Cairo University
Faculty of Engineering
Computer Department

Computer Networks-1
CMP4020
Fall 2023

Introduction to Networking and Omnet++

Contact information

- E-mail: Salma.cufece@gmail.com
- Please don't use Facebook or WhatsApp.
- Office hours: Sundays from 2:30 to 4 pm.
- Classroom :
  - Name: CMP4020_Fall_2023_networks_Semester
  - code: wuyftyb

Work Year Grades

- (5) labs (5 marks) not all labs will be graded
- Project (12 marks)
- (2) quizzes (8 marks)
- Midterm (15 marks)
- Calendar link. (subject to changes)

Today’s Outline

- Network Applications
- Client Server and Peer to Peer Networks
- Labs intended outcome
- Introduction Omnet++
- How to install
- Tic Toc Demo
- Lab 1 Requirement

Network applications

When you access the Internet, do you use the following applications?

Since these applications require the access to the Internet, we refer these applications as network applications.

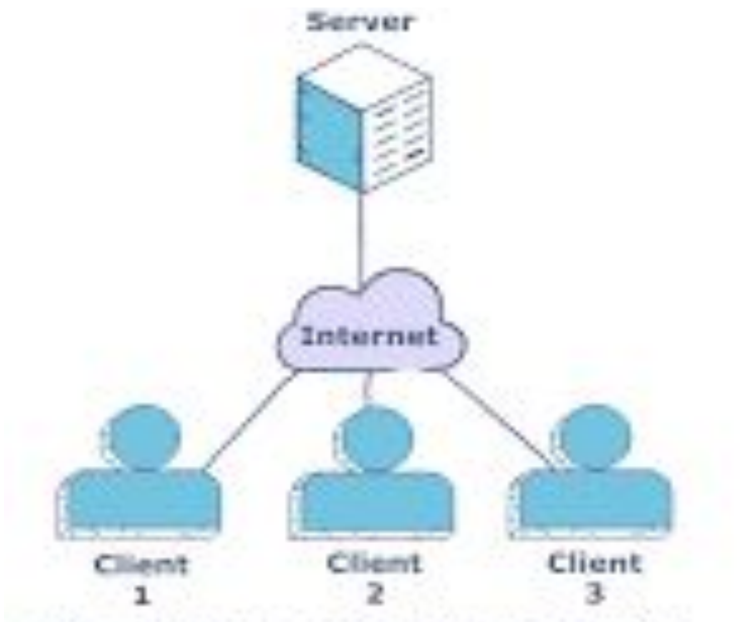
They all access some form of remote information or resources

These network applications can be technically called as network client software.

Server-Client vs Peer-Peer networks

Server-Client

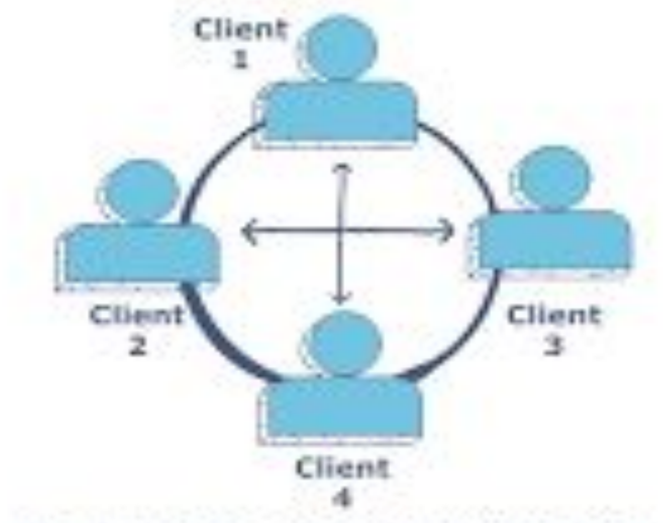
- Clients communicate with server.



Client-Server Network Model

Peer-Peer

- Clients communicate with each others directly.
- Each node is a client and a server.

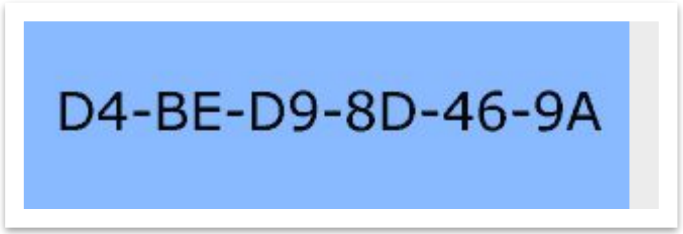


Peer-to-Peer Network Model

How Devices Communicate ?

Mac Address

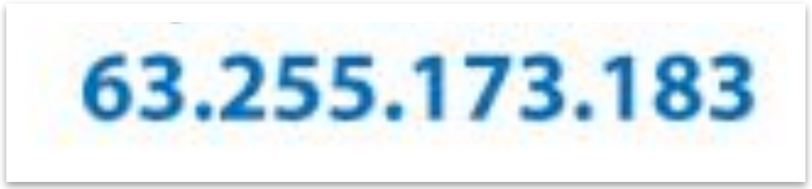
- Unique for each **Network** interface **card (NIC)** .
- Used in LANs.



