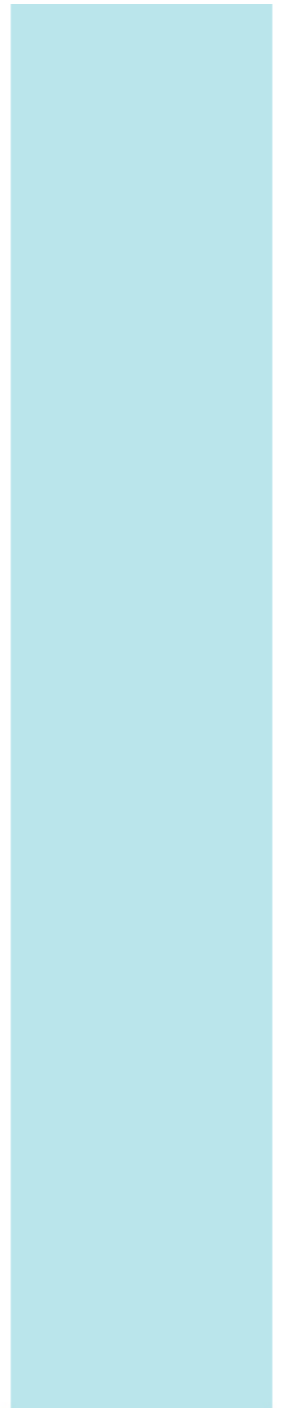


Network Simulation Using

Omnet++



SIAMAK SARMADY

Network Simulation Using Omnet++

Ver. 0.1

©2014-2016, Siamak Sarmady.

- Please inform mistakes and missing information to ([siamak.sarmady – at – gmail.com](mailto:siamak.sarmady@gmail.com))
-

Table of Contents

Simulating a Simple Ethernet Network

Create a new omnet project by using File → New → Omnet++ Project. Enter a project name and click on next.

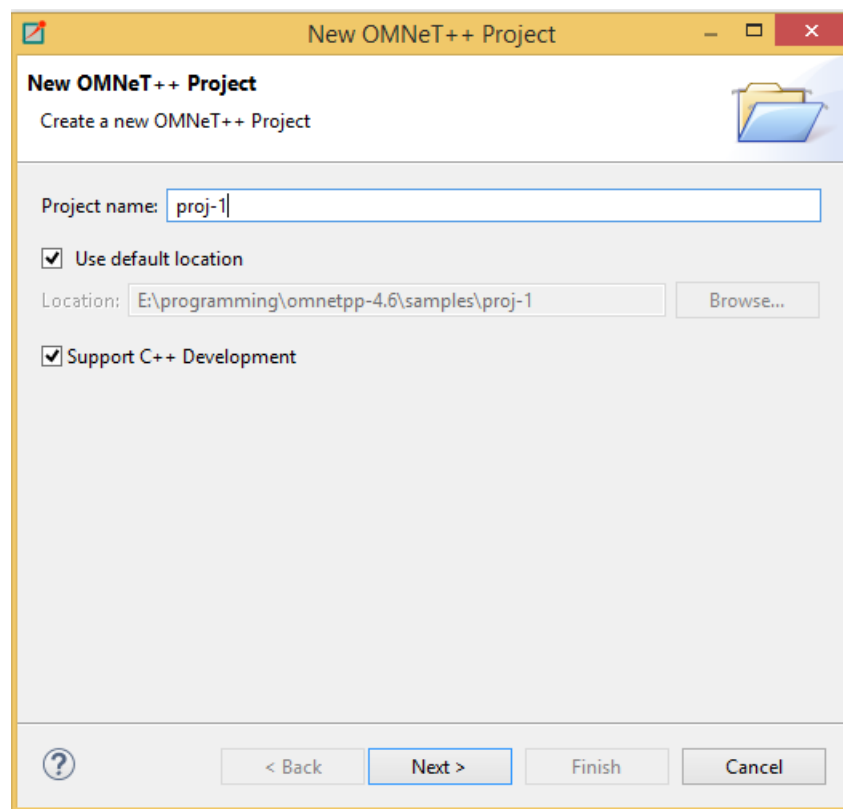
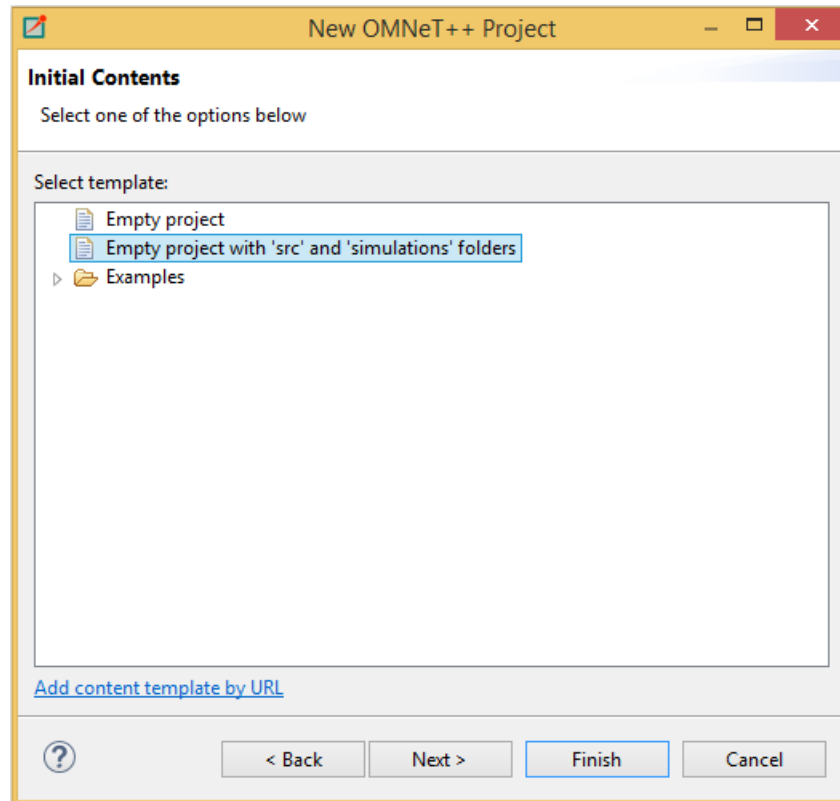


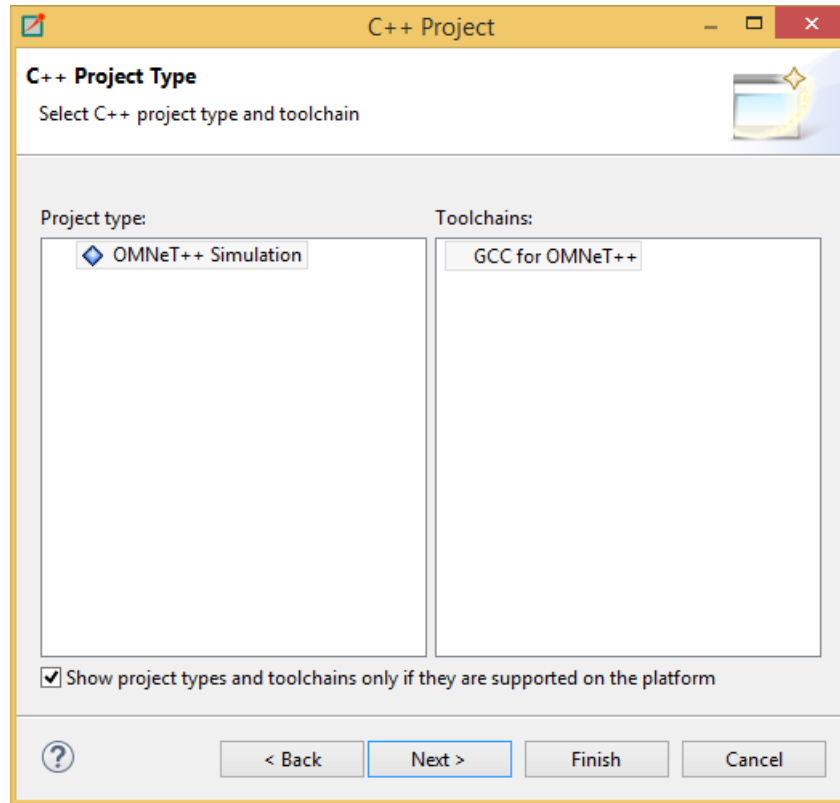
Figure 1-1: Creating a new Omnet++ project

Pressing the next button will bring you to a page in which you can select what kind of project you are planning to create.

