CSMA/CD

Introduction

This tutorial presents detailed examples that illustrate the modeling and analysis of the well-known **Aloha** and **CSMA** channel access protocols. In this lesson, you will learn how to

- Construct more advanced protocols
- Design a simple channel interface to a multi-tap bus
- Execute parametric simulations
- Analyze the simulated results against theoretical predictions

You will build two models: an **Aloha** model and a **CSMA** model. Because it is the simplest of the channel access methods, we will build the **Aloha** model first.

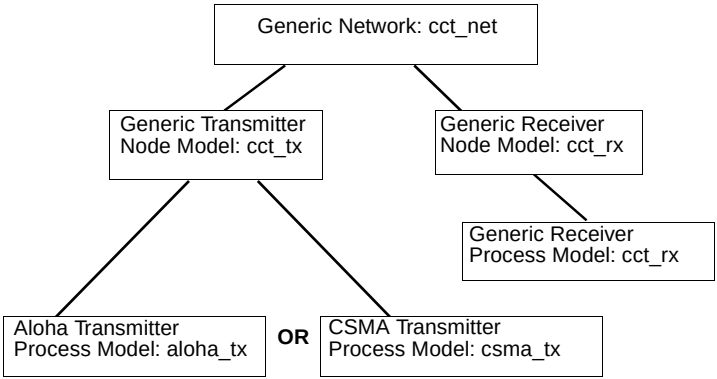
The main task is to design models that incorporate the **Aloha** random-channel-access method and the **1-persistent** carrier-sense-multiple-access (**CSMA**) method on a multi-tap bus link, where multiple nodes are connected through a shared channel. We will compare the performance of each method.

Getting Started

Before designing the models, you may be interested in an overview of the model hierarchy.

The design strategy for the **Aloha** and **CSMA** models is to employ the same network model. Both network models will use a common transmitter node model which sends packets, and a common receiver node model which performs network monitoring. By changing the process model attribute of the node models, new simulations using either **Aloha** or **CSMA** properties can be built quickly. The transmitter node process models will be unique, whereas the receiver node process model is generic and will remain unchanged.

Aloha and CSMA Modeling Hierarchy

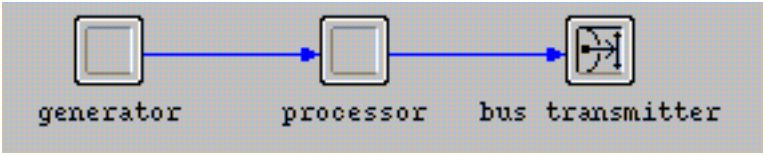


Designing the Generic Transmitter Node Model

In theory, the **Aloha** system could be modeled as just a simple source generator and a bus transmitter. However, by designing a more generalized model, you can reuse it later for the **CSMA** model.

The transmitter node must generate packets, process them, and send them on to the bus. This can be modeled using a simple source processor to generate packets, another processor to perform any necessary operations, and a bus transmitter to transmit the packets on the bus link.

Generic Transmitter Node Model



Bus transmitters also have internal queuing capability— they will issue all submitted packets onto the bus in FIFO order.

Designing the Aloha Transmitter Node Process Model

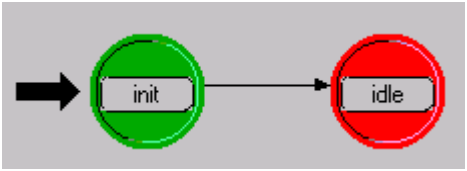
The Aloha transmitter process only has to receive packets from the generator and send them on to the transmitter.

The Aloha transmitter process has only one unforced state: waiting for the arrival of a packet from the generator. Since the generic transmitter node does not gather statistics, the **aloha_tx** process does not need to initialize or maintain state or global variables of its own. It does, however, need to retrieve a global attribute value that defines the number of generated packets. The transmitter process will retrieve this value once, before entering the main loop.

The process begins the simulation in a forced initialization state, then moves to an unforced idle state where it waits for packets to arrive.

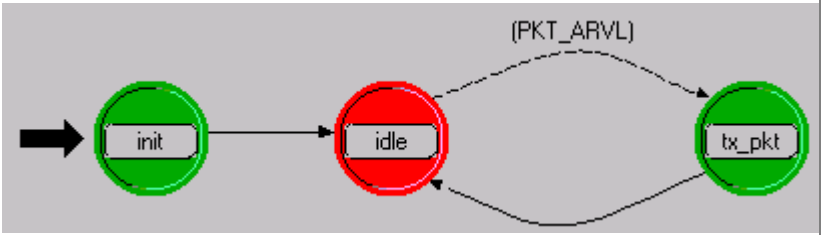
The process needs to be activated with a begin simulation interrupt so that when the simulation starts, the FSM executes the forced initialization state and then waits in the idle state, ready to transition when the first packet arrives.

Intermediate aloha_tx FSM



There is only one distinct event in the **aloha_tx** FSM, the arrival of a generated packet. At the unforced idle state, the packet arrival interrupt can be selectively detected by an appropriate transition.