D4-BE-D9-8D-46-9A

Ip Address

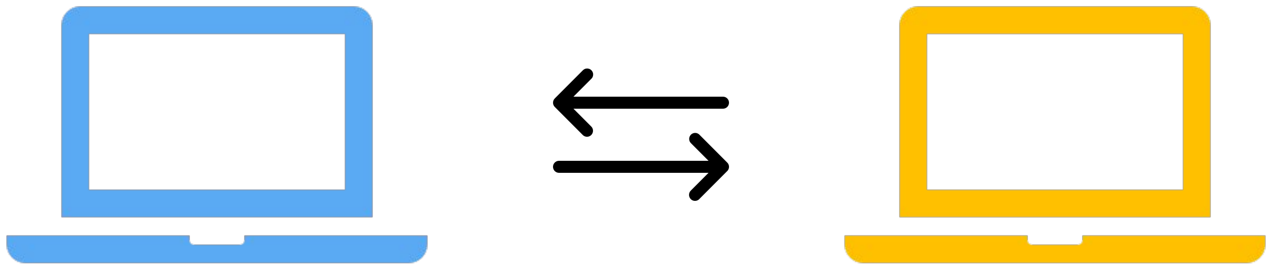
- Usually Dynamic.
- Necessary for any networks communication outside LANs.



63.255.173.183

Labs:

- We will use Peer-Peer Networks in the labs to simulate data link layer and physical layers protocols.
- Usually two nodes only.



Omnet++ [https://doc.omnetpp.org/omnetpp/manual/] Introduction

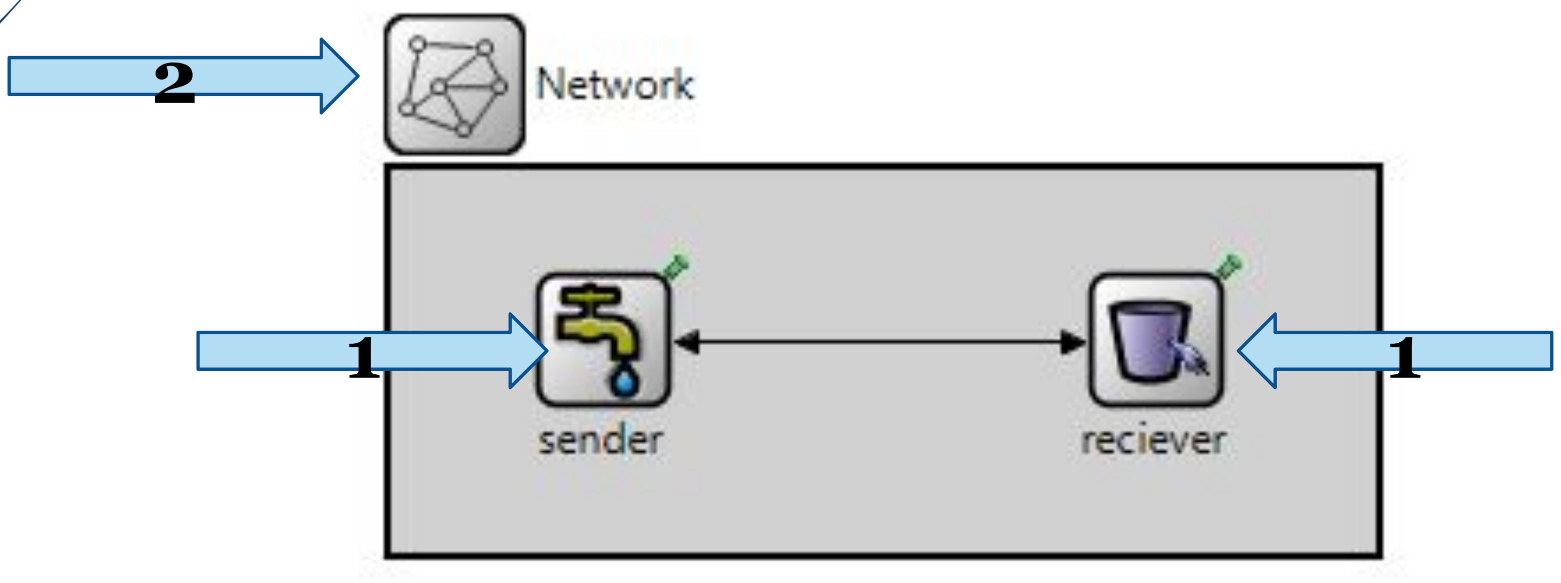
- Discrete event simulator [not limited to network protocols simulation]
  - Hierarchically nested modules
  - Modules communicate using messages through channels
- Basic machinery and tools to write simulations
  - Does not provide any components specifically for computer network simulations, queuing network simulations, system architecture simulations or any other area
- Written in C++
  - Source code publicly available
  - Simulation model for Internet, IPv6, Mobility, etc. available
- Free for academic use
  - Commercial version: OMNEST™

Omnet++

- Models algorithms through C++ class library
  - Utility classes (for random number generation, statistics collection, topology discovery etc.)
  - ➪ Use it to create simulation components (simple modules and channels)
- **And** Describes the Topology using:
  - Network Description Language (NED)
  - .ini files
- Eclipse-based simulation IDE for designing, running and evaluating simulations

