Basic Processes

Overview

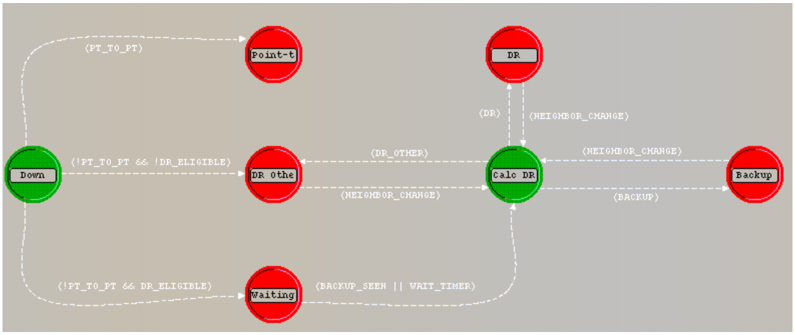
In OPNET, a network is made up of individual nodes, and a node is made of modules. The **process model** defines the behavior of a module. By understanding process models, you can build modules and combine them to build nodes to your exact specifications.

In this lesson, you will

- Create process and node models
- Define variables, macros, and transitions
- Run a simulation
- Analyze simulation results

This lesson shows how to build a module that counts the packets it receives then writes that number to a statistic that can be graphed. For each received packet, the process model increments the value of a variable and records the variable.

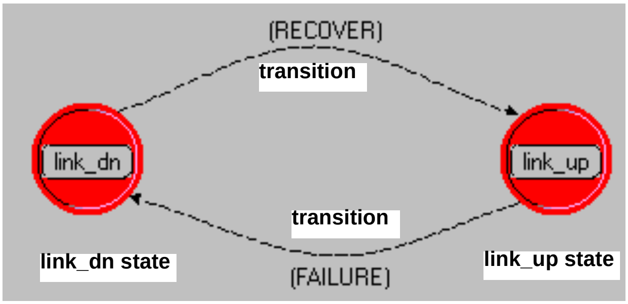
Example Process Model



A process model is a finite state machine (FSM). It represents the logic and behavior of a module. An FSM defines the states of the module and the criteria for changing the states.

OPNET uses the finite state machine (FSM) to implement the behavior of a module. FSMs use **states** and **tranistions** to determine what actions the module can take in response to an event.

States and Transitions



- **State**—The condition of a module. For example, a module may be waiting for a link to recover.
- **Transition**—A change of state in response to an event.

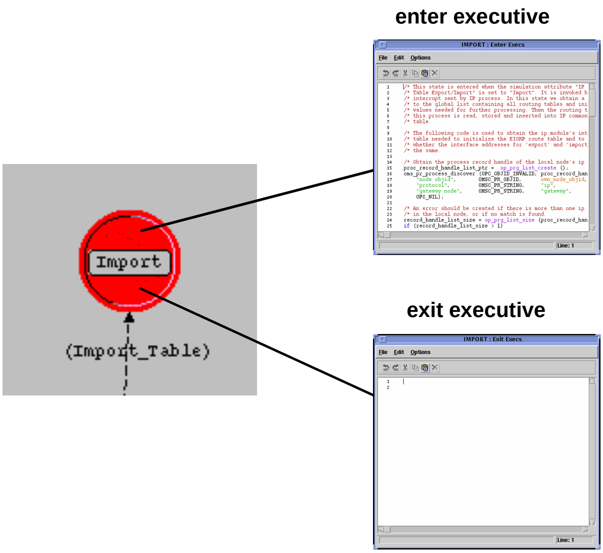
You can use an FSM to control module behavior because OPNET lets you attach fragments of C/C++ code to states and transitions.

OPNET allows you to attach fragments of C/C++ code to each part of an FSM. This code, augmented by OPNET-specific functions, is called Proto-C.

The three primary places to use Proto-C are as follows:

- **Enter Executive**—code that is executed when the module moves into a state
- **Exit Executive**—code that is executed when the module leaves a state
- **Transition Executive**—code that is executed in response to a specific event

Enter and Exit Executives of a State



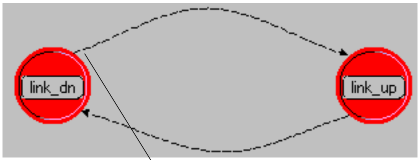
```
/* This state is entered when the simulation attribute "IP
/* Table Export/Import" is set to "Import". It is invoked b
/* interrupt sent by IP process. In this state we obtain a
/* to the global list containing all routing tables and ini
/* values needed for further processing. Then the routing t
/* this process is read, stored and inserted into IP common
/* table.

/* The following code is used to obtain the ip module's int
/* table needed to initialize the EIGRP route table and to
/* whether the interface addresses for 'export' and 'import
/* the same.

/* Obtain the process record handle of the local node's ip
proc_record_handle_list_ptr =  op_prg_list_create ();
oms_pr_process_discover (OPC_OBJID_INVALID, proc_record_han
    "node objid",       OMSC_PR_OBJID,      own_node_objid,
    "protocol",         OMSC_PR_STRING,     "ip",
    "gateway node",     OMSC_PR_STRING,     "gateway",
    OPC_NIL);

/* An error should be created if there is more than one ip
/* in the local node, or if no match is found.
record_handle_list_size = op_prg_list_size (proc_record_han
if (record_handle_list_size > 1)
```

OPNET simulations are made up of events. Process models respond to events and can schedule new ones.

When an event occurs that affects a module, the Simulation Kernel passes control to the module's process model via an interrupt. The process model responds to the event, changing state and executing related code, and then returns control to the Simulation Kernel (invocation 1).