Complete aloha_tx FSM



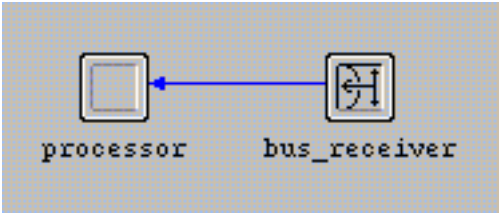
Packet arrival interrupts are the only interrupts expected, so it is safe to omit a default transition for the unforced idle state. When a packet arrival interrupt is delivered, the FSM should perform executives to acquire and transmit the packet in the **tx_pkt** state, then transition back to the **idle** state.

Designing the Generic Receiver Node Model

The generic receiver node model monitors the movement of packets across the bus.

The next step is to design the generic receiver node model. The model does not require a generator because it simply monitors packets moving across the bus. The node model consists of a bus receiver and a processor module.

Conceptual Generic Receiver Node Model

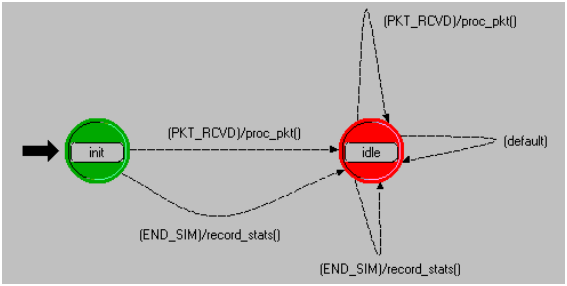


Designing the Generic Receiver Node Process Model

The generic receiver node process model is responsible for handling received packets for statistics-gathering purposes.

To process received packets for statistics collection, the **cct_rx** process needs one unforced state where it waits to receive collision-free packets (how the collisions are detected is presented later in this tutorial). At the end of the simulation, the process records the channel throughput and channel traffic values for analysis. Because the receiver node process manages the statistics-gathering variables, the process should initialize the variables at the start of the simulation. This leads to the design shown. Note the reference to the user-defined C functions **proc_pkt()** and **record_stats()** in the transition executives (these will be written later).

Complete cct_rx FSM



Building the Aloha Model

The **Aloha** process and node models will be created first. These models serve as the basis for an enhanced model that will be used to represent the **CSMA** system.

Building the Aloha model involves several steps:

- Creating the Aloha transmitter process model
- Creating a generic transmitter node model
- Creating a generic receiver process model
- Creating a generic receiver node model
- Building the network model

Creating the Aloha Transmitter Process Model

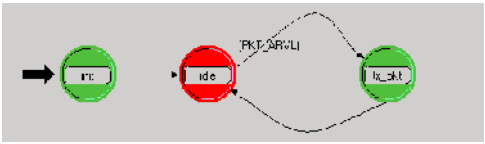
The first part of the Aloha model you will build is the transmitter process model:

1 Start Modeler if it is not already running.

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

3 Using the **Create State** toolbar button, place three states in the workspace.

Create State Toolbar Button



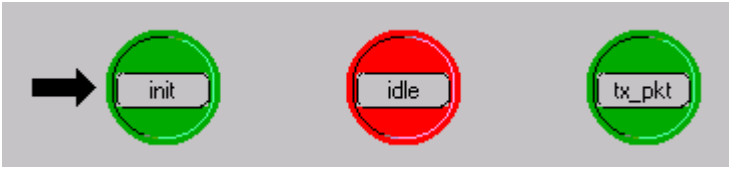
4 Make the following changes to the three states, from left to right:

4.1 To the first state, change the **name** attribute to **init** and the **status** as **forced**.

4.2 To the second state, change the **name** attribute to **idle**; leave the **status** as **unforced**.

4.3 To the third state, change the **name** attribute to **tx_pkt** and the **status** to **forced**.

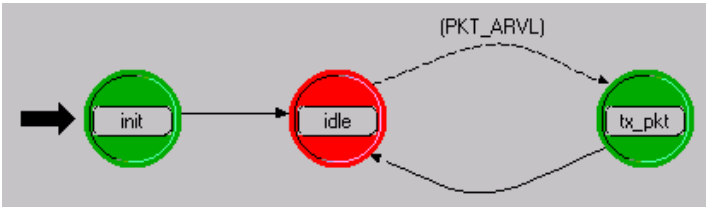
States of the Transmitter Process Model



Next, add the transitions between the states:

1. Draw the three transitions as shown.

Transitions of the Transmitter Process Model



2. For the transition from **idle** to **tx_pkt**, change the **condition** attribute to **PKT_ARVL** (using capital letters). To move the condition label, left-click on the label and drag it to a new position.

The **PKT_ARVL** macro determines if an interrupt received by the process is associated with a packet arriving on a stream. In this model, interrupts are only expected on the input stream from the generator, so the macro does not need to determine which input stream received the packet.

You will define this macro in the next step.

You are now ready to specify the code for the process model. Start with the header block:

1. Open the **Header Block** and enter the code shown. Save the changes.

```
/* Input stream from generator module */
#define IN_STRM 0

/* Output stream to bus transmitter module */
#define OUT_STRM 0

/* Conditional macros */
#define PKT_ARVL (op_intrpt_type() == OPC_INTRPT_STRM)

/* Global Variable */
extern int subm_pkts;
```

2. Save the changes.

The symbolic constants **IN_STRM** and **OUT_STRM** will be used in calls to Kernel Procedures that get packets from streams or send packets to streams. To achieve the desired functionality, these stream indices must be consistent with those defined at the node level.