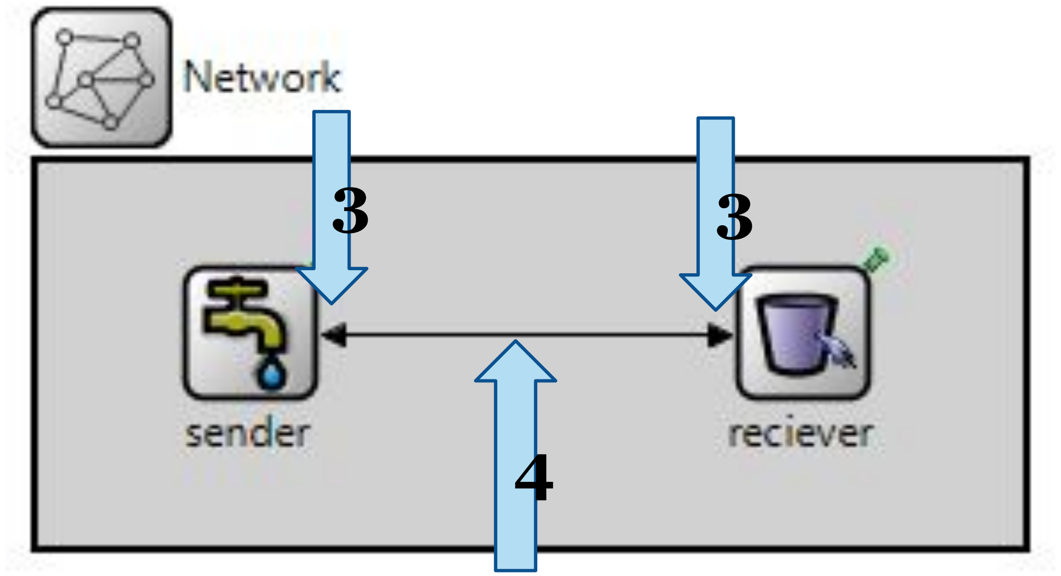
Omnet++
component based architecture

1. Models assembled from reusable components = modules [building blocks]
2. Modules can be combined (like LEGO blocks) to form compound modules.



Omnet++ component based architecture

3. Modules can be connected with each other through gates [in, out, inout]
4. Channels are used to carry messages between the gates [delay, errors, ...etc]



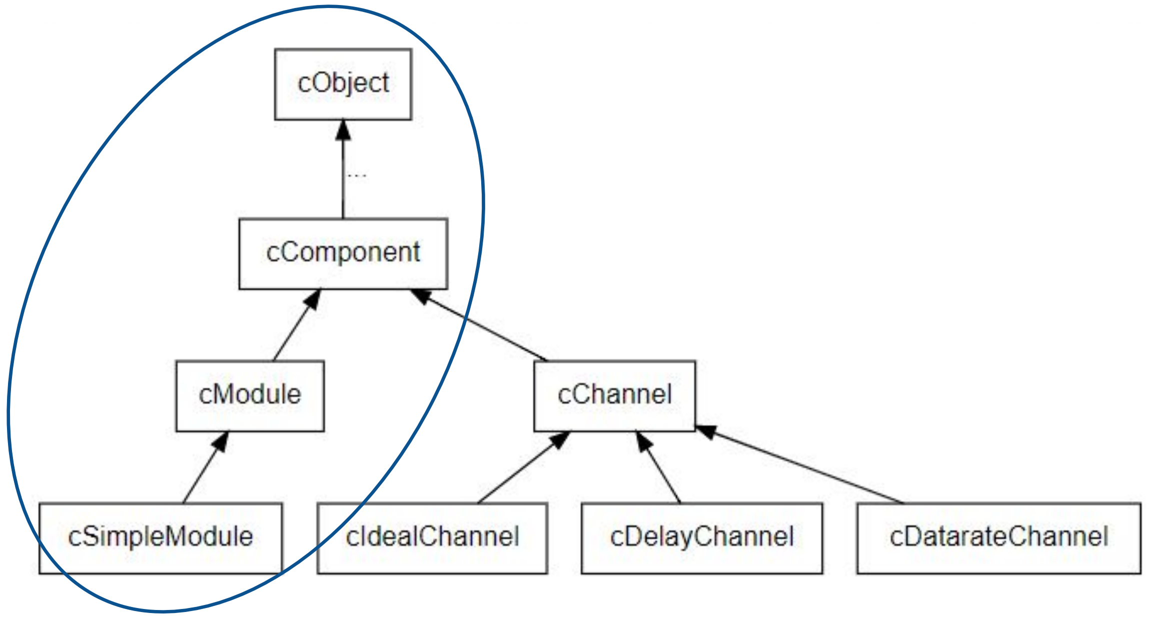
Omnet++ Hierarchical Modules

- Modules are instances of module types
- Compound Modules include one or more module.
- hierarchically nested modules
- The depth of module nesting is unlimited
- Simple modules at the lowest level of the module hierarchy

Omnet++
Hierarchical Modules

The user implements modules in C++ and the NED description.

