Iowa State University
Department of Electrical and Computer Engineering
CprE 489: Computer Networking and Data Communication
Lab Experiment #7
OPNET Tutorial

Objective

Become familiar with OPNET.

Pre-Lab

Read through the lab experiment and answer the following questions:

1) How do you launch OPNET?
2) What OPNET editor do you use to create a packet format?
3) What type of supported link do you create with the Link Model Editor?
4) What two types of OPNET models are required to define a peripheral node?
5) How do you verify that the switch node is connected to the peripheral nodes properly?
6) What five nodes do you use to build the tutorial network?
7) What is one metric that you will be collecting during the simulation?

Lab Expectations

Work through the lab and let the TA know if you have any questions. **Demonstrate your simulation for the exercise to the TA after you have completed it.** After the lab, write up a lab report with your partner. Be sure to

1) summarize what you learned in a few paragraphs
2) include your answers to the pre-lab questions
3) include your server and client Function Block codes for the exercise
4) specify the effort levels of each group member (totaling to 100%)
5) submit your lab report with a lab feedback form

Your lab report is due at the beginning of the next lab.

Problem Description

This introductory tutorial will guide you through some basic yet important OPNET components to create a simple network that consists of four peripheral nodes that generate traffic and a switch node that receives packets, processes them to know the destination address, and finally forwards the packet to the correct node.

This tutorial is presented in seven sections as follows:

1) Launching OPNET
2) Creating a Packet Format
3) Creating a Link Model
4) Creating Node and Process Models for the Peripheral Node
5) Creating Node and Process Models for the Switch Node
6) Building the Network
7) Running the Simulation

NOTE: Save yourself time now and work through this tutorial very carefully. Small mistakes are difficult to trace and may produce unexpected simulation results.

Procedure

1) Launching OPNET

Launch OPNET by entering `opnet` at the command prompt.

2) Creating a Packet Format

Packet formats are created with the Packet Format Editor. The Packet Format Editor lets you define the internal structure of a packet as a set of fields. A packet format contains one or more fields, represented in the editor as rectangular boxes. The size of the box is proportional to the number of bits specified as the field's size.

To create a new packet format:

1. Choose **File → New**, and then choose **Packet Format** from the list and click **OK**.
2. Choose  from the toolbar to create packet fields.
3. Packet fields are created by left-clicking in the editor window; we want to create two fields in the packet: source address and destination address.
4. Right-click to end the creation process.
5. Right-click on the first field and choose **Edit Attributes**.
6. Verify that **type = integer**.
7. Make the following changes:
    a. Name = **source_address**
    b. Set at creation = **unset** (like many other OPNET interfaces, this is a drop down menu)
8. Click **OK** to close the attributes box.
9. Do the same for the second field except for the name, which should be set to **dest_address**.
10. Choose **File → Save**, and save the packet format under the name **grp#_tut_pkt** (e.g., grp12_tut_pkt).

This completes creating the packet format.

3) Creating a Link Model

Link models are created with the Link Model Editor. Each new type of link can have different attribute interfaces and representation.

To create a new link model:

1. Open the link model editor by choosing **File → New**, and then select **Link Model** and click **OK**.
2. We are concerned with the point-to-point communication, so make sure that the *only* supported link type is **ptdup** (point-to-point duplex) in the Supported Link Types table under the Supported column.
3. Now we must ensure that our link supports the packet format that we created previously. In the Attributes table scroll down to the **packet formats** attribute, click in the **Initial Value** column. A list with all the supported packet formats will appear. **Uncheck** the two boxes below the list, then in the **Packet Formats** column look for the packet format we created in the first part of the tutorial (**grp#_tut_pkt**), change the value in the **Status** column to **supported**, finally click **OK** to close the dialog box.
4. Set the following attributes:
    a. **data rate** = **9600** (bps)
    b. **ecc model** (error correction model) = **ecc_zero_err** (no errors)
    c. **error model** = **error_zero_err** (no errors)
    d. **propdel model** (propagation delay model) = **dpt_propdel** (point-to-point propagation delay model, depends on the distance)

e. **txdel model** (transmission delay model) = **dpt_txdel** (point-to-point transmission delay, packet length (bits) / data rate (bps))

5. Choose **File → Declare External Files**, and select **link_delay** from the list.
6. Choose **File → Save**. Name the link **grp#_tut_link**.

This completes creating the link model.

4) Creating Node and Process Models for the Peripheral Node

Node Model

Node models are created using the Node Model Editor which allows you to define the behavior of each network object. Behavior is defined using different modules, each of which models some internal aspect of node behavior such as data creation, data storage, etc. Modules are connected through packet streams (or statistic wires). A network object is typically made up of multiple modules that define its behavior.

The functions of the peripheral node include:

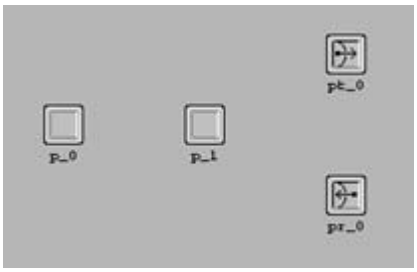
1) Generating packets (grp#_tut_pkt)
2) Filling the fields in generated packets (source and dest addresses)
3) Sending packets to different targets (randomly chosen)
4) Receiving packets from other nodes
5) Measuring the end-to-end delay
6) Destroy the received packet

To be able to do all the above functions the node should contain:

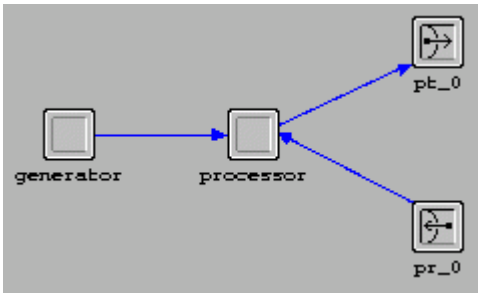
- A packet generator (function 1)
- A processing unit (functions 2, 5 and 6)
- A transmitter (function 3)
- A receiver (function 4)

To create the node model for the peripheral nodes:

1. Choose **File → New**, select **Node Model** from the list and click **OK**. The Node Model Editor opens.
2. Two kinds of modules with processing abilities can be created, processors  and queues  . **Place two processors** in the editor field – one for the **packet generator** and the other for the **processing unit**.
3. Place a point-to-point transmitter  .
4. Place a point-to-point receiver  .
5. Order the modules to look like this:



6. Right-click on p_0 and choose **Edit Attributes**.
7. In the (p_0) Attributes dialog box do the following:
    a. Set **name** = **generator**
    b. Click in the Value field for the **process model** attribute, a list will open, from the list choose **simple_source**, the available attributes will change according to the chosen process model.
    c. Set the Value for the **Packet Format** attribute to **grp#_tut_pkt**, so that the generated packets are of this format
    d. **Start time** = **0.0**
    e. Righ-click in the Value field for **Packet Interarrival Time** and choose **Promote Attribute to Higher Level**, to be able to set and change this attribute in the project editor (a higher level)
    f. Click **OK**
8. For p_1 we will only change the name (the rest will be done later). To do so right-click on p_1, choose **Set Name**, change it to **processor** and click **OK**.
9. Left-click on pt_0 (the transmitter), press and hold the shift key then left click on pr_0 (the receiver), to select both of them together. Right-click on one of them and choose **Edit Attributes**, in the lower left corner select the "**Apply changes to selected objects**" checkbox so that the changes will apply for the two modules. Click on the value of the channel attribute and change the **data rate** to **9600** and the **packet formats** to **ONLY grp#_tut_pkt** (don't forget to **uncheck the two boxes below** the packet format list), to make sure that both the transmitter and receiver are using the correct data rate and support the appropriate packet format, click **OK** twice and say **YES** to the warning that will appear to return to the editor window.
10. Using packet streams  connect the modules as shown in the figure below:



To connect two modules together click the packet stream button in the toolbar, click on the first module, and then click on the second module.

11. To verify the connectivity right-click on the processor and choose **Show Connectivity**. You will see the list of streams going into and out of the processor, verify that:
    a. Processor output port #0 is connected to the transmitter input port #0 (processor [0] → pt_0 [0])
    b. Processor input port #0 is connected to the generator output port #0 (generator [0] → processor [0])
    c. Processor input port #1 is connected to the receiver output port #0 (pr_0 [0] → processor [1])
    d. Close the Show Connectivity window
12. Choose **Interface → Node Interfaces**, the Node Interfaces window will appear. In the **Node Types** table under the **Supported** column change the values to **no** except for the **fixed** node type. In the Attributes table change the status of all attributes to **hidden** except for the **generator.Packet Interarrival Time** which should be promoted.
13. Choose **File → Save**. Name the file **grp#_tut_node**, and leave the node editor open.

The Node Model is almost finished; we will now create the process model that will be assigned to the processor unit.

Process Model

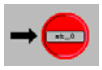
Process models are created using the Process Model Editor. The Process Editor lets you create process models which control the underlying functionality of the node models created in the Node Editor. Process models are represented by finite state machines (FSMs), and are created with icons that represent states and lines that represent transitions between states. Operations performed in a state or on a transition are described in embedded C or C++ code blocks. Each state contains two code blocks: the Enter Executive block that is executed as the state is entered and the Exit Executive block which is executed when the state is left.

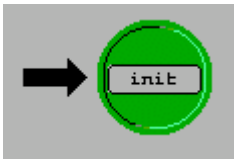
States can either be forced  or unforced  . A forced state will execute its enter executives and exit executives, then pass control to another state via a transition. An unforced state will execute its enter executives and pause, allowing the simulation to turn its attention to other entities and events in the model. A process enters an unforced state to await further interrupts, such as arriving packets or timer expiration.

As we said above the processor should be able to:

- Fill packet fields appropriately
- Measure end-to-end delay
- Destroy the packet

To create the process model for the peripheral nodes:

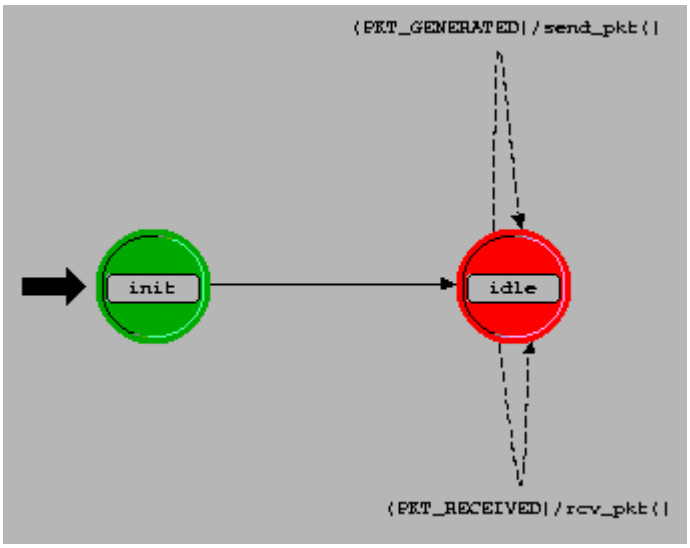
1. Choose **File → New**, select **Process Model** from the list and click **OK**. The Process Model Editor will appear.
2. Click on the create state button  in the toolbar, and place two states in the editor field. Right-click to end the state creation process.
3. The black arrow in front of state st_0  indicates that this will be the initial state where the execution will begin. The primary goal of this state is to initialize the State Variables  , thus we will enter this state only once at the beginning of execution and will never return to it. That is the reason why we will make it a forced state in the next step.
4. Right-click on st_0 and choose **Edit Attributes**. Set the **name** = **init** and the **status** = **forced**, click **OK**. The state color should become green to indicate that it is a forced state, as shown below.