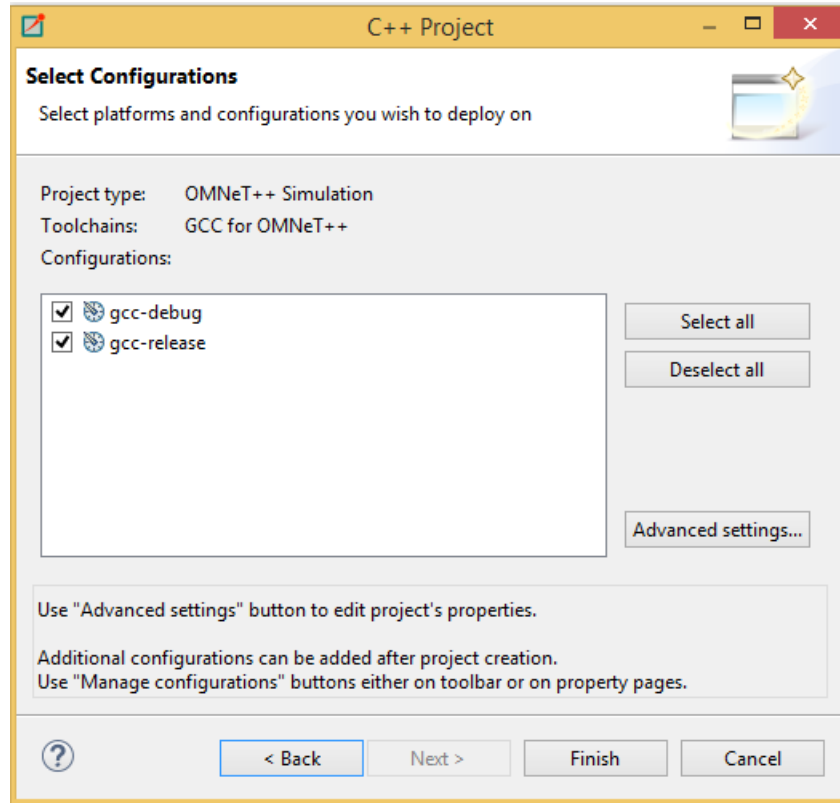
A SIMPLE ETHERNET NETWORK



Select the "Empty project with src and simulation folders". This option is same as empty project but will allow you to keep your project files in a more orderly fashion.



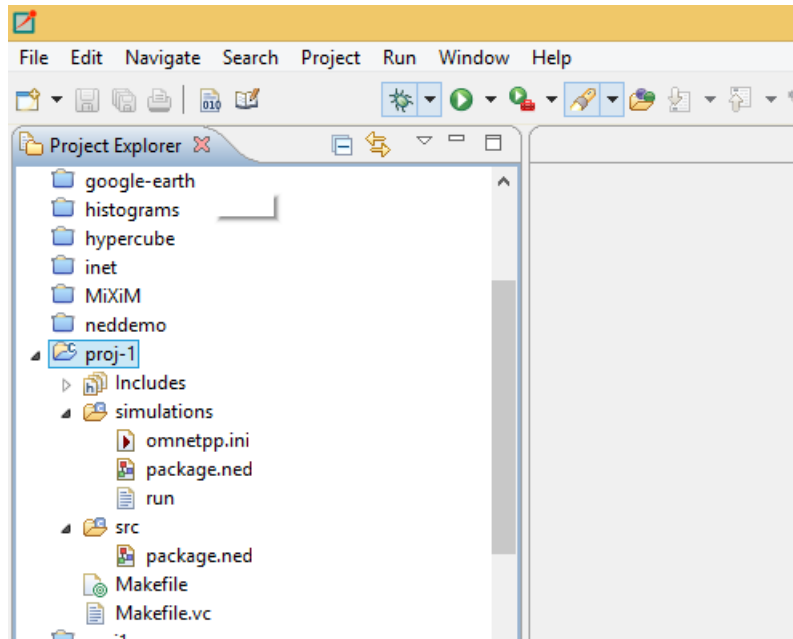
Pressing the next will bring you to the simulation tool chain selection window (compilers and simulation tools). Just accept the defaults and press next.



In the final window, you are given an option to select configurations of your project. By choosing "Advanced Settings" you can change the detailed settings of the project. These settings can be modified later. So just select finish button and Omnet++ will create the project.

If you expand the project folder in the left hand side "Project explorer" panel, you will see what files have been created for the project.

A SIMPLE ETHERNET NETWORK

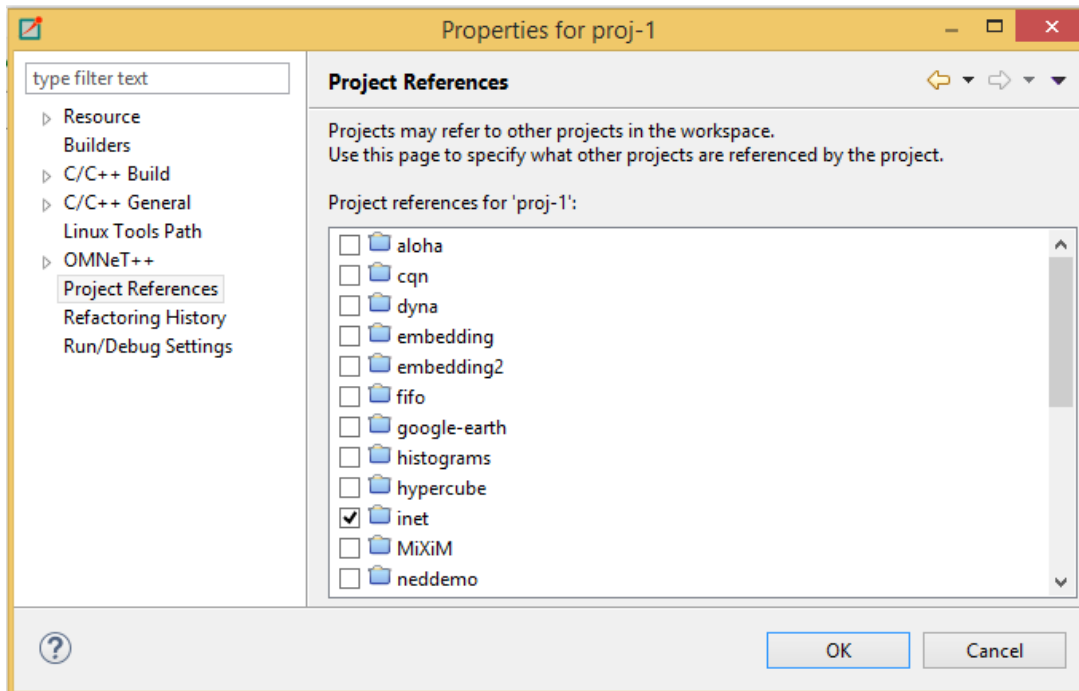


We intend to use the inet library in our simulation projects. As such we assume you have already installed and built "inet" library.

In order to use components from "inet" library, we need to enable it in our project. Right click on the project and select the "Settings...". Click on the "project references" and from the list of projects, put a tick mark on the inet project (which contains the library code).