- Usually by inheriting from the cSimpleModule class

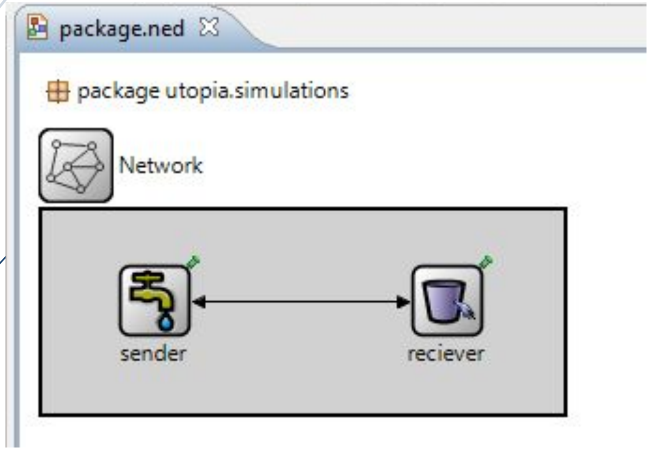


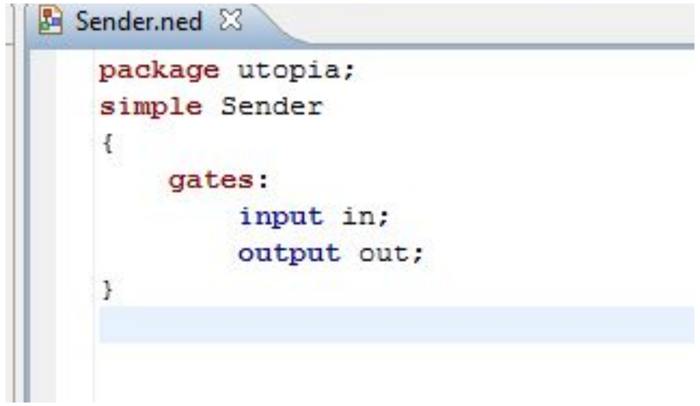
Omnet++ Hierarchical Modules

The user implements modules in C++ and the NED description.

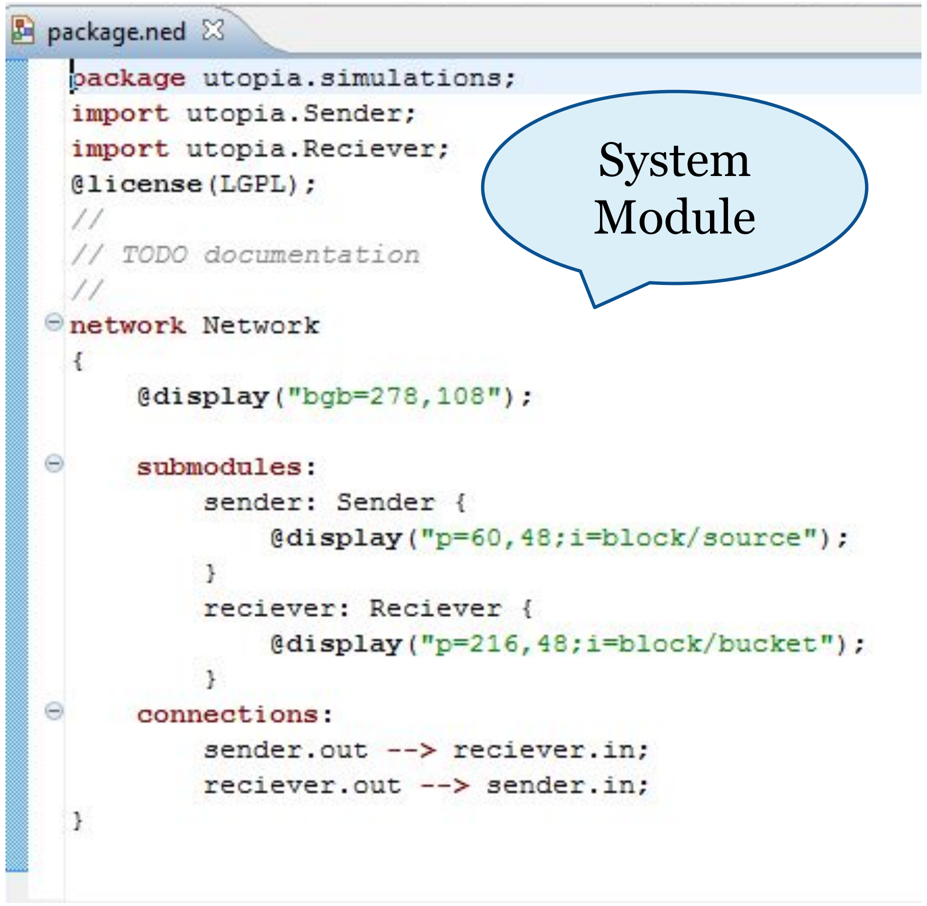
- Usually by inheriting from the cSimpleModule class
- Basically override the functions:
  - **initialize()**
  - **handleMessage(cMessage *msg)**
- Define the modules gates, parameters, and submodules in the .NED file.
- Use the manual chapter 4.2 for your reference.