5. Edit the attributes of st_1 and change the **name** = **idle** and leave the status as unforced.
6. Click on the create transition button  on the toolbar, and create the following transitions:
   a. From **init state to idle** state
   b. From **idle state and back to itself**. Right-click on this transition and choose **Edit Attributes**, change the **condition** value to **PKT_GENERATED** and the **executive** to **send_pkt()**. These changes mean that this transition takes place only when the

PKT_GENERATED condition is satisfied (=TRUE), and the send_pkt() function is executed when the transition occurs to fill the fields of the generated packet and send it. Finally, click **OK**. Unlike the previous transition in (a), the line representing this transition will become dashed to indicate that this is a conditional transition, and a label will appear that states the condition and the corresponding executive. Conditions are usually defined in the Header Block  and functions (executives) are defined in the Function Block  .

c. The only thing left is dealing with the received packets from other nodes, so create another **transition from the idle state to itself again** and edit its attributes to change the condition to **PKT_RECEIVED** and the executive to **rcv_pkt()**. The state transition diagram should look like the figure below.



7. Click on  to enter the state variables as shown in the table below, and then click OK.

Type	Name
int	src_addr
Distribution *	dist_addr
Stathandle	ete_delay

The ***ete_delay*** state variable is a statistic handler that we will use to record the end-to-end delay value. The ***dist_addr*** state variable is a pointer to a distribution – the type of the distribution will be stated later. (Note that dist_addr does not represent the destination address.) The scope of the state variables is the whole process model; they can be accessed in any code block within the process model.

8. Choose **Interfaces → Global Statistics**. The Global Statistics dialog box appears; in the first row in the table set the Stat name to **ETE Delay** and click OK. This will enable us to choose the ETE Delay global statistic from the project editor. We are not done yet with this statistic – we need to associate it with the ete_delay state variable, and this is usually done in the init state.
9. Choose **Interfaces → Model Attributes**. The Model Attributes dialog box appears; in the first row set the **attribute name** to **Node Address** and the **Type** to **integer**. Click **OK**. This will allow us to set the node address from the project editor and will allow it to vary from node to node,

reading the value of this attribute and assigning it to the state variable ***src_addr*** is done in the init state.

10. Open the Header Block by clicking on  and enter the following:

```
// Define stream numbers

#define GNR_STRM 0
#define RCV_STRM 1
#define TX_STRM 0

// Define transition macros

#define PKT_GENERATED (op_intrpt_type() == OPC_INTRPT_STRM && op_intrpt_strm() ==
GNR_STRM)
#define PKT_RECEIVED (op_intrpt_type() == OPC_INTRPT_STRM  && op_intrpt_strm() ==
RCV_STRM)
```

Save the Header Block by choosing **File → Save**.

The PKT_GENERATED condition becomes true when the interrupt is a stream interrupt (i.e. packet arrival on a stream) and the stream is the one coming from the generator. If you remember the generator was connected to the processor on input port #0 – that's why we defined GNR_STRM as 0. The same goes for the PKT_RECEIVED macro.

11. Open the Enter Executive of the init state by double clicking on the upper half of the init state, and write the following:

```
op_ima_obj_attr_get(op_id_self(), "Node Address", &src_addr);
dist_addr = op_dist_load ("uniform_int", 0, 3);
ete_delay = op_stat_reg ("ETE Delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
```

Save the Enter Executive.

The first line reads the value of the Node Address attribute (that is set at a higher level in the project editor) and puts it in the state variable src_addr. The second line assigns a uniform integer distribution to the state variable dist_addr that randomly returns integer values between 0 and 3. The third line maps the ete_delay state variable to the global statistic ETE Delay.

12. We now need to define the transition executives, click on the Function block button  in the toolbar and write down the following:

```
void send_pkt()
{
 Packet * pkptr;
 pkptr = op_pk_get (GNR_STRM);

 int d;
 d = (int)op_dist_outcome (dist_addr);

 if(d == src_addr) {
  op_pk_destroy(pkptr);
 }
 else {
  op_pk_nfd_set_int32 (pkptr, "dest_address", d);
  op_pk_nfd_set_int32 (pkptr, "source_address", src_addr);
  op_pk_send (pkptr, TX_STRM);
 }
}

void rcv_pkt()
{
 Packet *pkptr;
 double ete;
 pkptr = op_pk_get (RCV_STRM);
 ete = op_sim_time () - op_pk_creation_time_get (pkptr);
 op_stat_write (ete_delay, ete);
 op_pk_destroy (pkptr);
}
```

Save the Function Block.

The first line in the send_pkt() function defines a pointer to a packet. The following line assigns this pointer to the packet coming from the generator stream. The next two lines initialize the destination address. The following if statement determines whether to send the packet or not. If the packet's source address is the same as the generated destination address (if condition), the packet is destroyed and nothing is sent. If the packet has a destination address that is not the same as the source address (else condition), the packet is okay to be sent and the fields in the packet pointed to by "pkptr" are initialized. The last statement sends this packet on the stream going from the processor to the transmitter.