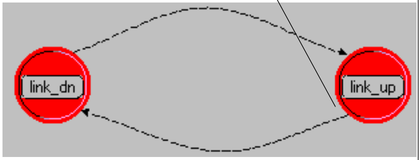
The next time OPNET invokes the process model, it determines the state in which it was left (invocation 2), responds to the new event, and passes control back to the Simulation Kernel.

Which State is Active?



Invocation 1 starts here and changes to the link_up state

Invocation 2 starts in the link_up state



When a process model enters a state, it executes the enter executives and then, if the state is unforced (as above), the process model stops execution and returns control to the simulation. We’ll discuss forced and unforced states in more detail later.

Designing the Model

The packet-counting process model you will build contains three states: an initializing state, an idle state, and an arrival state.

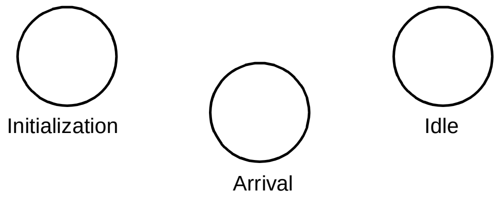
When a packet arrives at the module containing your process model, the model must increment a counter then dispose of the packet.

Therefore, the module has two primary states that

1) **Wait** for a packet to arrive
2) **Process** the packet after it arrives

You also need an initialization state that sets the appropriate variables to zero. The process model will have the states shown in the following figure.

The Three States Required for the Model

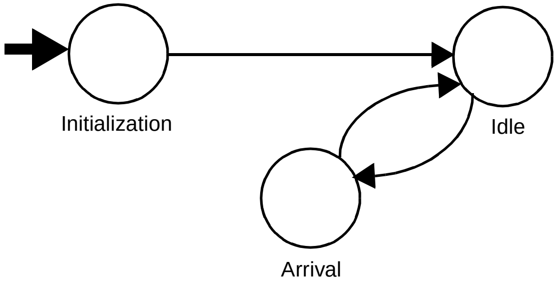


The packet-counting model contains three transitions: initialization-to-idle, idle-to-arrival, and arrival-to-idle.

The first time the process model is invoked (in this case, at the beginning of the simulation), it begins in the initialization state. After initialization—which sets the packet-counter variable to zero—the process model transitions to the idle state and waits for the first packet to arrive.

The process model is activated again and transitions to the arrival state when a packet arrives. The arrival state increments the packet-counter variable and destroys the packet. Without pausing, the process model then transitions back to the idle state to wait for the next packet.

Three Transitions are Required



Implementing the Process Model

The first step in implementing the process model is to open the Process Editor and place the model's three states in the workspace.

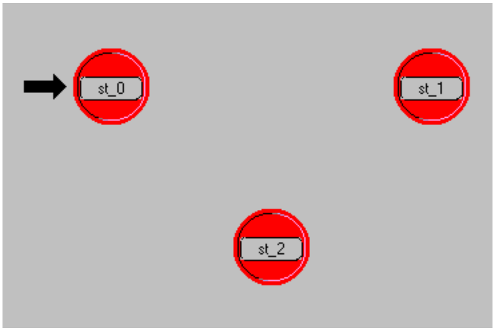
1 Start OPNET if it is not already running.

2 Choose **File > New...,** then select **Process Model** from the pull-down menu. Click **OK**.

3 Click the **Create State** action button and place three states in the workspace as shown.

Placing Three States

Create State action button



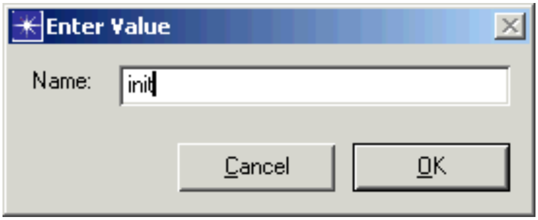
Notice that the first state you create is automatically the initial state; this is indicated by a heavy arrow.

Give each state a unique name that describes its function. You will use the name **init** for the initialization state, **idle** for the idle state, and **arrival** for the arrival state.

1 Right-click on the initial state (the one with the heavy arrow) and select **Set Name** from the Object pop-up menu.

2 Name the state **init** and press **Return**.

Naming the State



3 Repeat the procedure with the other two states and name them as follows:

— **st_1: idle**

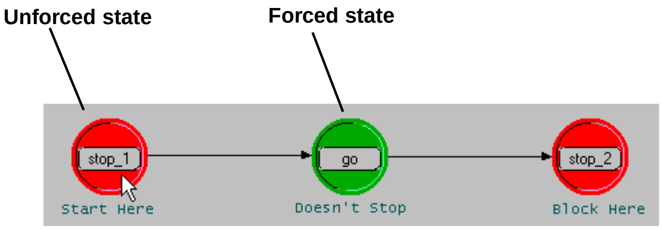
— **st_2: arrival**

An unforced (red) state is one that returns control of the simulation to the Simulation Kernel after executing its enter executives. A forced (green) state is one that does not return control, but instead immediately executes the exit executives and transitions to another state.

A newly-created state is, by default, unforced: after executing its enter executives, the process model blocks (that is, stops execution and returns control to the Simulation Kernel). The next time the process model is invoked, execution begins again from the state in which it last blocked.

A forced state does not stop after the enter executives. Execution proceeds to the exit executives and transitions to the next state.

Forced and Unforced States



Two states in the process model you are building are forced states. The **init** state is forced because it can proceed directly to the **idle** state after initializing variables. The **arrival** state is forced because the process model should return to the **idle** state after counting and destroying each packet.

1 Change the **init** state to a forced state by right-clicking on it and selecting **Make State Forced** from the Object pop-up menu.

➥ The **init** state becomes green.

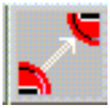
2 Repeat step 1 for the **arrival** state so that it is also a forced state.

Create State Transitions

The next step in building the process models is to create the transitions between the states. There are two types of transitions, unconditional and conditional. When a transition is conditional, the transition must evaluate to true before control passes from the source state to the destination state.