Omnet++
Hierarchical Modules Example




Sender.ned

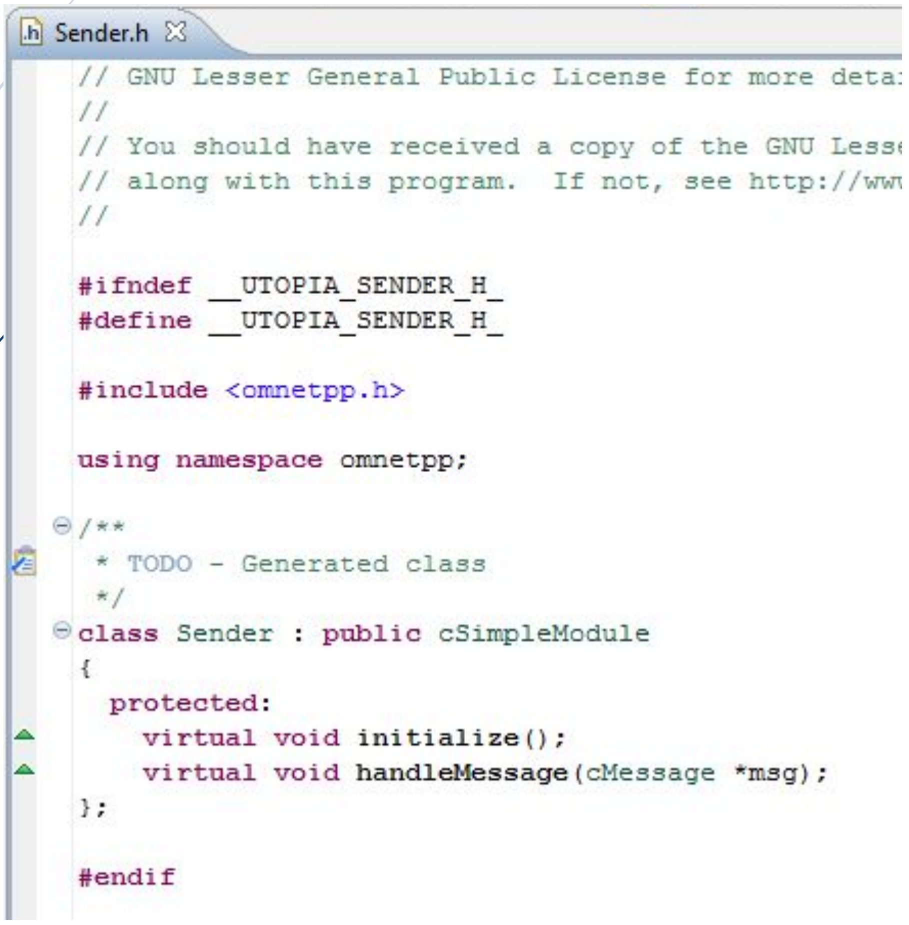
```
package utopia;
simple Sender
{
    gates:
        input in;
        output out;
}
```


package.ned

```
package utopia.simulations;
import utopia.Sender;
import utopia.Reciever;
@license(LGPL);
//
// TODO documentation
//
network Network
{
    @display("bgb=278,108");

    submodules:
        sender: Sender {
            @display("p=60,48;i=block/source");
        }
        reciever: Reciever {
            @display("p=216,48;i=block/bucket");
        }
    connections:
        sender.out --> reciever.in;
        reciever.out --> sender.in;
}
```