Next, enter the state variables:

1. Open the **State Variable Block** and enter the following information. The default type, **int**, is acceptable.

   Values for State Variable Block

Type	Name	Comments
int	max_packet_count	Number of packets to process

2. Click **OK** to close the dialog box when you are finished.

The variable **max_packet_count** will hold the maximum number of packets to be processed in the simulation. This will be retrieved from a simulation attribute and compared with the packet count.

Define the actions for the init state in its enter executives block:

1. Double-click on the top of the **init** state to open the **enter executives block** and enter the following code.

```
/* Get the maximum packet count, */
/* set at simulation run-time */
op_ima_sim_attr_get_int32 ("max packet count",
    &max_packet_count);
```

2. Save your changes.

Also, specify the actions for the **tx_pkt** state:

1. Double-click on the top of the **tx_pkt** state to open the **enter executives block**, and enter the following code:

```
/* Outgoing packet */
Packet *out_pkt;

/* A packet has arrived for transmission.  Acquire */
/* the packet from the input stream, send the packet */
/* and update the global submitted packet counter. */
out_pkt = op_pk_get (IN_STRM);
op_pk_send (out_pkt, OUT_STRM);
++subm_pkts;

/* Compare the total number of packets submitted with */
/* the maximum set for this simulation run.  If equal */
/* end the simulation run. */
if (subm_pkts == max_packet_count)
    {
    op_sim_end ("max packet count reached.", "", "", "");
    }
```

2. Save your changes.

The **tx_pkt** state executive is entered when the process receives a stream interrupt from the Simulation Kernel. This interrupt coincides with the arrival of the generated packet. After completing the executives of the **tx_pkt** state, the FSM transitions

back to the **idle** state. Because there are no other unforced states in the transition path, the FSM always re-enters the **idle** state before the next packet arrives and waits for the packet.

The **cct_rx** process model will later declare the global variable, **subm_pkts**, to globally accumulate the number of transmitted packets. Access to this variable in the **aloha_tx** process model is gained by declaring it in the model's header block using the C language extern storage class.

Next define the global attribute that will be set at simulation run-time and loaded into the state variable **max_packet_count**.

1. Choose **Interfaces > Global Attributes**.

2. Enter an attribute "max packet count" into the dialog box table, as shown:

Defining the Global Attribute

Attribute Name	Group	Type	Units	Default Value
max packet count		integer		0

3. Save your changes by clicking on the **OK** button.

The model is now complete, except for the model interface attributes.

You must also edit the process interfaces:

1. Choose **Interfaces > Process Interfaces**.
2. Change the initial value of the **begsim intrpt** attribute to **enabled**.
3. Change the **Status** of all the attributes to **hidden**.

You may want to add a comment to describe the process. When you are finished, click **OK** to close the dialog box.

1. **Compile** the process model. Supply the name **<initials>_aloha_tx**.
2. When the process model is finished compiling, close the Process Model Editor.

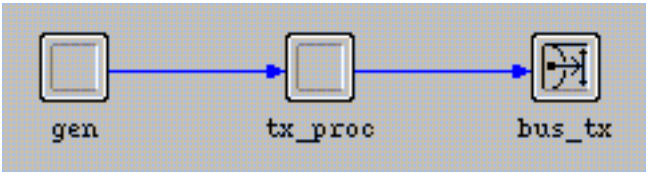
Creating the Generic Transmitter Node Model

You'll now create a node model of a generic transmitter that can support either Aloha or CSMA.

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

2. Using the appropriate toolbar buttons, create two processor modules and one bus transmitter module. (Display the tooltip to verify that you selected a **bus** transmitter.)

   Modules of the Generic Transmitter Node Model

3. For each module, set the name attribute with the names shown above.

4. Set the **process model** attribute for the **gen** processor to **simple_source**.

5. Connect the modules with packet streams as shown above.

6 Open the packet streams' attribute dialog boxes to see that **src stream** is set to **src stream [0]** and **dest stream** is set to **dest stream [0]**, conforming to the indices declared in the **<initials>_aloha_tx** process model header block.

Because you are interested in assigning different values to the generator's **interarrival time** attribute, you must promote it so its value can be set more easily at simulation time.

1 Open the **gen** processor's attribute dialog box.

2 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

3 Close the attribute dialog box.

You also need to set the processor's attributes appropriately:

1. Open the attribute dialog box for **tx_proc** and set the **process model** attribute to **<initials>_aloha_tx**.

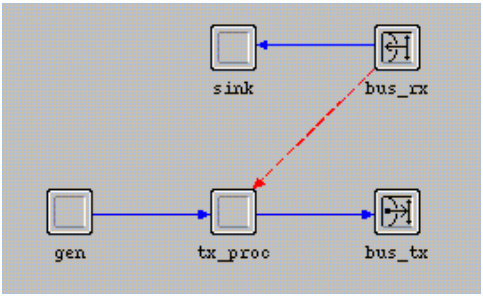
2. Close the dialog box.

Enhancing the Generic Transmitter Node Model

The generic transmitter node model you just created has the functionality necessary for the underlying **aloha_tx** process model. However, because you plan to exchange CSMA for the Aloha process model, it is useful to build hooks for the anticipated enhancements.