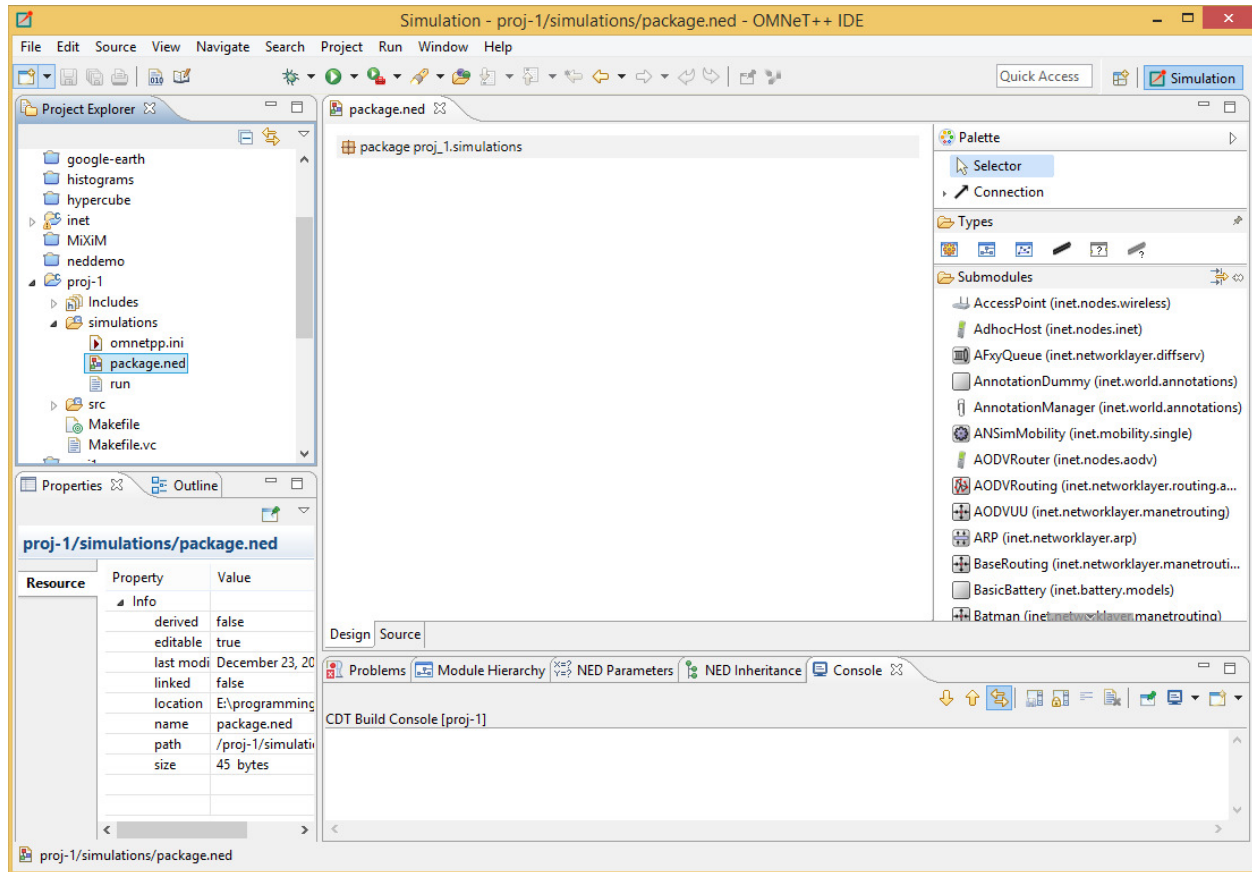
Adding the reference will cause the components to appear in the components palette (in NED design section of Omnetpp). Notice that if you do not perform this step the inet library components will not be available to you in this project.

A SIMPLE ETHERNET NETWORK



From the "Project Explorer" panel, open the simulations folder (under the project folder) and double click on the "package.ned" file. This is where we are going to design the architecture of our simple network.

A SIMPLE ETHERNET NETWORK

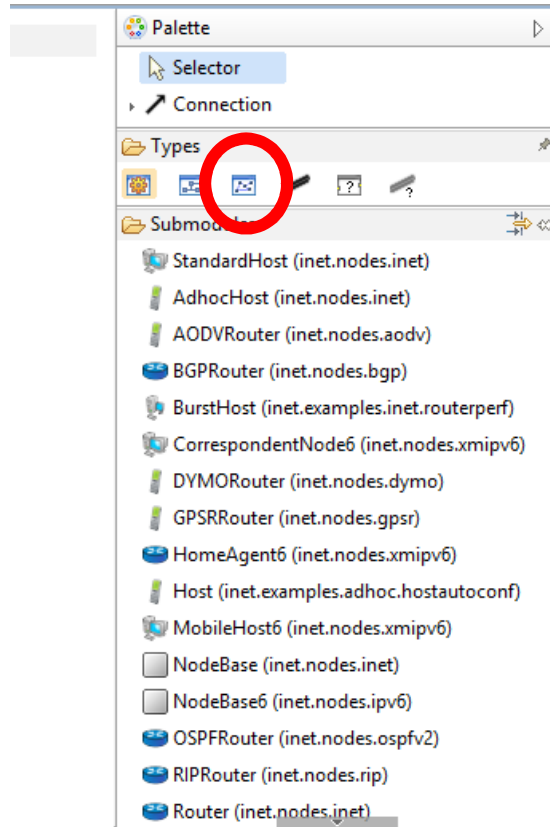


In the center of the IDE the network design area will appear. In the right hand side a palette of components (mostly from inet library) will appear. You can view the network design area either in graphical form or the source code of it. For the time being let's keep the graphical or design view.

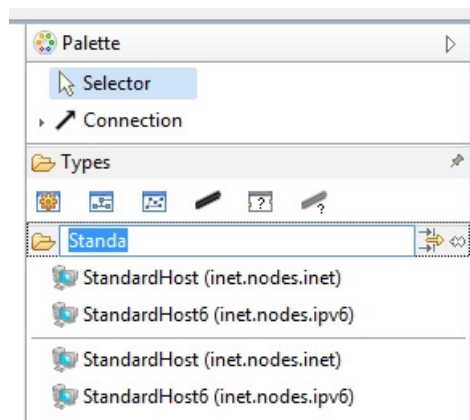
Now drag a network component and drop it into the design area. The network component is a container in which we will add the components of our network.

Note: the inet project should be open simultaneously with the project you have created, otherwise the components of inet will not appear in your palette.

A SIMPLE ETHERNET NETWORK



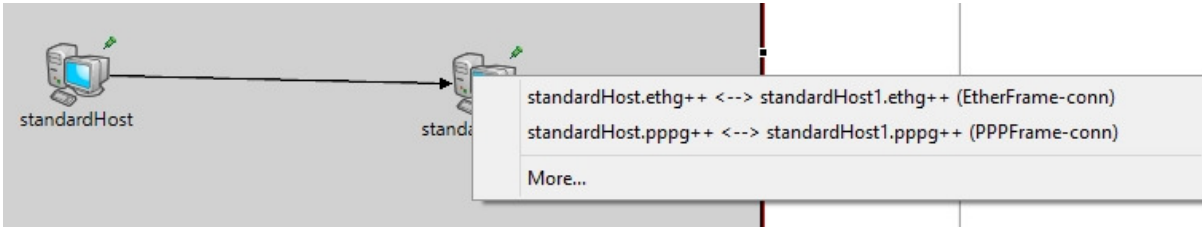
Now from the list of components, search for StandardHost. Searching is done using the search bar above the list of components.