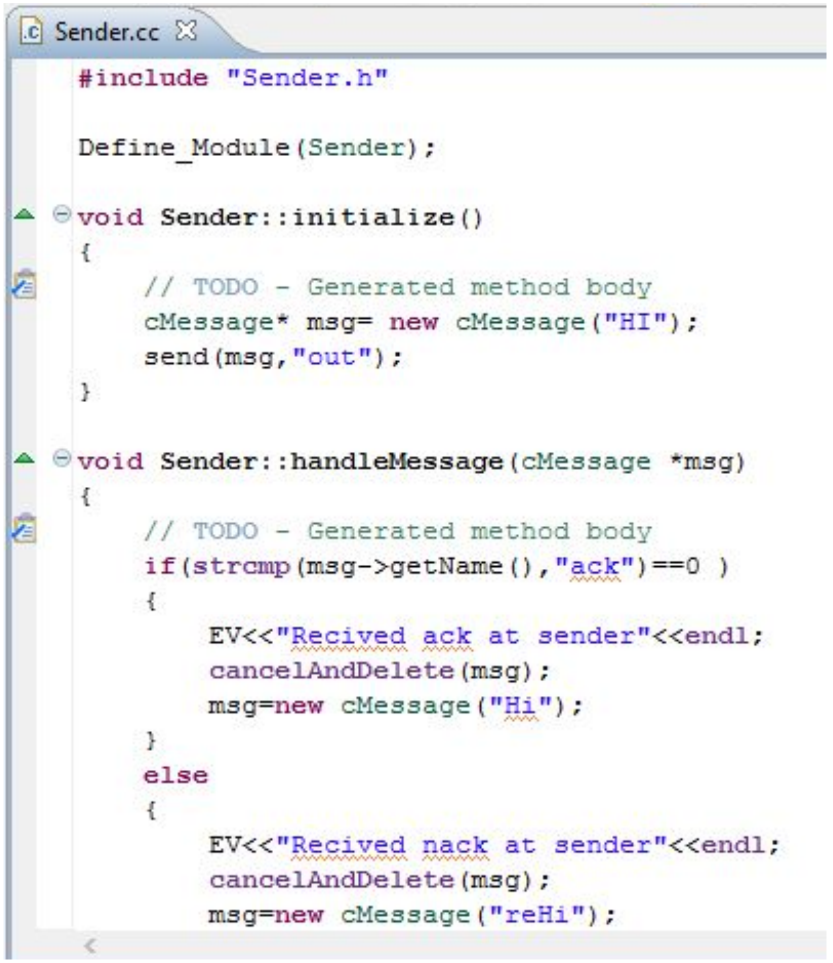
System Module

Omnet++
Hierarchical Modules Example



Sender.h

```
// GNU Lesser General Public License for more detai
//
// You should have received a copy of the GNU Lesse
// along with this program.  If not, see http://www
//

#ifndef __UTOPIA_SENDER_H_
#define __UTOPIA_SENDER_H_

#include <omnetpp.h>

using namespace omnetpp;

/**
 * TODO - Generated class
 */
class Sender : public cSimpleModule
{
  protected:
    virtual void initialize();
    virtual void handleMessage(cMessage *msg);
};

#endif
```



Sender.cc

```
#include "Sender.h"

Define_Module(Sender);

void Sender::initialize()
{
    // TODO - Generated method body
    cMessage* msg= new cMessage("HI");
    send(msg,"out");
}

void Sender::handleMessage(cMessage *msg)
{
    // TODO - Generated method body
    if(strcmp(msg->getName(),"ack")==0 )
    {
        EV<<"Recived ack at sender"<<endl;
        cancelAndDelete(msg);
        msg=new cMessage("Hi");
    }
    else
    {
        EV<<"Recived nack at sender"<<endl;
        cancelAndDelete(msg);
        msg=new cMessage("reHi");
```

Omnet++
Hierarchical Modules

initialize()

- This method is invoked after OMNeT++ has set up the network

handleMessage(cMessage *msg)

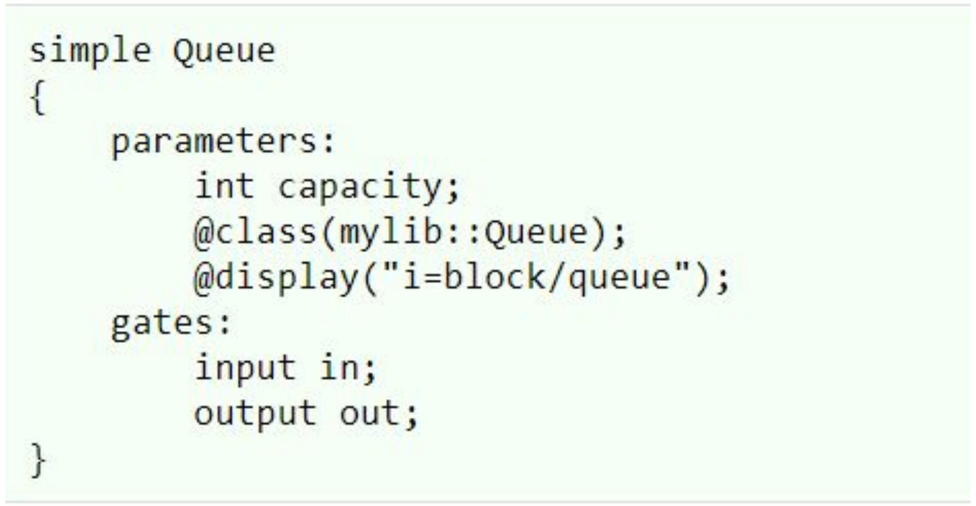
- It is invoked with the message as parameter whenever the module receives a message. handleMessage() is expected to process the message, and then return.

finish()

- is called when the simulation has terminated successfully

Omnet++
Hierarchical Modules

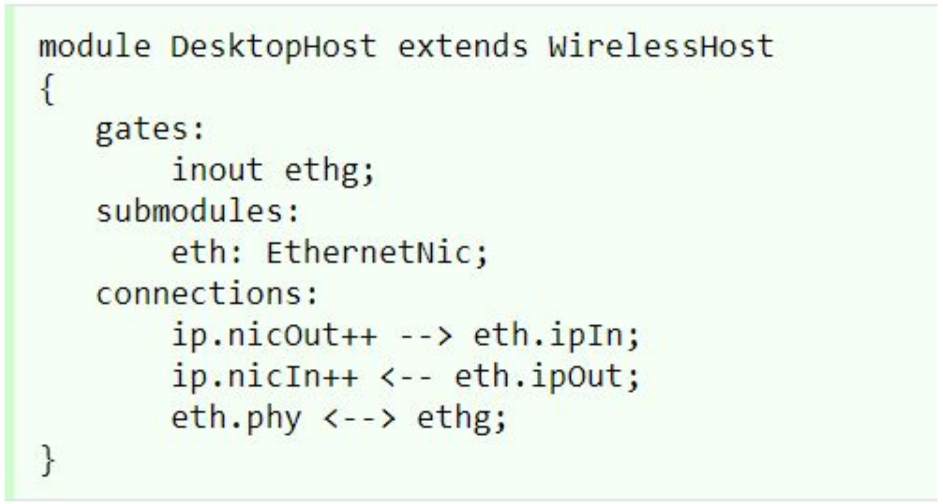
- The .NED file name should be the same as the driven simple module C++ class.
- Otherwise you should use the @class property.



```
simple Queue
{
    parameters:
        int capacity;
        @class(mylib::Queue);
        @display("i=block/queue");
    gates:
        input in;
        output out;
}
```