1 Click the **Create Transition** action button.

Create Transition Action Button

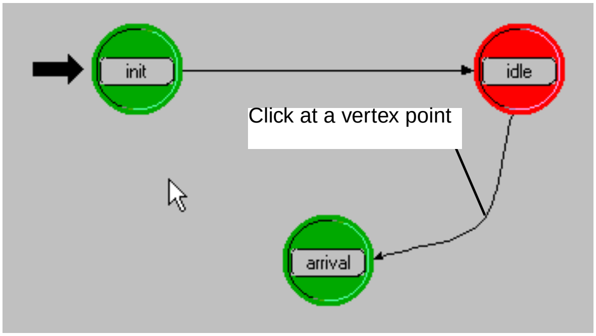


2 Draw the transition by first clicking on the **init** state and then clicking on the **idle** state.

➥ The arrow of the transition points from the source state to the destination state. The transition appears as a solid line; this indicates that the transition is unconditional.

3 Draw a curved transition by clicking first on the **idle** state, then at a vertex point between the **idle** and **arrival** states, and finally on the **arrival** state.

Drawing Transitions

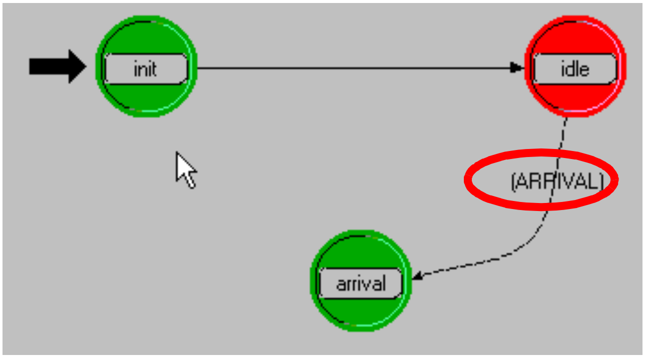


4 Right-click to stop drawing transitions.

To create a conditional transition, you place a value in the **condition** attribute of the transition. You must make the transition between **idle** and **arrival** conditional because the process model should transition to the **arrival** state only when a packet arrives and not in response to any other event.

1. Right-click on the transition between the **idle** and **arrival** states and select **Edit Attributes** from the pop-up menu.
2. Change the value of the **condition** attribute to **ARRIVAL** (be sure to use all capital letters), press **Return**, then click **OK**.
3. Close the Attributes dialog box.

“condition” Attribute has Value of “arrival”



➥ The transition is now a dashed line and has a label with the name of the condition. We will define the **ARRIVAL** condition macro later to mean that the transition will be used only when a packet arrives and the process model is in the idle state.

There are two more transitions to implement in the process model. The first is an unconditional transition from the **arrival** state to the **idle** state, and the second is a conditional transition from the **idle** state to itself.

1 Click on the **Create Transition** action button and draw a transition from **arrival** to **idle**.

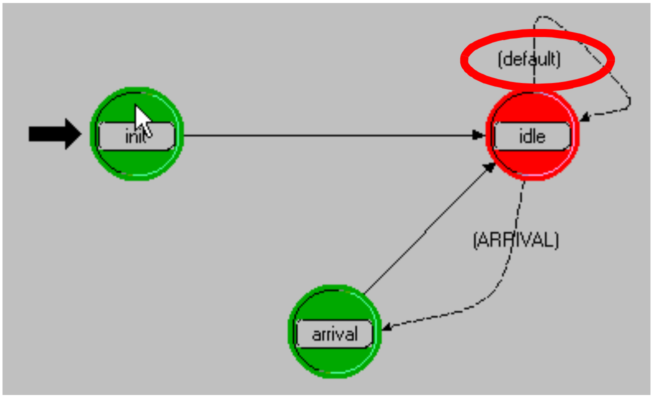
2 Draw a transition from the **idle** state back to itself, as shown in the following diagram.. Hint: click on two points in the workspace while drawing the transition before bringing it back to the **idle** state.

3 Right-click to stop drawing transitions.

4 Right-click on the transition from **idle** back to itself to bring up its Attributes dialog box.

5 Change the value of the **condition** attribute to **default** (be sure to use all lower-case letters), press **Return**, then click **OK**.

“condition” Attribute has Value of “default”



The Simulation Kernel operates by maintaining an event list for the entire simulation. An event that reaches the top of that list becomes an interrupt and is delivered to the correct place in the simulation (often to a specific module).

You may be wondering why you included a transition from idle back to itself and named that transition **default**. As the simulation executes, the Simulation Kernel manages a list of events to take place. As each event reaches the top (or head) of that list, it becomes an interrupt. Interrupts are often delivered to specific modules, and this occurrence is what activates the module's process model.

When the process model that you are building is in the **idle** state, it will transition to the **arrival** state if the **ARRIVAL** condition is true. You will shortly define **ARRIVAL** to evaluate to true if the interrupt delivered to the module is a packet arrival interrupt. If it is a different type of interrupt, however, there must be a transition that the process model can follow. The **default** transition handles these different interrupt types.

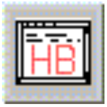
Define Conditions and Variables

In Proto-C, macros often replace more complicated expressions in transition conditions and executives. The use of macros saves space and simplifies the task of interpreting an FSM diagram. You specify macros with the C preprocessor **#define** statement and usually place them in the process model header block. The header block may also contain **#include** statements, **struct** and **typedef** definitions, **extern** and global variable declarations, function declarations, and C-style comments.

The next step is to define the **ARRIVAL** macro.

1 Click the **Edit Header Block** action button.

Edit Header Block Action Button



➥ An edit pad appears.