The first three lines in rcv_pkt() should be clear, the fourth line calculates the end-to-end delay by subtracting the packet creation time retrieved by the function op_pk_creation_time_get() from the current simulation time which is obtained from the function op_sim_time(). In the following line the value of the end-to-end delay is written in the global statistic ETE Delay through the state variable ete_delay. Finally the packet is destroyed in the last line; it is important to remember to destroy the packet after finishing processing it, because if you forget to do so the memory will be filled with garbage and sometimes causing the simulation to terminate.

13. Choose **Interfaces → Process Interfaces**. In the process interfaces window verify that the **begsim intrpt** attribute value is set to **enabled** – this will enable the begin simulation interrupt which will cause the process model to execute the init state and go to the idle state. If this attribute is not set to enabled, the init state will not be executed at the beginning of the simulation and will wait until the first interrupt occurs, which will be a stream interrupt in our case caused by the first packet arriving from the generator to the processor. This would cause the packet to be dropped. Click **OK** to close the dialog box.

14. The final step is to compile the process model. To do so click on **Compile Process Model**  button on the toolbar. You will be prompted to enter a name for this process model; set it to

grp#_tut_node_process. When the compilation completes close the compilation dialog box, and close the process model editor.

15. Return back to the node editor. Right-click on the processor module and choose **Edit Attributes**. Change the value of the **process model** attribute to **grp#_tut_node_process**. Verify that the **Node address** attribute is **promoted** and the **begsim intrpt** is **enabled**. Click **OK** to close the dialog box then choose **File → Save** to save the changes to the node model.

This completes creating the node and process models for the peripheral nodes.

5) Creating Node and Process Models for the Switch Node

As we did for the peripheral node, we will start with creating the node model then the process model for the switch node. It should be easier by now.

Node Model

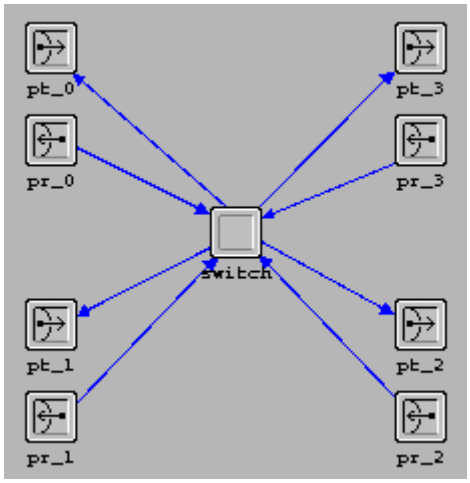
The functions of the switch node include:

1) Receiving packets from the four peripheral nodes
2) Processing packets to decide where to send them
3) Transmitting packets to the appropriate target

To fulfill these duties our node model should consist of 4 point-to-point receivers, 4 point-to-point transmitters and a processing unit.

To create the node model for the switch node:

1. Choose **File → New**, select **Node Model** from the list and click **OK**. The node model editor appears.
2. Using what you have learned create the node model shown in the figure below:



It is better if you create the links in order (i.e. switch→pt_0, then switch→pt_1, and so on, and the same for the receivers).

3. Select all transmitters and receivers. Change the channel **data rate** to **9600** and the **supported packet** format to **grp#_tut_pkt**. Do not forget to check the "**Apply changes to selected objects checkbox**."
4. Right-click on the switch and choose **Show Connectivity**, verify that it looks like this

```
stream    :  switch [0] -> pt_0 [0]
stream    :  switch [1] -> pt_1 [0]
stream    :  switch [2] -> pt_2 [0]
stream    :  switch [3] -> pt_3 [0]
stream    :  pr_0 [0] -> switch [0]
stream    :  pr_1 [0] -> switch [1]
stream    :  pr_2 [0] -> switch [2]
stream    :  pr_3 [0] -> switch [3]
```

If you need to change the stream assignment right click on the packet stream and choose **Edit Attributes**, change the **src stream** and **dest stream** attributes as desired. You will not need to do so if you made the connections in order as stated in step 2.

5. Set the supported node types to **fixed** only (as you did in part 4 step 12), from **Interfaces → Node Interfaces**.
6. Choose **File → Save** and name the model **grp#_tut_switch**. Do not close the Node Model Editor.

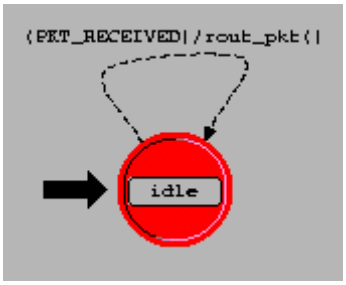
Process Model

The only operation that takes place in the switch is to decide to which target should the received packet be forwarded. Thus, our state transition diagram can consist of a single state that has one transition conditioned on receiving a packet and occupied with a transition executive to do the decision making.

Ask yourselves, how will the switch know the output stream the received packet should be forwarded to? In other words, how will it know which node is on the other end of the connection? (Hint: this will become clear at step 7 of building the network.)

To create the process model for the switch node:

1. Choose **File → New**, select **Process Model** from the list and click **OK**.
2. Place one state in the editors field and change its name to **idle**.
3. Draw a transition from the idle state and back to itself. Change the condition on this transition to PKT_RECEIVED and the executive to rout_pkt(). Your state transition diagram should look like this:



4. Now to define the transition macro, open the Header Block and write:

```
#define PKT_RECEIVED (op_intrpt_type() == OPC_INTRPT_STRM)
```

Save the Header Block.