Omnet++ Hierarchical Modules

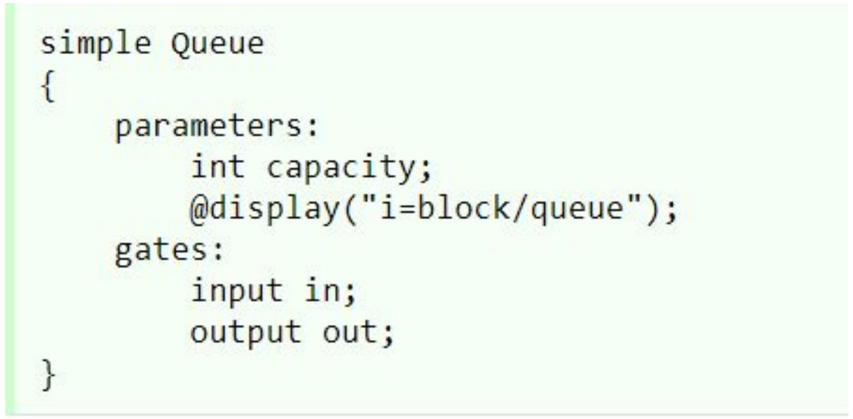
- Compound Modules in .NED contains:
  - Types (optional)
  - Parameters (optional)
  - Gates (optional)
  - Submodules
  - connections



```
module DesktopHost extends WirelessHost
{
    gates:
        inout ethg;
    submodules:
        eth: EthernetNic;
    connections:
        ip.nicOut++ --> eth.ipIn;
        ip.nicIn++ <-- eth.ipOut;
        eth.phy <--> ethg;
}
```

Omnet++ Hierarchical Modules

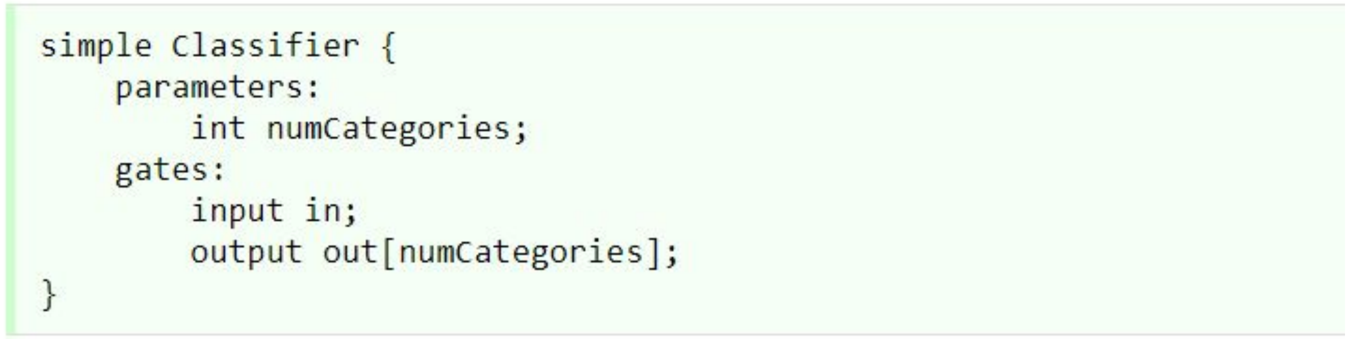
- Simple Modules in .NED contains:
  - Parameters (optional)
  - Gates [could be vectors of gates static or dynamic]



```
simple Queue
{
    parameters:
        int capacity;
        @display("i=block/queue");
    gates:
        input in;
        output out;
}
```

Omnet++
Gates

- Gates types in .NED :
  - in
  - out
  - inout (not recommended)



```
simple Classifier {
    parameters:
        int numCategories;
    gates:
        input in;
        output out[numCategories];
}
```

Omnet++ Connections

- Connections cannot span across hierarchy levels.
- Input and output gates are connected with a normal arrow

  node0.gout-->node1.gin
- Inout gates are connected with a double-headed arrow

  Node0.ginout<-->Node1.ginout
- When the ++ operator is used , it will increment the number of gates in the vector.