2 Enter the following code into the edit pad:

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

3 Choose **File > Save** from the edit pad menu to save the changes and close the pad.

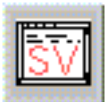
The **ARRIVAL** condition defined above tests if the current interrupt (which caused the FSM to awaken and execute) occurred due to an arriving packet. It compares the value returned by the Kernel Procedure **op_intrpt_type()** with the OPNET constant of **OPC_INTRPT_STRM**. If the comparison evaluates to true, this indicates that the interrupt is due to a packet arriving on an input stream.

You can declare variables in two places. Variables declared in the **temporary variables block** do not retain their values between invocations of the FSM. Variables declared in the **state variables block** retain their values from invocation to invocation.

The next step is to declare two state variables. One will store the value of the packet count; the other is the “handle” for the local statistic used to analyze the count.

1 Click the **Edit State Variables** action button.

Edit State Variables Action Button



2 Enter the following values into the dialog box:

Values for State Variables Dialog Box

Type	Name	Comments
int	pk_count	Counts total packets
Stathandle	pk_cnt_stathandle	Statistic to record packet count

Clicking on the value field for **Comments** opens an edit pad. After you type the comment, Choose **File > Save**.

3 Click the **OK** button to save the changes and close the State Variables dialog box.

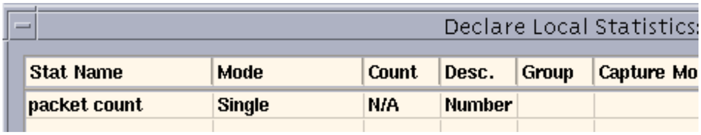
Statistics save values of interest for later analysis. When creating a statistic, you must declare that statistic in the process model that records it. You will create one statistic for this process.

1 Choose **Interfaces > Local Statistics**.

2 Enter **packet count** as the first **Stat Name**.

➥ The mode is automatically set to **Single**.

Declare Local Statistics Dialog Box



Stat Name	Mode	Count	Desc.	Group	Capture Mo
packet count	Single	N/A	Number		

3 Click on the **Desc.** for **packet count**.

➥ An edit pad appears.

4 Enter a short description for the statistic: **Number of packets received.**

5 Choose **File > Save** to save the changes and close the pad.

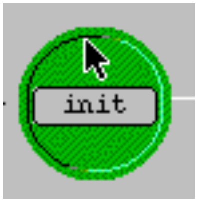
6 Close the Declare Local Statistics dialog box by clicking on the **OK** button.

Create State Executives

The next step is to create the state executives needed in the FSM. In the first state, **init**, the executives set the packet count to 0 and associate the name of the local statistic with its statistic handle. Although the **enter executives** and **exit executives** of forced states are executed without interruption, standard practice is to place all forced state executives in the **enter executives** block.

1 Double-click on the top half of the **init** state.

Open the Enter Executives of the init State



➥ The Enter Execs edit pad appears.

2 Enter the following code to initialize the state variable **pk_count**:

```
pk_count = 0;
pk_cnt_stathandle = op_stat_reg ("packet count",
      OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
```

3 Choose **File > Save** to close the edit pad.

The second line of the preceding code registers the **packet count** statistic and sets up a handle for the **arrival** state to use when recording the number of packets received.

Earlier, the code necessary to implement the desired behavior was provided for you, along with an explanation of how the code works (the code is shown again).

```
pk_count = 0;
pk_cnt_stathandle = op_stat_reg (“packet count”,
   OPC_STAT_INDEX_NONE, OPC_STAT_LOCAL);
```

The **pk_count** and **pk_cnt_stathandle** variables are state variables you defined, but the **op_stat_reg()** function is an OPNET Simulation Kernel Procedure (KP). OPNET provides hundreds of built-in functions that you can use for a variety of purposes: anything from manipulating a queue to creating animations.

Because there are so many functions, it can be difficult to determine the most appropriate one for a particular task. However, OPNET provides a summary of the most frequently-used KPs. You will review this summary in the next step of this tutorial, when you write the code for the **enter execs** in the **arrival** state.

- Double-click on the top half of the **arrival** state.

  ➥ The **enter execs** edit pad appears.

In the **arrival** state, you first want to increment the packet count. Then you need to determine the packet stream for the current interrupt, then get the pointer to the packet so that you can destroy it. Finally, you want to write out the statistic.

How should you implement this?

The first part, incrementing the packet count, is simple — just increment the **pk_count** variable using the following code:

```
++pk_count;
```

Determining the packet stream for the current interrupt, getting the packet's pointer, and destroying the packet is a bit more complex, but can be done using some of the KPs. To determine which ones, look first at the list of most commonly used KPs, the Essential Kernel Procedures.

- Choose **Help > Essential Kernel Procedures**.

  ➥ A list of the most frequently-used KPs opens in an Acrobat Reader window.

If the Essential Kernel Procedures document replaces the tutorial, you can return to the tutorial by choosing **Document > Go To Previous Doc** from the Acrobat menu bar.

The first task, determining the packet stream for the current interrupt, has to do with processing an interrupt. Find the section called **Interrupt Processing**.

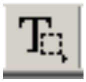
From the description, you determine that **op_intrpt_strm()** will return the necessary information.

The next two tasks, getting the packet's pointer and then using that pointer to destroy the packet, both involve packet manipulation. Look in the **Packet Generation and Processing** section. You can use **op_pk_get()** to determine the packet's pointer, and then use that return value (the pointer) to call **op_pk_destroy()** to destroy the packet.

You can type these function calls into the process model, or copy them from the Acrobat file. In the following section, we'll copy these function calls from the Acrobat file:

1 In the Acrobat menu, click on the Text Select tool button.

Text Select Tool Button



2 Copy the **op_pk_destroy()** function call:

2.1 Hold down the Shift key as you select the **op_pk_destroy()** function call in the Acrobat file.

2.2 Copy and paste the function call into the enter execs section of the **arrival** state, just after the line where you incremented the packet count.