This definition is good enough because we will treat all the stream interrupts in the same way, not like we did in the node where we had to distinguish between packets coming from the generator and packets coming from the receiver.

5. To define the function rout_pkt() open the Function Block, and type the following:

```
void rout_pkt()
{
int dest_address;
Packet * pkptr;
pkptr = op_pk_get(op_intrpt_strm ());
op_pk_nfd_get_int32 (pkptr, "dest_address", &dest_address);
op_pk_send (pkptr, dest_address);
}
```

Save the Function Block.

6. Save the Process Model by choosing **File → Save** and name it **grp#_tut_switch_process**.
7. Compile  the Process Model and close the Process Model Editor.

Now that we are done with the process model all that is left is to assign this process model to the node model. To do so

1. Return to the Node Model editor.
2. Right-click on the switch module, and choose **Edit Attributes**. For the process model attribute change its value to **grp#_tut_switch_process** and set the **begsim intrpt** attribute to **enabled**. Click **OK**.
3. Save the node model.

This completes creating the node and process models for the switch node.

6) Building the Network

The Project Editor is the main staging area for creating a network simulation. From this editor, you can build a network model using models from the standard library (or using models created by users), choose statistics about the network, run a simulation, and view the results.

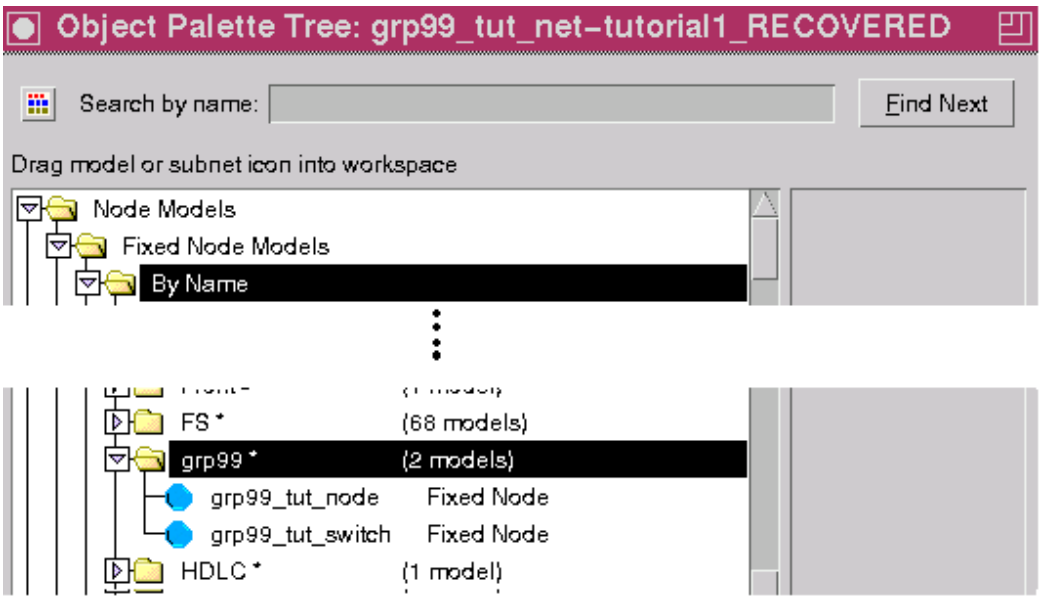
To create the network:

1. Choose **File → New**, then select **Project** and click **OK**.
2. Name the project as **grp#_tut_net** and the scenario as **tutorial**.
3. In the startup wizard click **Quit**, the project editor opens.
4. Open the **object palette** by clicking on  in the toolbar.
5. From the object palette place a subnet 

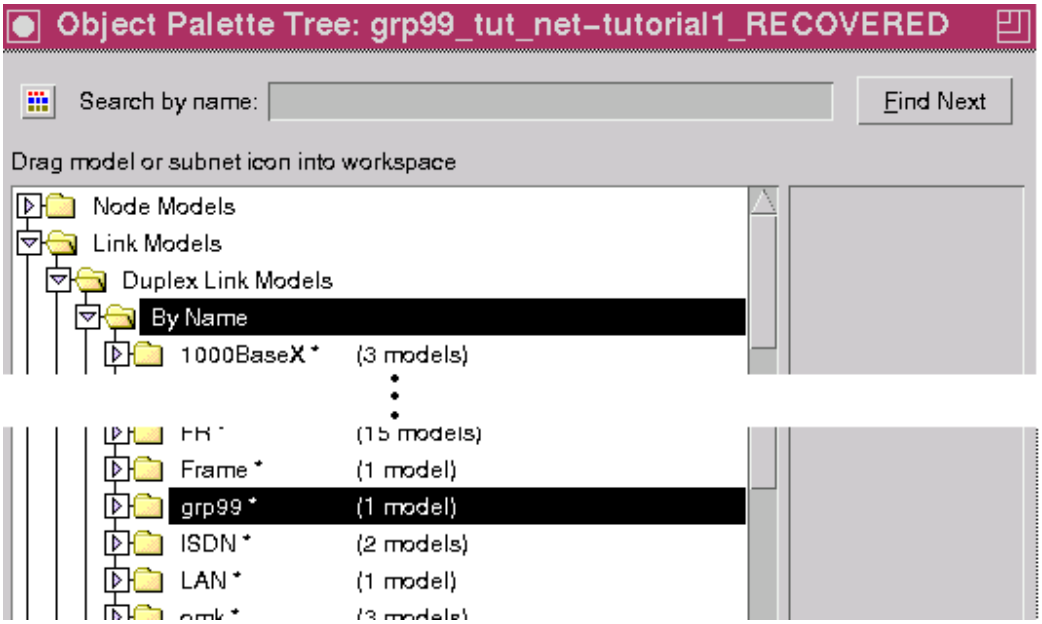
in the project editor field; left-click on the subnet, left-click in the project editor field (the world map) and finally right-click to finish.
6. Double-click on the subnet to specify its components. You will be adding four peripheral nodes and one switch node to your subnet, as specified in the next step.