The enhancements will consist of a bus receiver module (to support the eventual full duplex capability of the CSMA protocol), and a sink processor to accept and destroy packets received by the receiver module. The enhancements also include an inactive (disabled) statistic wire which, when enabled in the CSMA model, will both inform the process (contained in the **tx_proc** module) of the busy status of the channel, as well as provide interrupts to the process when the channel condition changes.

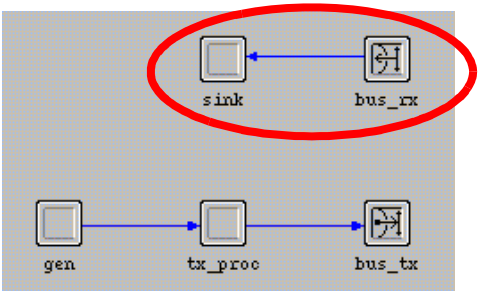
The Enhanced Transmitter Node Model



Add the following features to the node model:

1. Using the appropriate toolbar buttons, add one processor module and one bus receiver module.

Adding Modules

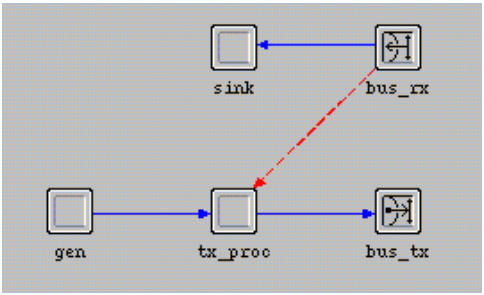


2. Change the **name** of the new processor module to **sink** and the **name** of the bus receiver to **bus_rx**.

3. Connect the new modules with a packet stream as shown.

4. Using the **Create Statistic Wire** toolbar button, connect the **bus_rx** module with the **tx_proc** module.

Adding a Statistic Wire

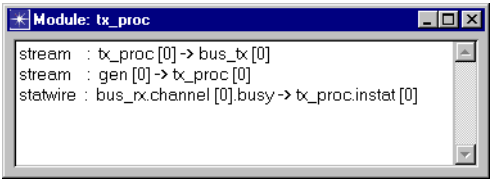


5 Open the attribute dialog box for the **statistic wire** and change both the **rising edge trigger** and **falling edge trigger** attributes to **disabled**. Close the dialog box when finished.

Double-check the module connectivity to make sure all objects in the model have been connected in the correct order:

1 Right-click on the **tx_proc** module and choose **Show Connectivity** from the Object pop-up menu. The objects should be connected as shown in the following figure.

Checking Connectivity



2 If the connections do not match the figure, modify the connectors as follows:

2.1 Right-click on the packet stream between the **gen** and **tx_proc** modules.

2.2 Choose **Edit Attributes**.

2.3 Change the value of the **src stream** attribute to **src stream [0]**.

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

2.5 Right-click on the statistic wire between the **bus_rx** and **tx_proc** modules.

2.6 Choose **Edit Attributes**.

2.7 Change the value of the **dest stat** attribute to **instat [0].**

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

Next, define the interface attributes and write the completed model to disk.

1 Choose **Interfaces > Node Interfaces**.

2 In the **Node Types** table, change the **Supported** value to **no** for the **mobile** and **satellite** types.

3 Change the **Status** of all the attributes to **hidden**, except for the one with **promoted** status, **gen.Packet Interarrival Time**.

4 If you would like, add a comment to describe the node. When you are finished, click **OK** to save the changes.

5 Save the model as **<initials>_cct_tx** and close the Node Editor.

Creating the Generic Receiver Process and Node Models

Next, you can create the generic receiver process and node models. Since the sole purpose of the receiver process is to count packets and record statistics, it can be used to monitor network performance whether the packets are transmitted in accordance with the **Aloha** or the **CSMA** channel access methods.

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

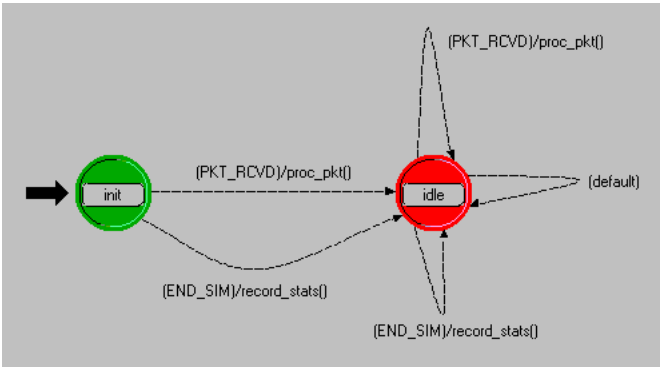
2. Using the **Create State** toolbar button, place two states in the tool window.

3. For the initial state, change the **name** attribute to **init** and the **status** to **forced**.

4. For the other state, change the **name** attribute to **idle**. (Leave the **status** as **unforced**.)

Draw the five state transitions shown in the following figure.

Adding Transitions to the Generic Receiver Node



1. For the first (top) transition between the states, change the **condition** attribute to **PKT_RCVD** and the **executive** attribute to **proc_pkt()**.