2.3 Enter a semicolon at the end of the function call, so the line looks like this:

```
op_pk_destroy (pkptr);
```

3 Replace the **pkptr** argument with the following code

```
op_pk_get (op_intrpt_strm ())
```

so that the resulting line of code looks like this:

```
op_pk_destroy (op_pk_get (op_intrpt_strm ()));
```

The final task is to write out the value of the statistic. Refer to the **Statistic Recording** section for information on the **op_stat_write()** KP.

The final code for the enter exec of the arrival state should look like this:

```
++pk_count;
op_pk_destroy (op_pk_get (op_intrpt_strm ()));
op_stat_write (pk_cnt_stathandle, pk_count);
```

Finally, save the enter exec code:

Choose **File > Save** to close the edit pad.

The Essential Kernel Procedures are those that are most commonly used. Many additional KPs are available. You can display a description of any KP by choosing **Help > All Kernel Procedures** from the Process Editor menu.

In both KP help files, each KP name is a hypertext link to the complete documentation for that KP.

Edit Process Interfaces

To control the attributes visible at the node level, edit the Process Interfaces. You can set initial attribute values or hide attributes.

1. Choose **Interfaces > Process Interfaces**.

   ➥ The Process Interfaces dialog box appears.

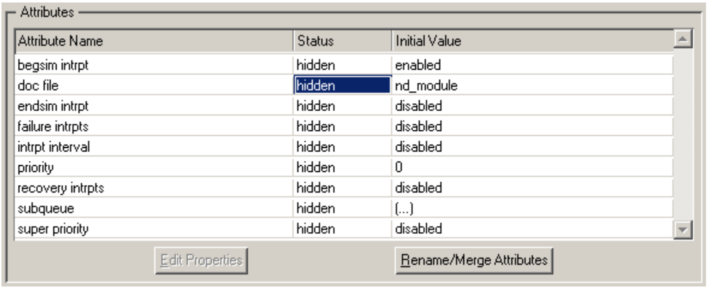
2. Change the **Initial Value** for the **begsim intrpt** attribute to **enabled**.

3 Verify that the **Initial Value** for each of the following attributes is set to **disabled**: **endsim intrpt**, **failure intrpts**, **intrpt interval**, **recovery intrpts**, and **super priority**.

4 Verify that the **Initial Value** of the **priority** attribute is **0**.

5 For all attributes, change the value of **Status** to hidden by left-clicking in the Status column and selecting **hidden**.

Changing the Value to "hidden"



Attribute Name	Status	Initial Value
begsim intrpt	hidden	enabled
doc file	hidden	nd_module
endsim intrpt	hidden	disabled
failure intrpts	hidden	disabled
intrpt interval	hidden	disabled
priority	hidden	0
recovery intrpts	hidden	disabled
subqueue	hidden	(...)
super priority	hidden	disabled

6 Click **OK** to close the Process Interfaces dialog box.

Compile the Model

All process models must be compiled before they can be used in a simulation. Compiling a model makes its object code available to a processor module’s **process model** attribute.

To compile a process model:

1 Click the **Compile Process Model** action button.

Compile Process Model Action Button

2 Save the process model as **<initials>_packet_count**.

➥ When the compile is successfully completed, OPNET beeps once and displays the following message in the message area:

```
Process model object file produced.
```

If you did not save the process model before compiling it, you will also see this message:

```
Wrote File: (<file name>).
```

3 Close the Process Editor.

Implementing the Node and Network Models

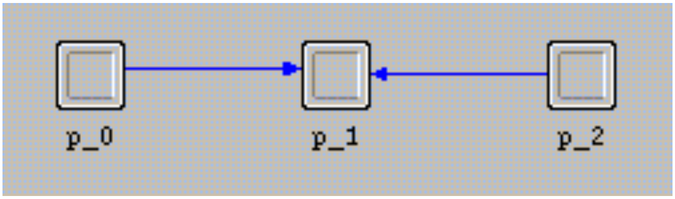
The node model in this lesson has two packet generator modules that send packets to a general processor module. The processor module uses the previously defined **packet_count** process to count and discard packets and to update an output statistic. Note that the process model could accept any number of incoming packet streams because it automatically determines the correct stream when an interrupt occurs.

Create the Node Model

First, create a node model that uses the **packet_count** process.

1. Choose **File > New...,** then select **Node Model** from the pull-down menu. Click **OK**.
2. Use the **Create Processor** action button to create three processor modules.
3. Use the **Create Packet Stream** action button to connect the modules with packet streams as shown. The model should resemble the one in the following diagram.

Initial Node Model



Now change the attributes of each module. The first processor module, p_0, will be a generator module:

1 Right-click on the icon and select **Edit Attributes** from the pop-up menu to open the module's Attributes dialog box.

2 Change the **name** attribute to **src1**.

3 Change the **process model** attribute to **simple_source**.

4 Close the dialog box.

The second processor module, p_1, will count packets:

1 Open the Attributes dialog box of the module.

2 Change the **name** attribute to **count**.

3 Change the **process model** attribute to **<initials>_packet_count**.

4 Close the dialog box.

Set up the third processor module to generate packets:

1 Open the Attributes dialog box of the module.

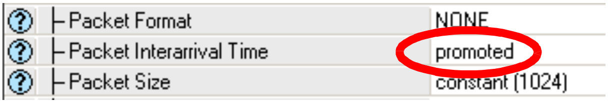
2 Change the **name** attribute to **src2**.

3 Change the **process model** attribute to **simple_source**.

4 Click on **Packet Interarrival Time** in the left column to highlight the Attribute name, then right-click and select **Promote Attribute to Higher Level** from the pop-up menu.

➥ The word **promoted** appears in the Value cell of the attribute.

