> fastEthernet <--> router1.ethg[0];
        client2.ethg[0] <--> fastEthernet <--> router1.ethg[1];

        router1.ethg[2] <--> fastEthernet <--> router2.ethg[0];
        router2.ethg[1] <--> fastEthernet <--> server1.ethg[0];
}
```

CCN Scenario files

CCN Scenario files specify what the node will do and when in terms of CCN related actions



```
[eInterestMode]
ContentName = /b3c/wowmom/movie1  , RequestTime = 1/*s*/
ContentName = /b3c/wowmom/movie2  , RequestTime = 10/*s*/

[ePreCacheMode]
ContentName = /b3c/wowmom/movie1  , StartChunk = 0 , ChunksCount = 100 , UpdateTime = 0/*s*/
ContentName = /b3c/wowmom/movie2  , StartChunk = 0 , ChunksCount = 110 , UpdateTime = 0/*s*/
ContentName = /b3c/wowmom/movie3  , StartChunk = 0 , ChunksCount = 115 , UpdateTime = 0/*s*/


[eFwdRulesMode]
ContentPrefix = /b3c/wowmom , NextHop = router1.eth[0] , AccessFrom = client1.eth[0] , UpdateTime = 0/*s*/

[eCommentsMode]
------------------------------------------------------------------------------
comments go here
```

Express *Interest* for named content at a specified time

Pre-Load content in the cache (as ranges of chunks) at a specified time

Learn about some content at a specified time, by adding a FIB entry

More to come in the future ...

CCN Lite: Where to hook what ?

- CCN application ?
  - Write the application as a separate OMNet++ simple or compound module, which communicates with the Ccn module through ***CcnAppMessages***

- Transport functionality (Strategy Layer) ?
  1. Write the Strategy as a separate OMNet++ simple or compound module (layer), which communicates with the Ccn module through ***CcnAppMessages*** (An application should then run on top of the Strategy layer, so you would have to have a north and south interface)
     - Extend the ***CcnAppMessage*** definition to match your needs for control communication between the Ccn module and the Strategy module

  2. Implement the Strategy layer as a class that derives from the Ccn class, and extends the Ccn module providing a new module (you would only need to override the ***sendInterest()*** method.
     - Extend the ***CcnAppMessage*** definition to match your needs for control communication between the Strategy module and the Application module

  3. Implement the Strategy functionality as code placed directly within the Ccn module (e.g. in ***sendInterest()*** method) – UGLIEST approach of the three but probably fastest

- Routing protocol ?
  1. Implement the Routing protocol in a class that derives from the Ccn class, and extends the Ccn module functionality as a new module.

  2. Create a separate module for the Routing protocol and then place it with the Ccn module in a compound module that specifies an extended “CCN layer” (by analogy to the network layer functionality in INET which groups ARP, routing and IP modules).

- Caching strategy ? – Hmmm.....