2. For the second (bottom) transition between the states, change the **condition** attribute to **END_SIM** and the **executive** attribute to **record_stats()**.

3. For the first (top) transition from **idle** back to itself, change the **condition** attribute to **PKT_RCVD** and the **executive** attribute to **proc_pkt()**.

4 For the second (middle) transition from **idle** back to itself, change the **condition** attribute to **default**.

5 For the third (bottom) transition from **idle** back to itself, change the **condition** attribute to **END_SIM** and the **executive** attribute to **record_stats()**.

Next, enter the code for the header block and the state variables.

1 Using the appropriate toolbar button, open the **Header Block** and type in the definitions shown.

```
/* Input stream from bus receiver */
#define IN_STRM 0

/*Conditional macros */
#define PKT_RCVD (op_intrpt_type () == OPC_INTRPT_STRM)
#define END_SIM (op_intrpt_type () == OPC_INTRPT_ENDSIM)

/* Global variable */
int subm_pkts = 0;
```

2 Save the header block.

The index for the input stream from the bus receiver module (**IN_STRM**) is defined here. The **PKT_RCVD** macro determines if the interrupt delivered to the process is a stream interrupt. Only one kind of stream

interrupt is ever expected, so no further qualifications are necessary. The **END_SIM** macro determines if the interrupt received by the process is associated with an end-of-simulation interrupt from the Simulation Kernel.

The global variable **subm_pkts** is used so that all transmitting nodes can contribute their individual transmission attempts to this accumulator. Declaring a variable in a process model header block causes it to behave as a global variable within the executable simulation.

The generic receiver process uses the **rcvd_pkts** state variable to keep track of the number of valid received packets. Define this variable as follows:

1. Open the **state variables block** and define the following variable:

Defining the rcvd_pkts State Variable

Type	Name	Comments
int	rcvd_pkts	Received packet counter

2. Save the state variables block.

Next, enter the code that defines the functionality of the process model.

1. Open the **function block** and enter the following code:

```
/* This function gets the received packet, destroys */
/* it, and logs the incremented received packet total*/
static void proc_pkt (void)
    {
    Packet* in_pkt;
    FIN (proc_pkt());
    /* Get packet from bus receiver input stream */
    in_pkt = op_pk_get (IN_STRM);

    /*Destroy the received packet */
    op_pk_destroy (in_pkt);

    /* Increment the count of received packet */
    ++rcvd_pkts;
    FOUT;
    }

/* This function writes the end-of-simulation channel */
/* traffic and channel throughput statistics to a */
/* scalar file                    */
static void record_stats (void)
    {
    double cur_time;
    FIN (record_stats());
    cur_time = op_sim_time();
    /* Record final statistics */
    op_stat_scalar_write ("Channel Traffic G",
       (double) subm_pkts / cur_time);
```

```
op_stat_scalar_write ("Channel Throughput S",
   (double) rcvd_pkts / cur_time);
FOUT;
}
```

2 Save the function block.

As defined in the function block earlier, the **proc_pkt()** function acquires each received packet as it arrives, destroys it, and increments the count of received packets. The **record_stats()** function is called when the simulation terminates.

The **op_stat_scalar_write** function sends the channel throughput and traffic data to a scalar file that you will specify when you configure the simulation.

The init state initializes the state variable used to count received packets. Define it as follows:

1 Double-click on the top of the **init** state to open the **enter executives block** and enter the following code:

```
/* Initialize accumulator */
rcvd_pkts = 0;
```

2 Save the enter executives.

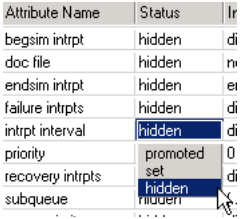
Finally, you can define the process interfaces.

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

2. Change the initial value of the **endsim intrpt** attribute to **enabled**.

3. Change the **Status** of all the attributes to **hidden**.

Hiding Attributes



4. If you want, add a comment to describe the process then click **OK** to save your changes.

Now compile the model.

1. Click on the **Compile Process Model** toolbar button.

2. Supply the file name **<initials>_cct_rx** and click on the **Save** button.

3. Close the compilation dialog box and the Process Model Editor.

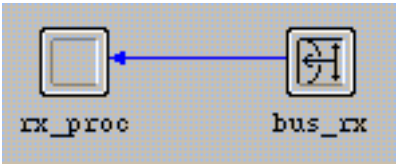
Creating the Generic Receiver Node Model

The next step is to create a generic receiver node model.

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

2. Using the appropriate toolbar buttons, create one processor module and one bus receiver module. (Display the tooltip to verify that you selected a **bus** receiver.)

   Modules of the Generic Receiver Node Model

3. For each module, change the **name** attribute as shown.

4. Connect the modules with a packet stream as shown.

   The input stream index defaults to stream 0, conforming to the index declared in the **cct_rx** process model header block.

5 Open the processor's attribute dialog box and set the **process model** attribute to **<initials>_cct_rx**. Close the dialog box when finished.

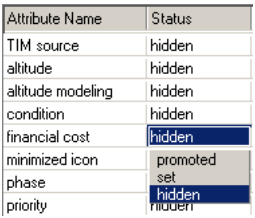
The generic receiver node model is now complete, except for the interface attributes.