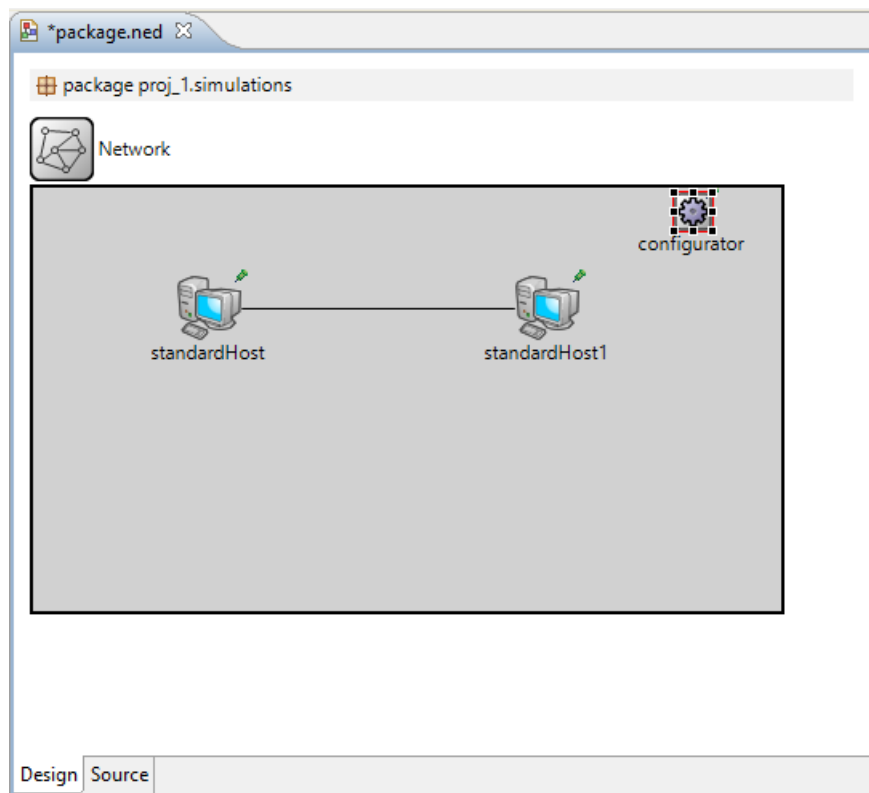
Drag two of the StandardHost components to the design area. One of them will be called StandardHost while the other will be called StandardHost1. While you can rename these components by right clicking on them and selecting properties, let's keep the default names for the time being.

A SIMPLE ETHERNET NETWORK

Now use the "Connection" tool in the palette to build a connection between the two Standard hosts. When connecting the two, select the ethg interfaces of the two hosts (and not the ppp). Ethg is an Ethernet interface while ppp is used to model a point to point connection.



Now also add an IPv4NetworkConfigurator and rename it from its properties (right click on the component) to "configurator".



Now view the source of the NED file (i.e. the network design) and add the "Ethernet100" channel type. Finally use the channel type for the connection between the two hosts (last line of the network definitions).

A SIMPLE ETHERNET NETWORK

```
package proj_1.simulations;

import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator;
import inet.nodes.inet.StandardHost;

channel Ethernet100 extends ned.DatarateChannel
{
    datarate = 100Mbps;
    delay = 100us;
    ber = 1e-10;
}

network Network
{
    @display("bgb=467,264");
    submodules:
        standardHost: StandardHost {
            @display("p=110,75");
        }
        standardHost1: StandardHost {
            @display("p=321,75");
        }
        configurator: IPv4NetworkConfigurator {
            @display("p=412,15");
        }
    connections:
        standardHost.ethg++ <--> Ethernet100 <--> standardHost1.ethg++;
}
```

Note: Check your ned file to make sure the source code looks like the above code before proceeding to the next step.

Now, in order to run the simulation we need to add a configuration. Double click on the "omnetpp.ini" file inside the simulations folder. Add the following contents to the file. These settings are the main part of the simulation configuration. They specify the specifications of each node including the configurations of each layer (physical, network, transport and application).

[General]

network = Network

*.configurator.netmask = "255.255.0.0"

*.configurator.networkAddress = "10.10.0.0"

*.standardHost.numPingApps = 1

*.standardHost.pingApp[0].destAddr = "standardHost1"

A SIMPLE ETHERNET NETWORK

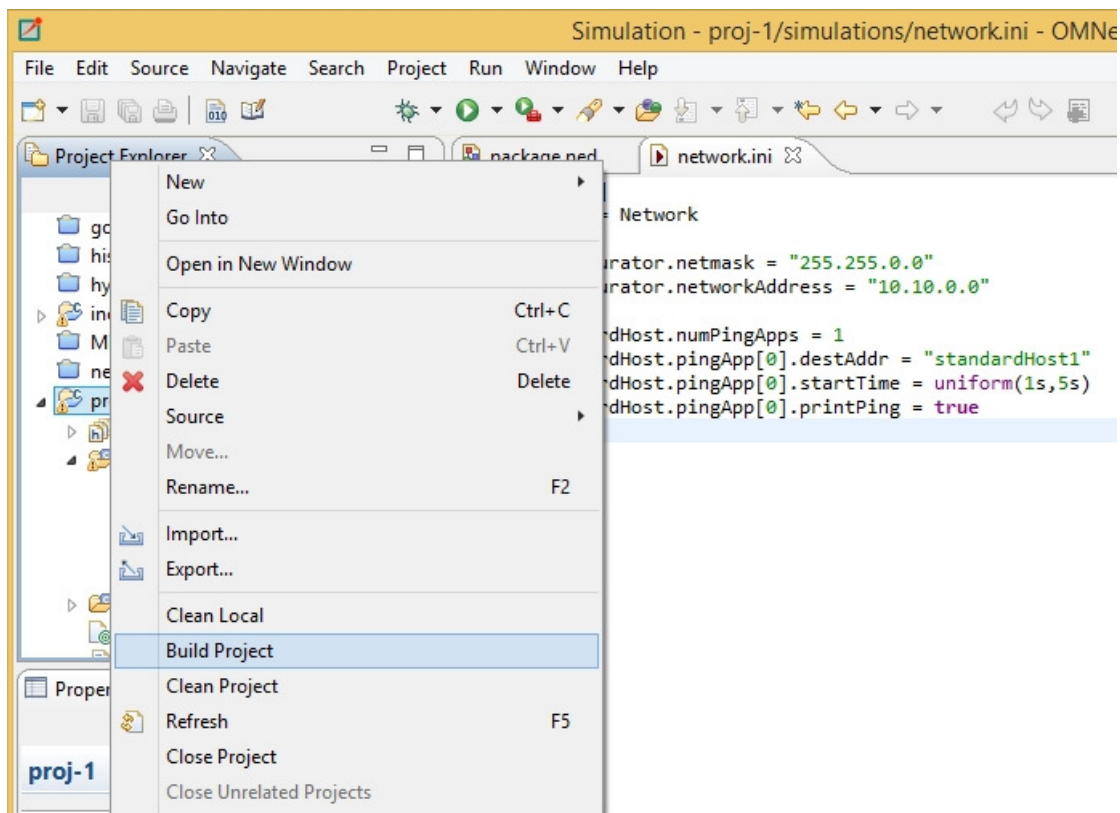
```
*.standardHost.pingApp[0].startTime = uniform(1s,5s)
```

```
*.standardHost.pingApp[0].printPing = true
```

In above configuration, we have changed the default mask and the network address of the configurator to "255.255.0.0" and "10.10.0.0" respectively. The component will assign an IP address and mask to each of the nodes.