NOTE:	You will find the components that you created and that you will use to build the network in the locations as shown in the figures below.

First, the nodes:



Then the link:



7. Inside the subnet, first place four peripheral nodes **(grp#_tut_node)** then place one switch node **(grp#_tut_switch)**. When you are finished do the following:
    a. Change the **name** of the switch node to **switch**
    b. Using the link you created, connect the four nodes with the switch *in order* (i.e. node_0 then node_1 and so on)
    c. Verify that the links are connected by choosing the Verify Links button  from the toolbar. The status will be displayed in the window's status bar.
    d. For each node, right-click and choose **Edit Attributes**. Change the **process.Node Address** attribute to the **node number** (e.g. make it 0 for node_0, 1 for node_1, etc)
    e. Select all peripheral nodes, right-click and choose **Edit Attributes**, select the checkbox "**Apply Changes to Selected Objects**," and click in the **Value** column for the **generator.Packet Interarrival Time** attribute. A dialog box appears. Verify that the **Distribution Name = constant** and change the **Mean Outcome = 4**. Click **OK** to close

the dialog box. Click **OK** again to close the attributes dialog box and say **Yes** to the warning.

7) Running the Simulation

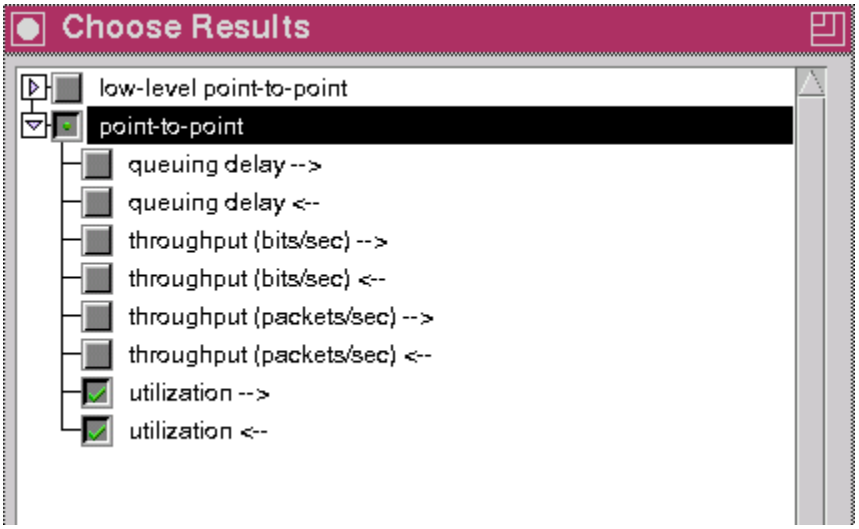
Before running any simulation in OPNET, we must first choose the statistics that we want to measure. In this experiment we are interested in the end-to-end delay and the link utilization experienced by all the peripheral nodes. To select the ETE Delay statistic:

1. Right-click in any blank area in the editor field and select **Choose Individual DES Statistics**.
2. Expand **Global statistics** and choose **ETE Delay**

3. Click **OK**.

To select the link utilization statistic:

1. Right-click on any link and select **Choose Individual DES Statistics**.
2. Expand the **point-to-point** group and choose the **utilization** statistics (**both of them**).



3. Click **OK**.

We want to compare the network performance in two cases (for two different interarrival time values). We can run the simulation twice changing the packet interarrival time in each execution, or we can create two scenarios with different settings.

To create another scenario:

1. Choose **Scenarios → Duplicate scenarios**, click **OK** to name the new scenario as tutorial1.
2. Change the **interarrival time** (as you did in part 6 step 7e) for all peripheral nodes to 8.

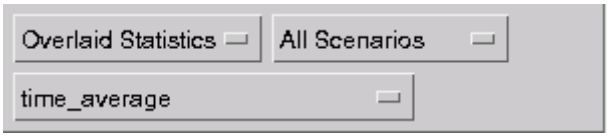
Finally we can now run the simulation. (Save your models before you run the simulation.) Because we have two scenarios it easier to run the simulation as follows:

1. Choose **Scenarios → Manage scenarios**.
2. In the dialog box set the **Results** column to **<collect>** for both scenarios and verify that **Sim Duration** equals 1 hour.
3. Click **OK**. The simulation will start executing.

A new window will open and the simulation will run 1 to 2 minutes. When it finishes, the close button will be enabled; click on it.