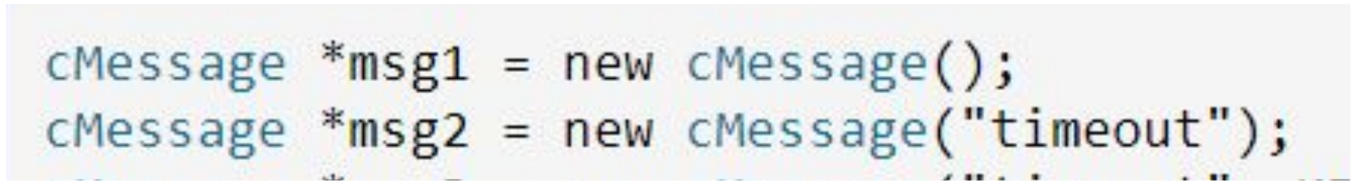
  a.gout++ --> b.gin++;

Omnet++ Messages

- Messages objects represent events, packets, commands, jobs, customers …. etc.
- Modules interact with each others though messages.
- Events are generated, scheduled ,queued , invoked as message objects.
- Messages are represented with the cMessage class and its subclass cPacket.

Omnet++ Messages

- Define new Message



```
cMessage *msg1 = new cMessage();
cMessage *msg2 = new cMessage("timeout");
```

- Duplicate Message

  *cMessage *copy = msg1->dup();*

Omnet++ Messages

- Cancel and delete Message [should always do that]

   cancelAndDelete(msg)

   Don’t use messages after deleting them.

   After this the message pointer can be re allocated to new other message
- Get message Name “content for now”

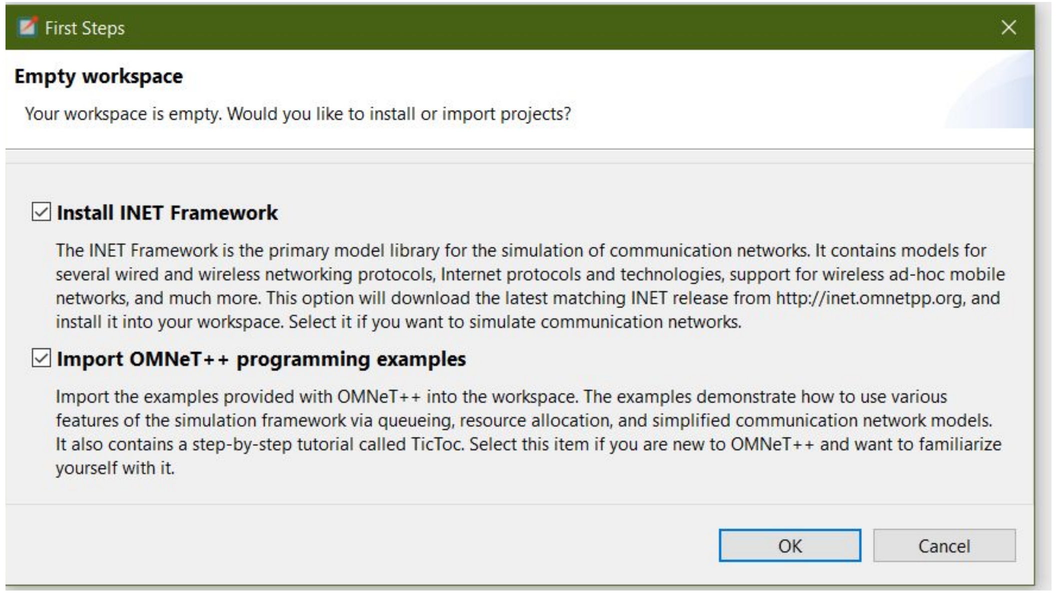
   msg->getName()
- We will visit this topic again later.
- Use the manual chapter 5 for your reference.

Omnet++
Extra Notes

- Use the EV<<“hello”<<endl; in the c++ instead of cout for debugging.
- Use getName() inside any module to get the instance name of it.
- In the .ini file , don’t forget to include the system module name.
- You can also assign global parameters in the .ini file.

Omnet++ Installation

- Install it inside the “C” drive to avoid any make errors.
- Follow this short video to install the version(5.6.2) of Omnet++ from here .
- When this screen shows, click ok and make sure that you have an internet connection on.
- Link to today’s section record.



Empty workspace

Your workspace is empty. Would you like to install or import projects?

☑ **Install INET Framework**

The INET Framework is the primary model library for the simulation of communication networks. It contains models for several wired and wireless networking protocols, Internet protocols and technologies, support for wireless ad-hoc mobile networks, and much more. This option will download the latest matching INET release from http://inet.omnetpp.org, and install it into your workspace. Select it if you want to simulate communication networks.

☑ **Import OMNeT++ programming examples**

Import the examples provided with OMNeT++ into the workspace. The examples demonstrate how to use various features of the simulation framework via queueing, resource allocation, and simplified communication network models. It also contains a step-by-step tutorial called TicToc. Select this item if you are new to OMNeT++ and want to familiarize yourself with it.

Omnet++
Lab1 requirement

- *Modify the handleMessage() method in the Tx class so that :*
- *Tic starts the conversation with the word “Tic_0”*
- *Toc responds with “Toc_1”*
- *Tic responds with “Tic_2”*
- *Toc responds with “Toc_3”*
- *………………. and so on*
- *The last message to be sent should be “Toc_9”.*

Thank You !!