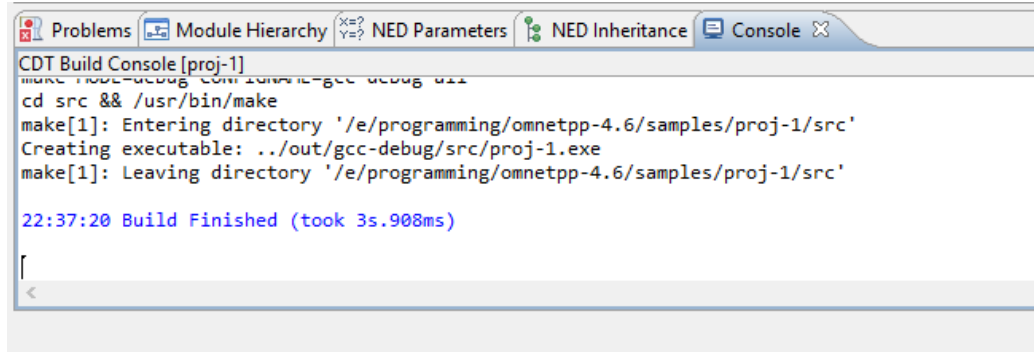
We then configure the ping-app which is already included in the StandardHost component. We configure the ping application on the first host and set the destination of the ping as the second host. We use the name of the second host for the destination address (so the ARP protocol inside the Standardhost will need to find the corresponding IP address).

Now right click on the project and select build. The build might take a few minutes.



You can check whether the build process (creating the simulation executable file) has been successful or not. In case your build has finished without any error, you may proceed to run the simulation.

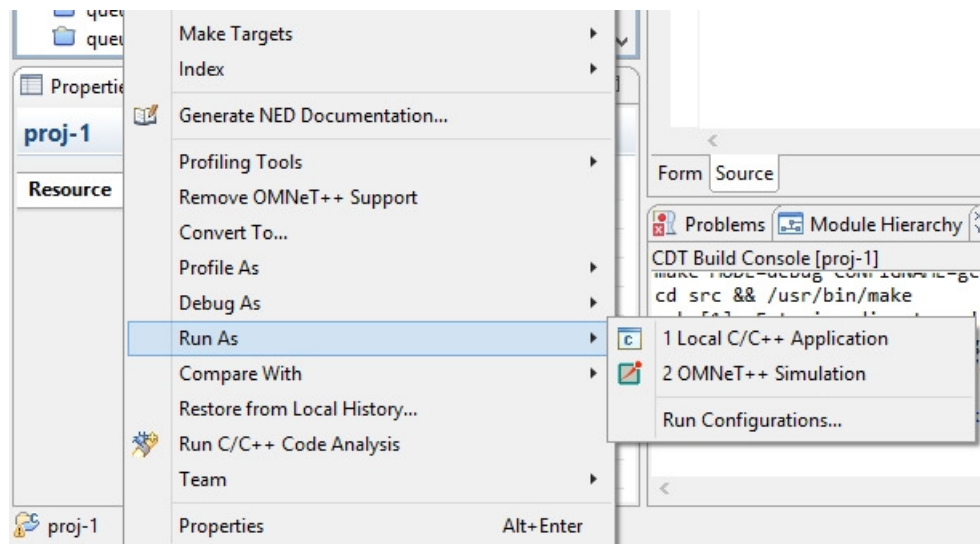
A SIMPLE ETHERNET NETWORK



```
CDT Build Console [proj-1]
make[1]: Entering directory '/e/programming/omnetpp-4.6/samples/proj-1/src'
cd src && /usr/bin/make
make[1]: Entering directory '/e/programming/omnetpp-4.6/samples/proj-1/src'
Creating executable: ../out/gcc-debug/src/proj-1.exe
make[1]: Leaving directory '/e/programming/omnetpp-4.6/samples/proj-1/src'

22:37:20 Build Finished (took 3s.908ms)
```

After the build is complete, right click on the project folder again, select the project, then select the run. From the sub menu select the "Omnet++ Simulation".



You will then need to select the initialization file of the project. Select the "omnetpp.ini" file of the project we created (i.e. proj-1).

A run configuration will be created and the executable simulation file which was created in build step will be launched. The launch process might take anything above between a few seconds and a minute. From the simulation window press the Start button and the simulation will start.

A SIMPLE ETHERNET NETWORK

The screenshot shows the OMNeT++/Tkenv simulation environment. The main window displays a network diagram with two standard hosts connected via eth0 interfaces. The network is configured with IP addresses 10.0.0.1/30. The event log at the bottom shows the following events:

Event#	Time	Src/Dest	Name
#10	3.195254009217	standardHost --> standardHost1	arpREQ ARP req: 10.0.0.2=? (s=10.0.0.1(0A-AA-00-00-00-01))
#19	3.195359769216	standardHost1 --> standardHost	arpREPLY ARP reply: 10.0.0.2=0A-AA-00-00-00-02 (d=10.0.0.1(0A-AA-00-00-00-01))
#27	3.195465529215	standardHost --> standardHost1	ping0 PING req 10.0.0.1 to 10.0.0.2 (56 bytes) id=53 seq=0 ICMP echo request 10.0.0.1 to 10.0.0.2 IPv4: 10.0.0.1 > 10.0.0.2 ETH: 0A-AA-00-00-00-01 > 0A-AA-00-00-00-02 (102 bytes)
#36	3.195573689214	standardHost1 --> standardHost	ping0-reply PING reply 10.0.0.2 to 10.0.0.1 (56 bytes) id=68 seq=0 ICMP echo reply 10.0.0.2 to 10.0.0.1 IPv4: 10.0.0.2 > 10.0.0.1 ETH: 0A-AA-00-00-00-02 > 0A-AA-00-00-00-01 (102 bytes)
#48	4.195254009217	standardHost --> standardHost1	ping1
#57	4.195362169216	standardHost1 --> standardHost	ping1-reply

The event section shows the simulation information:

#10 3.195254009217 standardHost --> standardHost1 arpREQ ARP req: 10.0.0.2=? (s=10.0.0.1(0A-AA-00-00-00-01)) ETH: 0A-AA-00-00-00-01 > FF-FF-FF-FF-FF-FF (72 bytes)

#19 3.195359769216 standardHost1 --> standardHost arpREPLY ARP reply: 10.0.0.2=0A-AA-00-00-00-02 (d=10.0.0.1(0A-AA-00-00-00-01)) ETH: 0A-AA-00-00-00-02 > 0A-AA-00-00-00-01 (72 bytes)

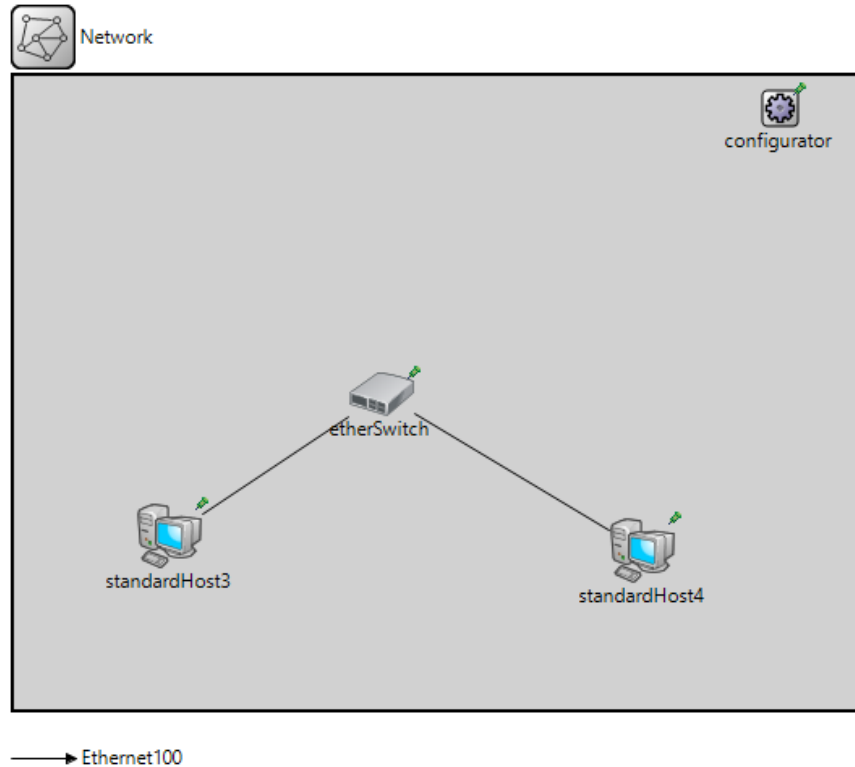
#27 3.195465529215 standardHost --> standardHost1 ping0 PING req 10.0.0.1 to 10.0.0.2 (56 bytes) id=53 seq=0 ICMP echo request 10.0.0.1 to 10.0.0.2 IPv4: 10.0.0.1 > 10.0.0.2 ETH: 0A-AA-00-00-00-01 > 0A-AA-00-00-00-02 (102 bytes)