1 Choose **Interfaces > Node Interfaces**.

2 In the **Node Types** table, change the **Supported** value to **no** for the **mobile** and **satellite** types.

3 In the **Attributes** table, change the **Status** of all the attributes to **hidden**.

Hiding Attributes



4 If you wish, add a comment to describe the node model. When you are finished, click **OK** to exit the dialog box.

5 **Save** the node model as **<initials>_cct_rx**, then close the Node Model Editor.

Creating a New Link Model

The behavior of a bus link is defined by its Transceiver Pipeline stages. The pipeline is a series of C or C++ procedures which can be modified to customize the link model.

For this lesson, you will create a custom bus link model whose pipeline stages use the default bus models, denoted by the **dbu_** model prefix. The following table lists pipeline stages by function.

Bus Transceiver Pipeline Model Stages

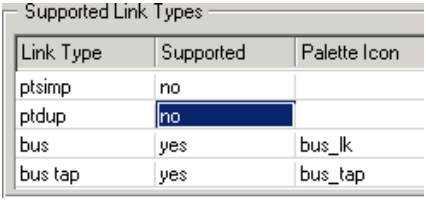
Model	Function
txdel	Computes the transmission delay associated with the transmission of a packet over a bus link (transmission delay is the time required to transmit the packet at the bit rate defined in the relevant bus transmitter module).
closure	Determines the connectivity between any two stations on the bus.
propdel	Calculates the propagation delay between a given transmitter and a receiver.
coll	Determines whether a packet has collided on the bus.
error	Calculates the number of bit errors in a packet.
ecc	Rejects packets exceeding the error correction threshold as well as any collided packets.

To create a new bus link model:

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

2. In the **Supported Link Types** table, change the **Supported** value to **no** for the **ptsimp** and **ptdup** types.

   Modifying the Link Types Supported

   This link model supports only the bus and bus tap types.

3. If you wish, add a comment to describe the link.

4. Save the file as **<initials>_cct_link** and close the Link Model Editor.

Creating the Network Model

The network model will be built so that it can be used when analyzing both the Aloha and CSMA protocols. This will be done by defining the nodes so that they reference the generic node models, and later changing the referenced process models at the node level.

The analytical Aloha model assumes that packets are always introduced into the network at exponentially distributed interarrival times. However, in this tutorial, the network model has a finite number of nodes that hold packets in their buffers until the previous outstanding transaction finishes. To closely follow the analytical model's assumptions, there must be a large number of transmitter nodes on the bus.

The network model will be constructed within a subnet so that a small-scale coordinate system can be used.

1. Choose **File > New...**, then select **Project** from the pull-down menu. Click **OK**.

2. Name the project **<initials>_cct_network** and the scenario **aloha**, then click **OK**.

3. In the Startup Wizard, use the following settings:

Startup Wizard Settings

Dialog Box Name	Value
Initial Topology	Default value: **Create empty scenario**
Choose Network Scale	**Office** (“Use metric units” selected)
Specify Size	**700 x 700 Meters**
Select Technologies	None
Review	Check values, then click **Finish**

To build your network more easily, you need a custom palette that has the necessary objects. To create the palette:

1. In the object palette, click on the **Configure Palette…** button.

2. In the **Configure Palette** dialog box, click **Clear**.

   ➥ All objects except the subnet are removed from the palette If you have the Wireless module installed, you will also see the mobile and satellite subnets.

3. Click on the **Link Models** button, then add **<initials>_cct_link** from the list of available link models. Click **OK** to close the dialog box when you are finished.

4 Click on the **Node Models** button, then add **<initials>_cct_rx** and **<initials>_cct_tx** from the list of available node models. Click **OK** to close the dialog box when you are finished.

5 Save the object palette by clicking on the **Save As...** button in the **Configure Palette** dialog box. Use **<initials>_cct** as the file name.

6 Click **OK** to close the **Configure Palette** dialog box.

➥ The **<initials>_cct** Object Palette is ready for use.

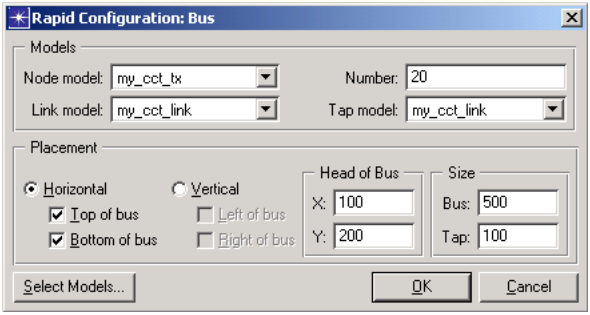
Instead of creating the entire bus network by hand, you can use rapid configuration to build it quickly:

1 Choose **Topology > Rapid Configuration...**

2 Select **Bus** from the menu of available configurations, then click **OK...**

3 Use the values shown in the following figure to complete the **Rapid Configuration: Bus** dialog box. Most of the values will be the same, but your models will be named **<initials>_cct_tx** and **<initials>_cct_link**.