Promoting the Attribute



Packet Format	NONE
Packet Interarrival Time	promoted
Packet Size	constant (1024)

5 Close the dialog box.

Both generators send packets to **count** at a mean interarrival rate of 1 packet/second. The first generator sends packets at a constant rate and the second generator sends packets at a variable rate, following a distribution you will specify.

Sometimes it is convenient or necessary to make a statistic available at the node level. A statistic promoted to the node level can be renamed and given a different description. Before saving the node model, promote the packet count statistic and rename it.

1. Choose **Interfaces > Node Statistics**.

2. Click the first field in the **Orig. Name** column, select **count.packet count** from the list that displays, and click **Promote**.

3. Change the **Prom. Name** to **node packet count**.

4. Change the **Desc.** field to: Number of packets received **at the node level**. Choose **File > Save** from the edit pad's menu to save the text and close the edit pad.

5. Close the **Statistic Promotion** dialog box by clicking the **OK** button.

6 Choose **File > Save** and save the node model as **<initials>_packet_count**. (The node and process models can have the same name because OPNET appends **.nd.m** to each node model and **.pr.m** to each process model.)

7 Close the Node Editor.

Create the Network Model

The network model for this lesson consists of a single node using the **packet_count** node model. After placing the node in the Project Editor workspace, you can specify statistics and animations to collect.

First, create a new project.

1 Choose **File > New...** and open a new project.

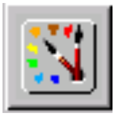
2 Name the project **<initials>_packet_count** and the scenario **constant**.

3 Choose **Quit** on the first screen of the Setup Wizard.

Second, create a custom object palette.

1 Open the object palette by clicking on the **object palette** action button.

Object Palette Action Button



2 Click the **Configure Palette...** button.

3 Click the **Clear** button in the Configure Palette dialog box to remove all but the default icons from the palette. The subnet icon remains. (If you have the Wireless module, the mobile and satellite subnet icons remain as well.)

4 Click the **Node Models** button and change the value of the **<initials>_packet_count** node model to **included**.

5 Click **OK** to close the Select Included Entries dialog box, and then click **OK** to close the Configure Palette dialog box.

6 At the prompt, click **OK** to save the custom model list with the default name.

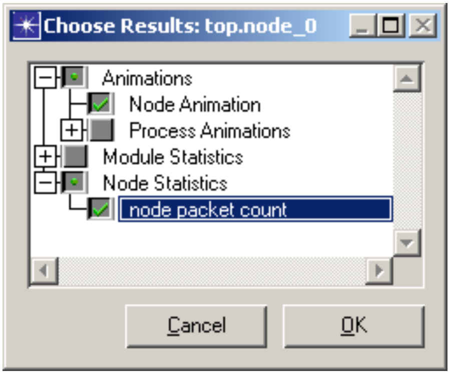
Next, place the node in the workspace and select the proper statistics for collection.

1 Place an **<initials>_packet_count** node in the workspace, then close the object palette.

2 Right-click on the node to open its Object pop-up menu, and choose **Choose Individual Statistics**.

➥ The Choose Results statistic browser opens.

Selecting Statistics



3 Collect the following statistics:

— **Animations > Node Animation**

— **Node Statistics > node packet count**

4 Right-click on **Node Statistics > node packet count** to open its Statistic pop-up menu, and choose **Record Statistic Animation**.

➥ A small “A” (for “animation”) appears in the check box next to **node packet count**.

5 Click **OK** to close the browser window.

In the preceding steps, you:

- Created an animation of packet flow within the node (**Animations > Node Animation**).
- Specified that the node packet count statistic be collected (**Node Statistics > node packet count**).
- Specified that the node packet count statistic be animated.

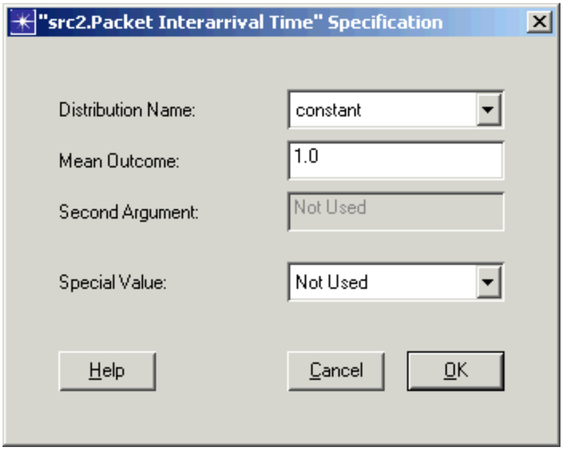
Recall that when building the node model, you changed the **interarrival pdf** attribute for the second generator module to **promoted** so that you could specify it later. You will now specify the distribution for that attribute.

1 Right-click on the node to open its Object pop-up menu, and choose **Edit Attributes**.

2 Click on **promoted** in the value column of **src2.Packet Interarrival Time**.

3 Verify that **Distribution Name** is **constant** and **Mean Outcome** is **1.0**.

Specify Distribution Dialog Box


"src2.Packet Interarrival Time" Specification

Distribution Name: constant

Mean Outcome: 1.0

Second Argument: Not Used

Special Value: Not Used

4 Click **OK** to close the Specification dialog box.

5 Click **OK** to close the Attributes dialog box.

To investigate the effect of different PDFs, you can create new scenarios based on this one. After simulating them, you can then compare the results. To create a new scenario and set the PDF:

1 Choose **Scenarios > Duplicate Scenario...**.

2 Name the new scenario **exponential**. Click **OK** to save the file.

3 Right-click on the node and choose **Edit Attributes** from the Object pop-up menu.

4 Click on the Value column of **src2.Packet Interarrival Time** to open its Specification dialog box.

5 Select **exponential** from the **Distribution Name** pull-down menu and set the **Mean Outcome** to **1.0**.

6 Click **OK** to close the Specification dialog box.

7 Click **OK** to close the Attributes dialog box.

8 Save the project with the default name.

Running the Simulation

You are now ready to configure and run the simulations. You can run both simulations with a single command from the Manage Scenarios dialog box.