#36 3.195573689214 standardHost1 --> standardHost ping0-reply PING reply 10.0.0.2 to 10.0.0.1 (56 bytes) id=68 seq=0 ICMP echo reply 10.0.0.2 to 10.0.0.1 IPv4: 10.0.0.2 > 10.0.0.1 ETH: 0A-AA-00-00-00-02 > 0A-AA-00-00-00-01 (102 bytes)

Apparently in the beginning an ARP request and reply have been used to find the IP address of the destination. After that the ping requests and replies have followed.

Using an Ethernet Switch

Inet library provides a component to simulate an Ethernet switch. You may use EtherSwitch for this purpose.



The following ned file shows how the above network has been configured.

```

package proj_1.simulations;

import inet.networklayer.configurator.ipv4.IPv4NetworkConfigurator;
import inet.node.ethernet.EtherHost;
import inet.node.ethernet.EtherSwitch;
import inet.node.inet.StandardHost;

network Network
{
    @display("bgb=526,391");
    submodules:
        configurator: IPv4NetworkConfigurator {
            @display("p=473,20");
        }
        etherSwitch: EtherSwitch {
            @display("p=227,196");
        }
        standardHost3: StandardHost {

```

A SIMPLE ETHERNET NETWORK

```
        @display("p=96,284");
    }
    standardHost4: StandardHost {
        @display("p=388,293");
    }
    connections:
        standardHost3.ethg++ <--> Ethernet100 <--> etherSwitch.ethg++;
        etherSwitch.ethg++ <--> Ethernet100 <--> standardHost4.ethg++;
}

channel Ethernet100 extends ned.DatarateChannel
{
    datarate = 100Mbps;
    delay = 1us;
    ber = 1e-10;
}
```

Adding a TCP application

In previous section we created traffic using Ping application and ICMP packets. Now let's use the network we created with TCP traffic.

As we stated before, the main part of the simulation is the initialization configuration file or "omnetpp.ini". We build the simulation configuration in this file.

Comment out the ping application configurations and add the following configurations instead.

[General]

network = Network

*.configurator.netmask = "255.255.0.0"

*.configurator.networkAddress = "10.10.0.0"

ICMP application

/*.standardHost.numPingApps = 1

/*.standardHost.pingApp[0].destAddr = "standardHost1"

/*.standardHost.pingApp[0].startTime = uniform(1s,5s)

/*.standardHost.pingApp[0].printPing = true

TCP application

*.standardHost.numTcpApps = 1

*.standardHost.tcpApp[0].typename = "TCPSessionApp"

*.standardHost.tcpApp[0].connectAddress = "standardHost1"

*.standardHost.tcpApp[0].connectPort = 10021

*.standardHost.tcpApp[0].tOpen = 0s

*.standardHost.tcpApp[0].tSend = 0s

*.standardHost.tcpApp[0].tClose = 0s

```
*.standardHost1.numTcpApps           = 1
*.standardHost1.tcpApp[0].typename   = "TCPSinkApp"
*.standardHost1.tcpApp[0].localAddress = "standardHost1"
*.standardHost1.tcpApp[0].localPort   = 10021
```

In the above configuration we have two series of settings. One set will configure the host which plays the role of TCP client (Standardhost) and the other configures the server node (StandardHost1). For the server we just configure one application which will be a simple TCP application called TCPSessionApp. The application will connect to host named "standardhost1" and the destination port number will be 10021. These are the addresses of the TCP server we are trying to connect. The next 3 parameters are set at zero which is enough for this experiment.

For the server we use TCPSinkApp. You could also use the TCPEchoApp but that will send back the exact information it receives for the response. We need to specify the port number on which the server will provide its service and the local network address it will bind to.

After updating the configurations, you can run the simulation again and this time you will see the TCP traffic is being simulated on the network link.

Using UDP applications

In this section we use a UDP application for the experiments. We comment out both the ping and TCP applications in our initialization configuration file and add the new applications for the UDP client and server applications.

[General]

network = Network

```
*.configurator.netmask = "255.255.0.0"
*.configurator.networkAddress = "10.10.0.0"
```

#icmp application

```
##*.standardHost.numPingApps = 1
##*.standardHost.pingApp[0].destAddr = "standardHost1"
##*.standardHost.pingApp[0].startTime = uniform(1s,5s)
##*.standardHost.pingApp[0].printPing = true
```

#TCP application

```
##*.standardHost.numTcpApps           = 1
##*.standardHost.tcpApp[0].typename   = "TCPSessionApp"
##*.standardHost.tcpApp[0].active     = true
##*.standardHost.tcpApp[0].connectAddress = "standardHost1"
##*.standardHost.tcpApp[0].connectPort   = 10021
##*.standardHost.tcpApp[0].tOpen       = 0s
```

A SIMPLE ETHERNET NETWORK

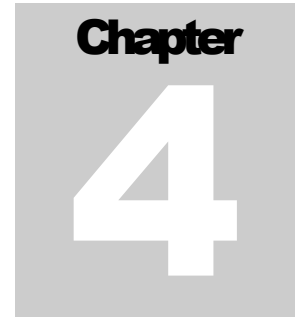
```
#*.standardHost.tcpApp[0].tSend           = 0s
#*.standardHost.tcpApp[0].tClose          = 0s

#*.standardHost1.numTcpApps               = 1
#*.standardHost1.tcpApp[0].typename      = "TCPSinkApp"
#*.standardHost1.tcpApp[0].localAddress  = "standardHost1"
#*.standardHost1.tcpApp[0].localPort     = 10021

#UDP application
*.standardHost.numUdpApps                 = 1
*.standardHost.udpApp[0].typename         = "UDPVideoStreamCli"
*.standardHost.udpApp[0].serverAddress    = "standardHost1"
*.standardHost.udpApp[0].serverPort      = 10025
*.standardHost.udpApp[0].startTime       = 0s

*.standardHost1.numUdpApps                = 1
*.standardHost1.udpApp[0].typename        = "UDPVideoStreamSvr"
*.standardHost1.udpApp[0].localAddress    = "standardHost1"
*.standardHost1.udpApp[0].localPort      = 10025
*.standardHost1.udpApp[0].sendInterval   = 0.1s
*.standardHost1.udpApp[0].packetLen      = normal(1000B, 500B)
*.standardHost1.udpApp[0].videoSize      = normal(100000B, 10000B)
```

As you see we have used a component called UDPVideoStreamCli for the client side and in the server side a UDPVideoStreamSvr component has been used. In the first set of settings we have again determined the server address and port number. In the server application settings we have again set the port address and the IP address the server will bind to. We have then determined the specifications of the video traffic the server will produce. For each of these parameters you may use different statistical distributions, plain values or