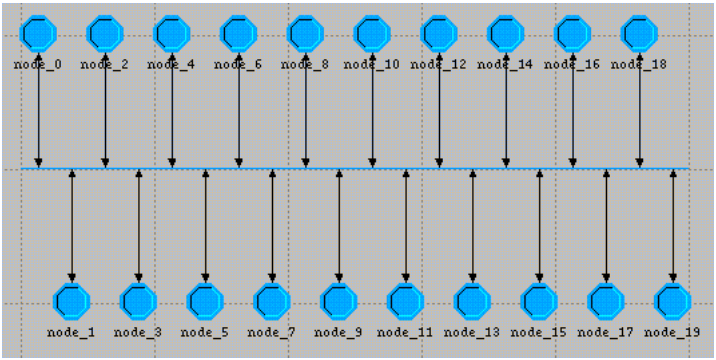
Rapid Configuration: Bus Dialog Box



4 Click **OK** when all the values are entered.

➥ The network is drawn in the workspace.

Bus Network Created



This network still needs a receiver node. To add this node and connect it to the network:

1. Click and drag the receiver node **<initials>_cct_rx** from the object palette into the left side of the workspace.

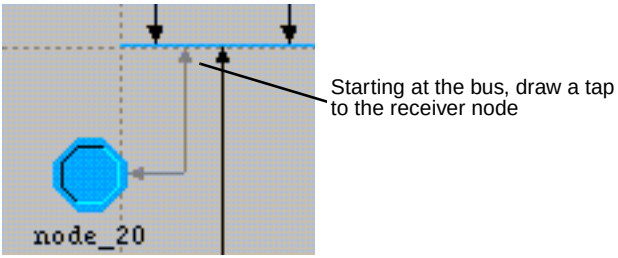
2. Click on the **<initials>_cct_link** tap link in the palette. Be sure to use the tap link.

   Bus Tap Icon

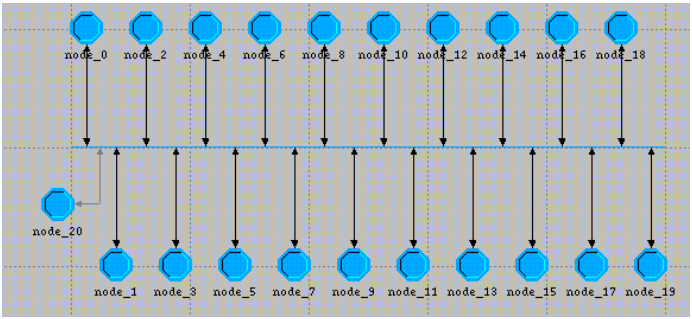
3. Draw a tap from the bus to the receiver node. Be sure to start at the bus. Drawing the tap from the node to the bus might produce different results.

   Drawing the Tap

4. Verify that the completed bus model looks like this:

Completed Bus Model



5 Save the model with the default name, **<initials>_cct_network**, and close the object palette.

Do not exit the Project Editor.

Executing the Aloha Simulation

The goal of this lesson is to observe how the performance of the protocols varies as a function of channel traffic. The interarrival time input parameter will be varied in a series of simulations to produce different levels of traffic and, therefore, different levels of throughput. You will run 12 simulations, each with a different interarrival time value, analyze the results, and draw conclusions.

Importing and Configuring the Simulation Sequence

1. Choose **Scenarios > Scenario Components > Import...**

2. Select **Simulation Sequence** from the pull-down menu, then select **cct_network-CSMA**.

3. Save the project.

4. Choose **DES > Configure/Run Discrete Event Simulation (Advanced)**.

   ➥ The **Simulation Set** dialog box opens. Notice that it contains an icon with two arrows, indicating a simulation sequence with multiple simulations. This is the file you imported.

   Simulation Sequence Icon

   Note: You would normally open the Simulation Sequence Editor—**DES > Configure/Run Discrete Event Simulation (Advanced)**—and create a simulation sequence file with specific settings. To save time, a nearly-complete file has been provided. However, you still need to specify an output scalar file which collates the scalar results produced by the 12 simulations.

5 Right-click on the simulation sequence icon and select **Edit Attributes**.

6 Click on the Execution tree node, then the Advanced tree node.

➥ The Application page appears

7 Verify that the **Network** model is set to **<initials>_cct_network-aloha**.

8 Click on the Outputs tree node, then the Statistics Collection tree node.

9 Set **Probe file** to **<NONE>**. You do not need to create or specify a Probe file. The **op_stat_scalar_write** function and the scalar file substitute for the Probe file.

10 Set the **Scalar file** to **<*your initials*>_cct_a**.

➥ This file will collect the output of the **op_stat_scalar_write** function you included in the function block of the **<initials>_cct_rx** model.

If the output scalar file **<initials>_cct_a** does not exist when the simulation sequence begins, one will be created so that scalar results may be recorded. If the file already exists, the simulation executables will append their scalar results to this

file. To avoid viewing obsolete results that might already exist in a similarly named file, the output scalar file **<initials>_cct_a** must be deleted if it exists.

11 Check the **Clear scalar file before running simulation set** check box.

12 Click on the Inputs tree node, then the Global Attributes node, and verify that **max packet count** is **1000.**

13 Click on the Object Attributes tree node.

➥ In the Value column, notice the 12 values that have been set for the attribute **Office Network.*.gen.Packet Interarrival Time**.