To configure the simulations:

1 Choose **Simulation > Configure Discrete Event Simulation...**.

2 Set the following values in the Configure Discrete Event Simulation dialog box:

— **Duration**: **100 seconds**

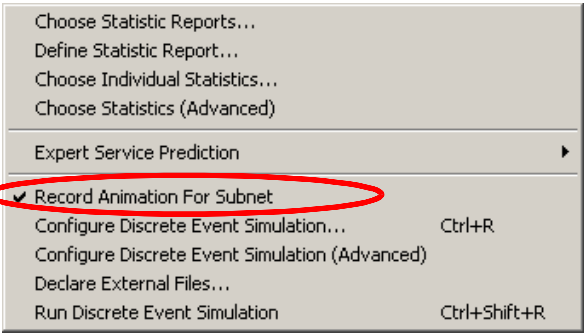
— **Seed**: **1471**

— **Values Per Statistic**: **100**

3 Close the Configure Discrete Event Simulation dialog box.

4 Choose **Simulation > Record Animation for Subnet.**

Record Animation for Subnet Item Checked



Choose Statistic Reports...
Define Statistic Report...
Choose Individual Statistics...
Choose Statistics (Advanced)
Expert Service Prediction
✓ Record Animation For Subnet
Configure Discrete Event Simulation... Ctrl+R
Configure Discrete Event Simulation (Advanced)
Declare External Files...
Run Discrete Event Simulation Ctrl+Shift+R

5 Switch to the **constant** scenario by choosing **Scenarios > Switch To Scenario > constant**.

6 Repeat steps 1–4 to configure the **constant** scenario.

To configure the repositories:

1 Choose **Edit > Preferences**.

2 Verify that the **repositories** preference is empty. Delete any entries. Click **OK** to close the dialog box.

To run the simulations:

1 Choose **Scenarios > Manage Scenarios...**.

2 Change the value of the **Results** column for each scenario from **uncollected** to **<collect>**.

3 Click **OK** to begin the simulation runs.

➥ OPNET runs two simulations, one for each scenario. The simulations should take less than one minute. If you have problems, see *"Troubleshooting Tutorial Simulations"*.

4 Close the Simulation Sequence dialog box.

Analyzing the Results

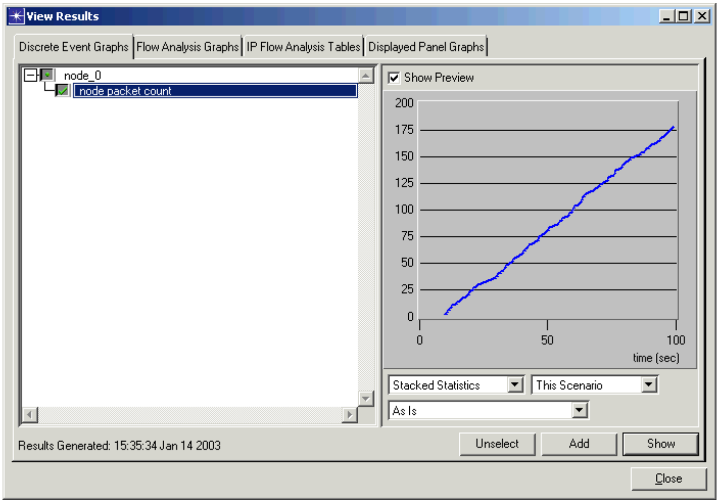
View results in the Project Editor.

1 If necessary, switch to the **exponential** scenario by choosing **Scenarios > Switch To > exponential**.

2 Right-click on **node_0**, then choose **View Results** from the Object pop-up menu.

➥ The View Results statistic browser opens, displaying the statistics that you can choose to view.

View Results Dialog Box


View Results

Discrete Event Graphs | Flow Analysis Graphs | IP Flow Analysis Tables | Displayed Panel Graphs

node_0
node packet count

☑ Show Preview

Stacked Statistics | This Scenario

As Is

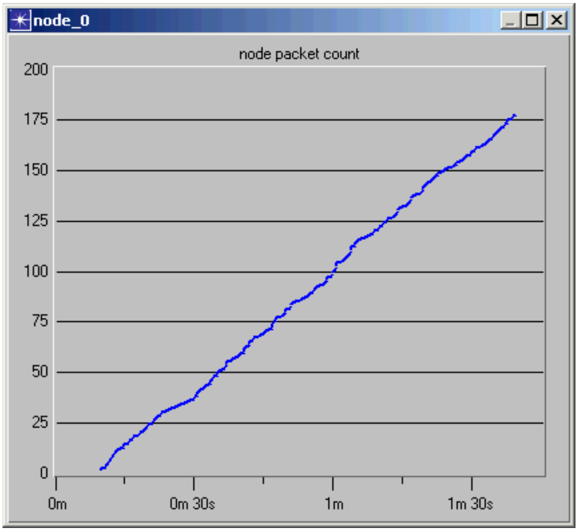
Results Generated: 15:35:34 Jan 14 2003

3 Place a check in the check box for **node_0 > node packet count**.

4 Click the **Show** button to display the graph.

The resulting graph shows the number of packets received by the count processor module over time during the simulation.

Number of Packets Received

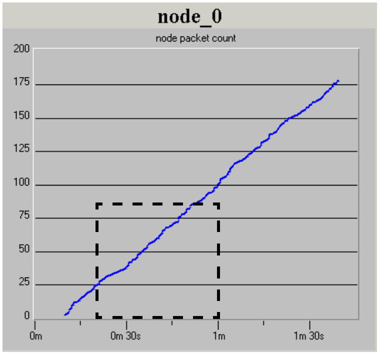


As illustrated in this graph, the results of the simulation indicate that the total number of packets received steadily increases over the simulation duration, though not at a perfectly constant rate. As expected, with two sources delivering packets each at a mean of 1 packet/second, the total number of packets received after 90 seconds of packet generation (packet generation commences after 10s of simulation time) is about 180.