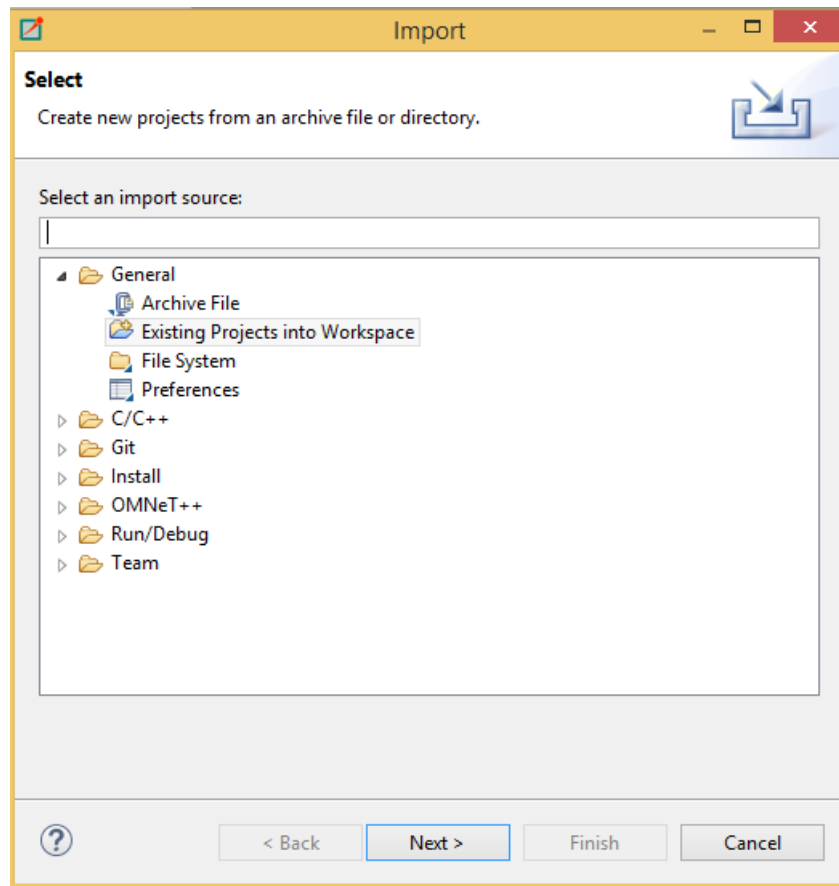
802.15.4 Network using InetManet Library

Creating 802.15.4 networks has not been possible with "Inet" library until version 2.5. However in version 3.0 the physical layer of Mixim library will be integrated into "Inet" library and the inet library will get 802.15.4 support. Mixim library itself is no more maintained. Until then it is possible to use the InetManet library for the purpose.

Installing InetManet

In order to install the InetManet library (version 2.2) follow these steps:

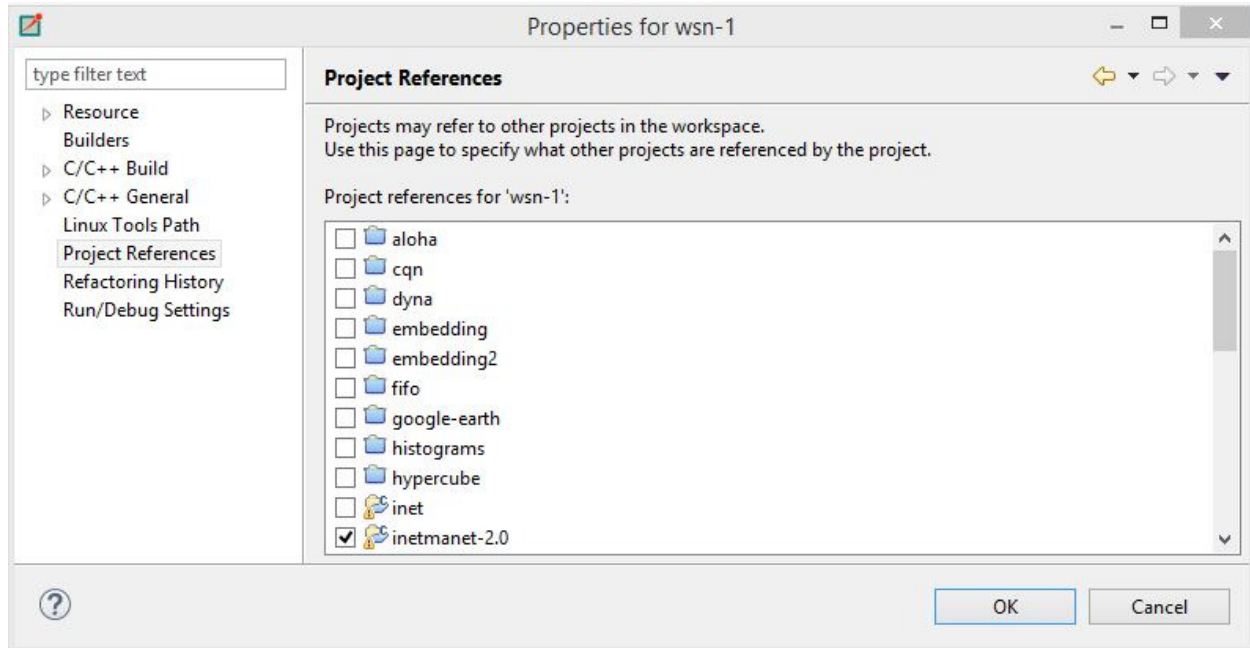
- Download the archive file from git repository of the InetManet project (<https://github.com/inetmanet/inetmanet>). From the branch selection combo box select "InetManet 2.2" and then click on the "Download Zip" button at the right hand side of the page.
- Now import the zip file into Omnet++ using File → Import → General → "Existing Project into Workspace".



- Click next and use the "Select Archive File" option to import the archive file you just downloaded.
- Now right click on the InetManet project and click on "Build" to build the project.
- After the project is built successfully, it will be available for use in your projects.

Creating the Project

Create a new project with the name "wsn-1". Again select the options to create separate directories for source (src directory) and simulations. You will then need to add a reference to the InetManet library for the project. You can either add the reference while creating the project or after that. If you have already created the project you can right click on the project and select properties. You should then add a reference in "Project References" section.



Network Design and Settings

We start by creating the network design. You may either use the components palette or just enter the source code of the network.

Assuming that the project name is "wsn1" the source code will look like the following:

```

package wsn_1.simulations;

import inet.examples.wpan.csma802154.nodecsma802154;
import inet.networklayer.autorouting.ipv4.IPv4NetworkConfigurator;
import inet.world.radio.ChannelControl;

network Network
{
    parameters:
        @display("bgb=536,253");
    submodules:
        host0: nodecsma802154 {
            @display("p=69,185");
        }
        host1: nodecsma802154 {
            parameters:
                @display("r=, ,#707070;p=439,185");
        }

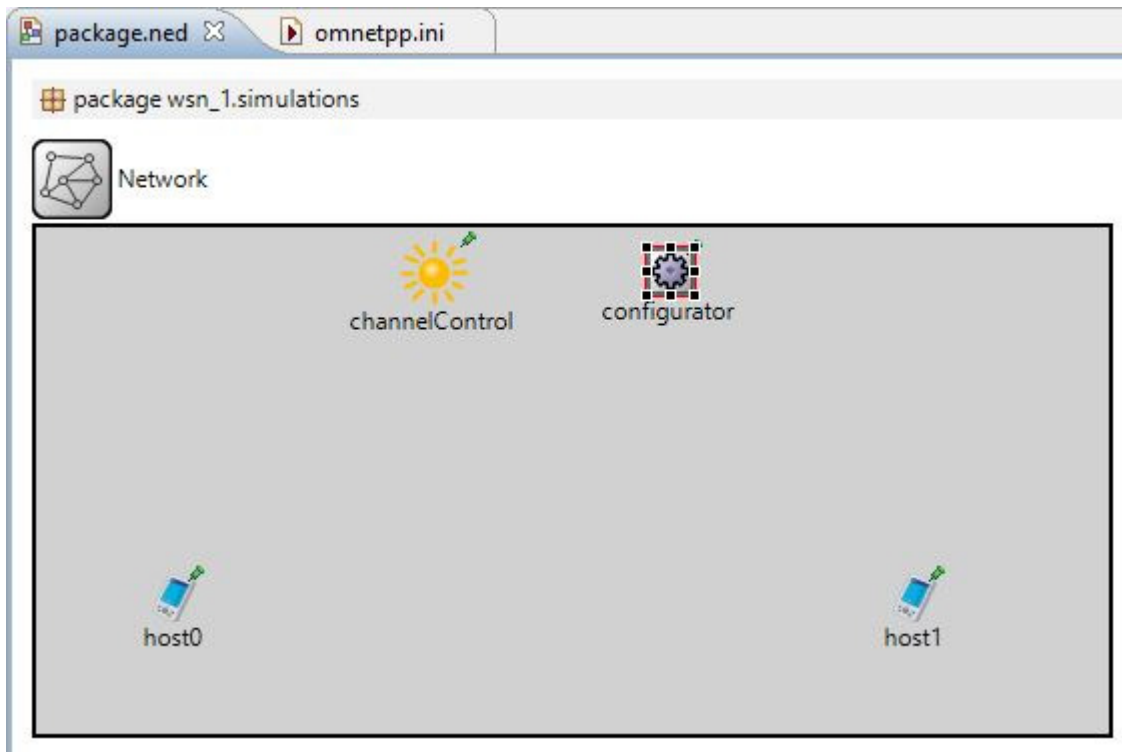
        channelControl: ChannelControl {
            parameters:
                @display("p=199,22");
        }
}