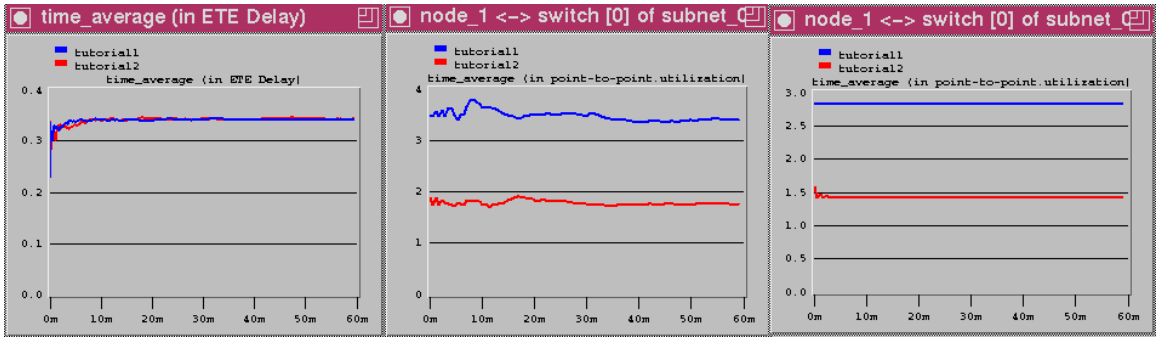
To view the results:

1. Click on the **View Results** button  in the toolbar.
2. In the lower right quarter of the view results window change the view parameters as follows:
   a. Change **This Scenario** to **All Scenarios**
   b. Change **Stacked Statistics** to **Overlaid Statistics**
   c. Change **As is** to **time_average**



3. In the upper left quarter of the view results window choose the statistics that we collected (once at a time) and click Show each time (i.e. choose Global Statistics → ETE Delay and click Show, then choose Object Statistics → subnet_0 → node_1 <-> switch[0] → point-to-point → utilization <-- and click Show, and do the same for utilization-->)

Your results should look close to the results below:

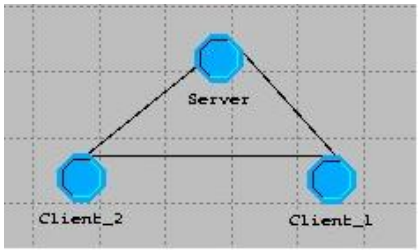


Exercise

Overview

In this exercise you will use what you have learned in the OPNET tutorial to create a small client-server network, which consists of two clients and one server. The network is a simple one, in which the server will be always sending packets to clients, and will not receive any from them. In this case, no initialization or handshaking of any type is required.

The network will be connected as a unidirectional ring, as shown here:



In this network, the server will always be sending packets to either one of the two clients with equal probabilities. A client in turn will check the destination address of the packet to see if it is directed for him or not. If it is, then he will just destroy the packet. If it is not, then he will forward it on the ring to the next client.

In this exercise, you will be using the same objects you just created in the OPNET Tutorial with some minor modifications. Specifically, you should make two modified versions of the peripheral node model (*not* the switch node model) you already have: one for the server node and the other for the client nodes.

Because we want to create a unidirectional ring, each node must have two ports: one always used for transmitting and the other always used for receiving. Thus, since we want to use the same full duplex point-to-point links that we created, each port must consist of a transmitter-receiver pair, although you will be only using the transmitter in one of the ports, and the receiver in the other.

Server Node

The server node is responsible for the following functions:

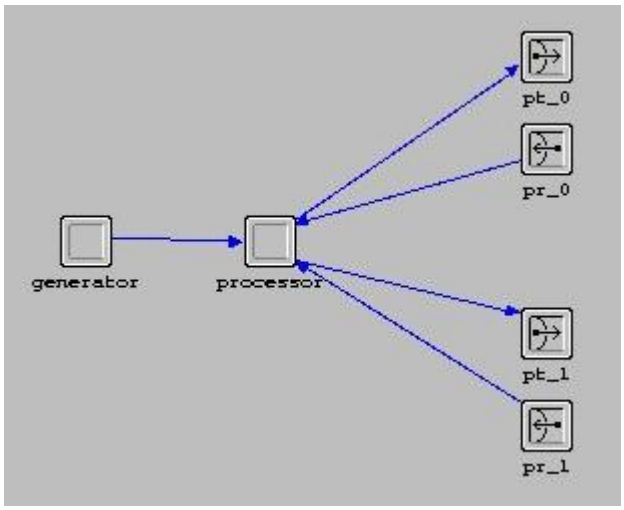
1) Generating packets
2) Filling the appropriate fields in packets
3) Sending packets in the forward direction on the ring
4) Making sure that all packets were received by clients and stop the simulation if not, by checking that no packets are received from the

To operate correctly, the server node should consist of:

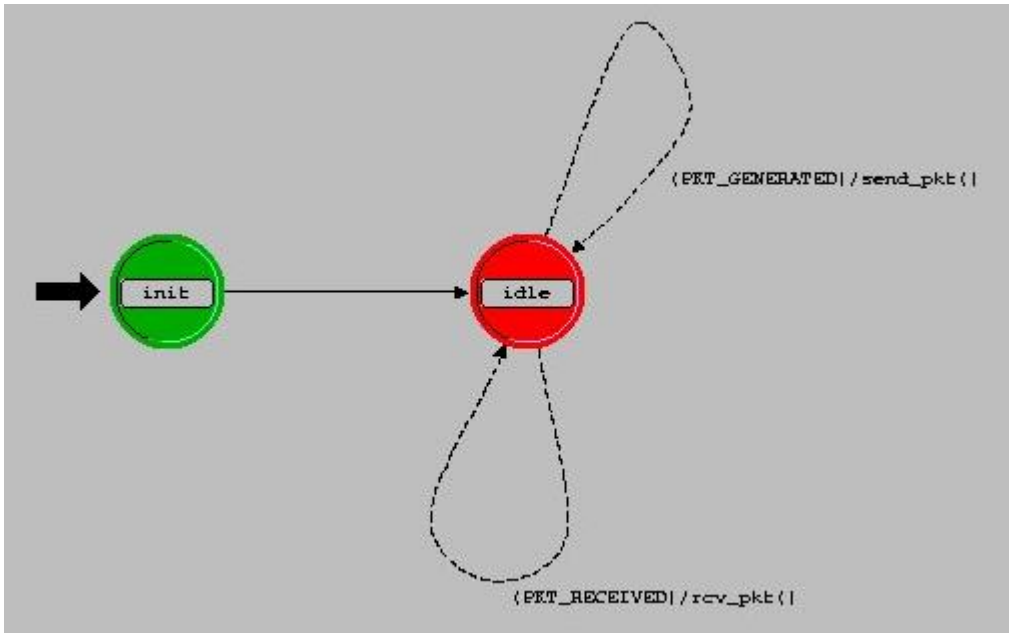
1) A packet generator (for function 1)
2) A processing unit (for functions 2 and 4)
3) A transmitter-receiver pair (for function 3)
4) A transmitter-receiver pair (for function 4)