A close-up visual examination of the graph will reveal more detail. To see this detail, magnify the graph.

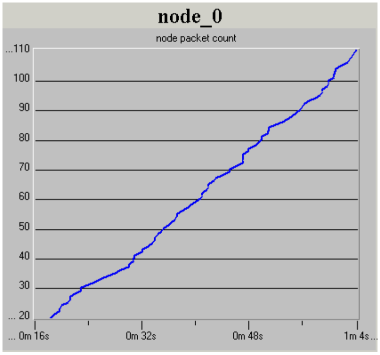
1 Drag the cursor along the graph, selecting a box that covers the trace from **20 seconds** to **1 minute** on the time axis.

Select the Area to Magnify



➥ A close-up of the selected graph area appears.

Magnified Area



You can contrast the graph of the packet count in the **exponential** scenario with the one in the **constant** scenario.

1 Close the current graph.

2 Close the View Results statistics browser.

3 Switch to the **constant** scenario by choosing **Scenarios > Switch To > constant**.

4 Choose **View Results** from the Object pop-up menu for **node_0**.

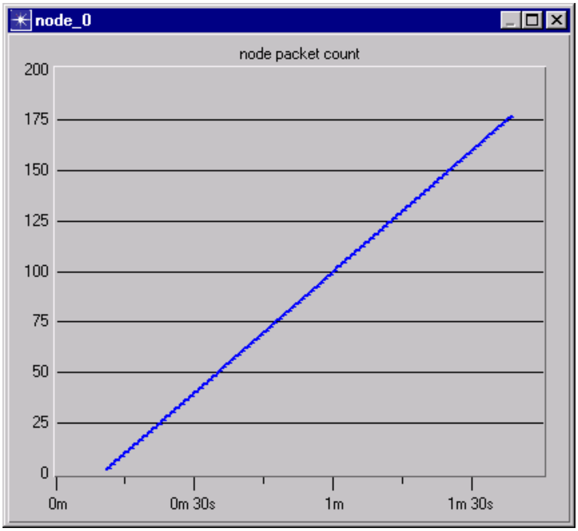
➥ The View Results statistic browser opens.

5 Display the graph for **node_0 > node packet count**.

6 Zoom in on a portion of this graph as you did for the last graph.

➥ Magnified (as shown in the following diagram), this graph reveals a step pattern that results from the two generators sending packets at the same constant arrival rate.

Zooming Reveals Step Pattern



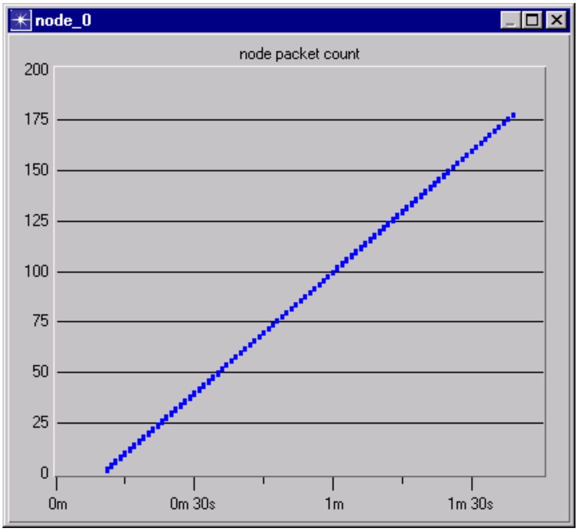
A discrete graph of this data will illustrate the true quantized property of the counting statistic and the underlying event-driven property of the packet arrivals.

1 Right-click on the graph to open the Graph pop-up menu.

2 Choose **Draw Style > Discrete** from the Graph pop-up menu.

The graph now displays the data as individual points.The discrete plotting style depicts only the discrete statistic values without connecting them.

Discrete Draw Style Applied



The discrete graph reveals that the Simulation Kernel repeatedly delivered two packets during each second of simulation time, each of which was counted separately. The graph also shows the event-scheduled nature of the OPNET Simulation Kernel. The simultaneous events (multiple packets arriving every second) caused the Simulation Kernel to advance simulated time once, but invoke the **packet_count** process twice. In turn, the **packet_count** process incremented the packet count statistic once each time it was invoked.

1 Close the graph dialog box.

Finally, you can view the animations created by OPNET during the simulation. The utility **op_vuanim** allows you to view statistic and packet animations.

1 Choose **Results > Play Animation**.

  ➥ **The Animation Viewer (op_vuanim)** opens, showing the statistics animation for the **constant** scenario.

2 To see the packet animation, choose **Windows > Animation Viewers > top.node_0.packet flow**.

3 To exit the Animation Viewer, choose **File > Exit.** Note that the Project Editor is still open.

4 To view the animations for the **exponential** scenario, change to the **exponential** scenario (**Scenarios > Switch to Scenario > exponential)**, then choose **Results > Play Animation**.

➥ The animation for the exponential scenario displays.

5 Exit the **Animation Viewer**. For more detailed information about **op_vuanim**, see the **Utility Programs** manual.

You have now completed the Basic Processes lesson of the tutorial. You should have a good understanding of how to control node behavior by creating custom process models.

The next lesson, Packet Switching, explores the creation of a packet switching network.

- To continue with the next lesson, save any work that has not yet been saved, then close all open editor windows in OPNET. Return to the main tutorial menu and choose **Packet Switching I** from the list of available lessons.
- To quit the tutorial, close Acrobat Reader.