```

802.15.4 NETWORK USING INETMANET LIBRARY

```
configurator: IPv4NetworkConfigurator {  
    parameters:  
        @display("p=317,22");  
}  
}
```

The above code will result into a network design which is visualized as the following figure in the NED designer.



Now that the network design is in place, we need to add the required settings for the simulation (omnetpp.ini). The following settings are typical settings required for the project. The settings use DSRUU for the routing of the network. Since we have only two nodes the routing protocol does not matter that much. In the next example we will have more than two nodes and more settings will be provided for the routing protocol.

```
[General]  
network = Network  
  
sim-time-limit = 1000s  
output-scalar-file-append = true  
num-rngs = 2  
  
cmdenv-express-mode = true  
tkenv-plugin-path = ../../../../Etc/plugins  
  
**.vector-recording = false
```


802.15.4 NETWORK USING INETMANET LIBRARY

```
**.constraintAreaMinX = 0m
**.constraintAreaMinY = 0m
**.constraintAreaMinZ = 0m
**.constraintAreaMaxX = 600m
**.constraintAreaMaxY = 250m
**.constraintAreaMaxZ = 0m

**.debug = true
**.coreDebug = false
**.channelNumber = 0

*.host0.x = 50m
*.host0.y = 50m
*.host0.numUdpApps          = 1
*.host0.udpAppType          = "UDPVideoStreamCli"
*.host0.udpApp[0].serverAddress = "host1"
*.host0.udpApp[0].serverPort  = 10025
*.host0.udpApp[0].startTime  = 0s

*.host1.x = 150m
*.host1.y = 50m
*.host1.numUdpApps          = 1
*.host1.udpAppType          = "UDPVideoStreamSvr"
*.host1.udpApp[0].localPort  = 10025
*.host1.udpApp[0].sendInterval = 0.1s
*.host1.udpApp[0].packetLen  = normal(1000B, 500B)
*.host1.udpApp[0].videoSize  = normal(100000B, 10000B)

# ip settings
**.ip.procDelay = 10us

# ARP configuration
**.arp.cacheTimeout = 100s

#####
# Parameters for the network interface and IFqueue #
#####
**.wlan.ifqType          = "DropTailQueue"
**.ifq.frameCapacity    = 50
**.macAckWaitDuration   = 0.001s
#####
# Parameters for MAC layer #
#####

#####
# Parameters for PHY layer #
#####
**.phy.transmitterPower = 0.5mW #[mW]
**.phy.sensitivity      = -85dBm #[dBm]
**.phy.thermalNoise     = -110dBm #[dBm]
**.phy.pathLossAlpha    = 2
**.phy.snirThreshold    = 4dB

#####
# Parameters for the channel control #
```

802.15.4 NETWORK USING INETMANET LIBRARY

```
#####  
# channel physical parameters  
*.channelControl.carrierFrequency = 2.4GHz  
*.channelControl.pMax = 2.0mW  
*.channelControl.sat = -85dBm  
*.channelControl.alpha = 2  
*.channelControl.numChannels = 27  
#####  
# Parameters for the display module in the hosts #  
#####  
# display parameters (same as channelControl parameters and mac parameters)  
**.disp.carrierFrequency = 2.4GHz  
**.disp.sat = -85dBm #[dBm]  
**.disp.alpha = 2  
**.disp.transmitterPower = 1.0mW #[mW]  
**.disp.sensitivity = -85dBm #[dBm]  
  
#####  
# Parameters for the Energy Model (units: mAh and mA) #  
#####  
**.phy.usage_radio_idle = 1.38mA #[mA]  
**.phy.usage_radio_recv = 9.6mA #[mA]  
**.phy.usage_radio_sleep = 0.06mA #[mA]  
  
**.battery.nominal = 25  
**.battery.capacity = 25  
**.battery.voltage = 10  
**.battery.publishDelta = 0.5  
**.battery.publishTime = 20s  
  
**.rxSetupTime = 0.00108s  
  
#####  
# Output vectors #  
#####  
**.End-to-end delay.vector-recording = true  
**.Mean end-to-end delay.vector-recording = true  
  
#####  
# Simulation runs #  
#####  
  
# manet routing  
**.routingProtocol="DSRUU"  
  
#**.use-default=yes  
  
# proccesing delay in the routing protocol, avoid sincronization  
**.broadcastDelay=uniform(0s,0.01s) # 10 mseconds  
**.unicastDelay=uniform(0s,0.005s)  
  
# // parameters: DSRUU;  
**.PrintDebug = true  
**.FlushLinkCache = true  
**.PromiscOperation = false
```

802.15.4 NETWORK USING INETMANET LIBRARY

```
** .UseNetworkLayerAck = false
** .RouteCacheTimeout = 300 #300 seconds
** .SendBufferTimeout = 300# //30 s
** .SendBufferSize = -1
** .RequestTableSize = -1
** .RequestTableIds = -1
** .MaxRequestRexmt = -1 #// 16,
** .MaxRequestPeriod = 10 #//10 SECONDS
** .RequestPeriod = 500 #//500 MILLISECONDS
** .NonpropRequestTimeout = 30# //30 MILLISECONDS
** .RexmtBufferSize = -1 #//MAINT_BUF_MAX_LEN
** .MaintHoldoffTime = 250# //250 MILLISECONDS
** .MaxMaintRexmt = 2 # //2
** .TryPassiveAcks = true #//1
** .PassiveAckTimeout = 100# //100 MILLISECONDS
** .GratReplyHoldOff = 1 #, //1 SECONDS
** .MAX_SALVAGE_COUNT = 15 # //15
** .LifoSize = 20
** .PathCache = true
** .ETX_Active=false
** .ETXHelloInterval = 1 #, // Second
** .ETXWindowNumHello = 10
** .ETXRetryBeforeFail=-1
** .RREPDestinationOnly = false
** .RREQMaxVisit =5 # // Max Number that a RREQ can be processes by a node
```

Now that the network design and the simulation settings are done, we are ready to run the simulation. Right click on the project and build it. Then right click again and select "Run As -> Omnet++ Simulation".

The actual location of the hosts has been defined in the settings file. The hosts are not moving in this example.

A streaming video application (over UDP) has been used over the network. You may again change the settings and statistical parameters of the video stream. Since the application UDP packets are much bigger than the small MTU of the 802.15.4 network, the packets will be fragmented. The results of the simulation show this fact.

802.15.4 NETWORK USING INETMANET LIBRARY