The receiver in the server node will not be used at all if the destination address in the generated packets and the addresses given to client nodes were correctly assigned. That is, since there are only two client nodes, the server should choose from only two addresses when filling the “dest_address” field in the generated packet.

Node Model



Process Model



State Variables

Type	Name
int	src_addr
Distribution *	dist_addr
Stathandle	ete_delay
int	out_port

Header Block

```
//Define stream numbers

#define GNR_STRM 0
#define RCV_STRM0 1
#define TX_STRM0 0
#define RCV_STRM1 2
#define TX_STRM1 1


//Define transition macros

#define PKT_GENERATED (op_intrpt_type()==OPC_INTRPT_STRM &&
op_intrpt_strm()==GNR_STRM)
#define PKT_RECEIVED (op_intrpt_type()==OPC_INTRPT_STRM &&
(op_intrpt_strm()==RCV_STRM0 || op_intrpt_strm()==RCV_STRM1))
```

Init State

```
op_ima_obj_attr_get(op_id_self(), "Node Address", &src_addr);
dist_addr = op_dist_load("uniform_int", 1, 2);
ete_delay = op_stat_reg("ETE Delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
out_port = TX_STRM1;}
```

Client Node

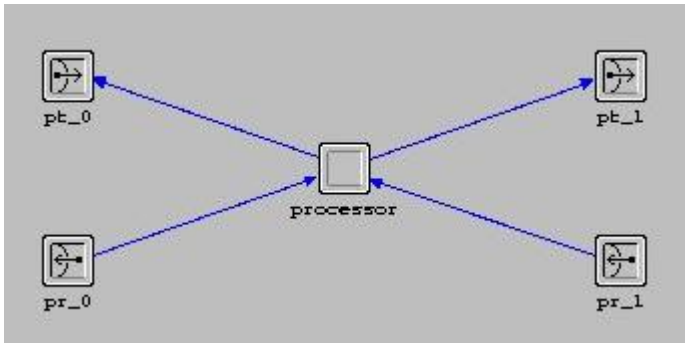
The client node is responsible for the following functions:

1) Receiving packets from previous node on the ring
2) Deciding if the received packet is directed for him or not
3) Forward the packet to the next node on the ring, if the packet was not for him

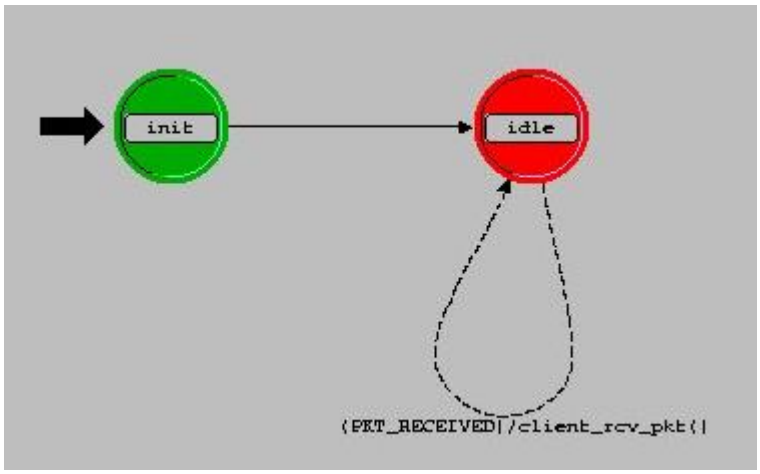
The client node should consist of:

1) A transmitter-receiver pair (for function 1)
2) A processing unit (for function 2)
3) A transmitter-receiver pair (for function 3)

Node Model



Process Model



State Variables

Type	Name
int	src_addr
Distribution *	dist_addr
Stathandle	ete_delay
int	out_port

Header Block

```
//Define stream numbers
#define RCV_STRM0 0
#define TX_STRM0 0
#define RCV_STRM1 1
#define TX_STRM1 1

//Define transition macros
#define PKT_RECEIVED (op_intrpt_type()==OPC_INTRPT_STRM && (op_intrpt_strm()==RCV_STRM0
|| op_intrpt_strm()==RCV_STRM1))
```

Init State

```
op_ima_obj_attr_get(op_id_self(), "Node Address", &src_addr);
dist_addr = op_dist_load("uniform_int", 0, 3);
ete_delay = op_stat_reg("ETE Delay", OPC_STAT_INDEX_NONE, OPC_STAT_GLOBAL);
```

Deliverables

For this exercise, you will need to write the code for the Server and Client Function Blocks and then demonstrate your simulation to the TA.

You can verify your work by checking the throughput on the three links, where it should be zero on the link between Client_2 and the Server. The throughput on the link between Client_1 and Client_2 should be approximately half that on the link between the Server and Client_1. The chart below is what you should demonstrate, and shows the throughput on the three links (averaged):

Server => Client_1

Client_1 => Client_2
Client_2 => Server