14 Click **Yes** to save changes and close the **Simulation Set** dialog box.

15 Choose **File > Save**.

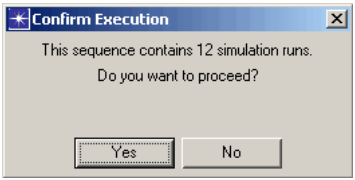
The simulation can now be executed. Because the simulation sequence executes many individual simulations, the total execution time might be several minutes.

1 Click on the **Execute Simulation Sequence** toolbar button.

Execute Simulation Sequence Toolbar Button

2 Click **Yes** in the **Confirm Execution** dialog box. A sequence composed of many runs may be time-consuming to execute, and this dialog box gives you the option of deferring the process.

Confirm Execution Dialog Box



➥ The 12 simulations display their progress as they execute. Any simulation run that generates 1000 packets (the value of **max packet count**) will terminate with a message similar to the one in the following figure:

Simulation Terminates if max packet count is reached

```
---
Beginning simulation of opnet_cct_network-aloha at 11:47:26 Wed Nov 12 2003
---
Kernel: development (not optimized), sequential
---
Simulation terminated by process (opnet_aloha_tx) at module (top.Office Network.node_13.tx_proc)
T (10308.6), EV (100886), MOD (top.Office Network.node_13.tx_proc), PROC (op_sim_end)
---
Max packet count reached.
---
Simulation Completed - Collating Results.
Events: Total (100,888), Average Speed (294,135 events/sec.)
Time: Elapsed (0.34 sec.), Simulated (2 hr. 51 min. 48 sec.)
---
```

3 When the simulations are complete, close the **Simulation Sequence** dialog box and the Simulation Sequence Editor. If you had problems, see *"Troubleshooting Modeler Tutorials"*.

Analyzing the Aloha Results

Aloha channel performance can be measured according to the number of successfully received packets as a function of the packets submitted, regardless of whether the packets are original or retransmitted. In this network, **channel throughput** is a typical measurement of network performance.

The results of each simulation are stored as two scalar values in the output scalar file, allowing you to view the network's performance as a function of an input parameter rather than a function of time. The channel throughput as a function of channel traffic across all of the simulations can be viewed. Because these are scalar results, you must use the Analysis Configuration Editor.

To open the scalar output file:

1. In the Project Editor, choose **File > New...**, then select **Analysis Configuration** from the pull-down menu. Click **OK**.

   ➥ The Analysis Configuration Editor opens.

2. Choose **File > Load Output Scalar File...**

3 Select **<initials>_cct_a** from the list of available files. There will be no indication that the file is selected.

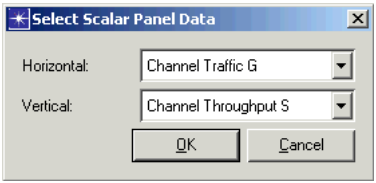
Draw the scalar panel:

1 Click on the **Create a Graph of Two Scalars** toolbar button.

Create a Graph of Two Scalars Toolbar Button

2 Select the horizontal variable **Channel Traffic G** first, then select the vertical variable **Channel Throughput S** from the menu of available scalars that appears.

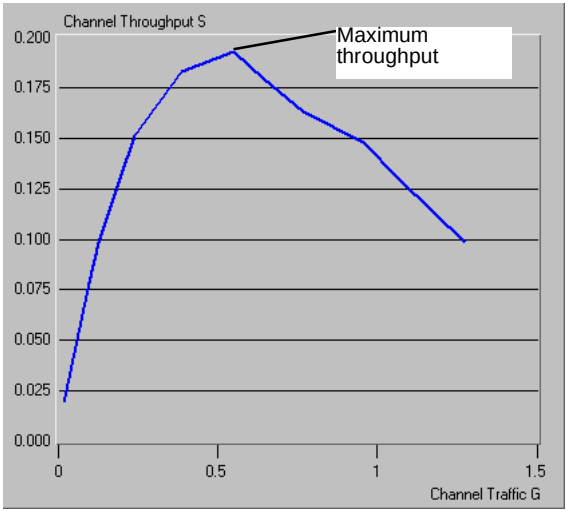
Select Scalar Panel Data Dialog Box



3 Click **OK**.

➥ The scalar graph appears in the workspace. Your graph should resemble the one in the following figure:

Aloha Protocol: Channel Throughput as a Function of Channel Traffic



Theoretical analyses have shown that a pure Aloha system has a channel throughput S as a function of channel traffic G given by $S = Ge^{-2G}$. This relationship gives a maximum channel throughput of $S_{max} = 1/2e \approx 0.18$.

At low traffic levels, collisions seldom occur. At high traffic levels, the channel is overwhelmed and excessive collisions prevent packets from being successfully received. This behavior is amply demonstrated by the simulation results. In particular, the maximum throughput is achieved near G = 0.5 and is close to the expected value of 0.18.

The theoretical results assume an essentially infinite number of sources to eliminate the buffering effects which emerge in a real network. The analytical model also assumes that the system is in an ideal steady state condition. Any differences in the measured performance of this model and the analytical models can be attributed to peculiarities of the random number seeds selected for individual simulations (which can be fixed by using multiple seeds) and the real world limitations (including finite simulation time and finite number of nodes) imposed by the models.

When you are finished viewing the graph, close the graph panel and the Analysis Configuration Editor.

Adding Deference

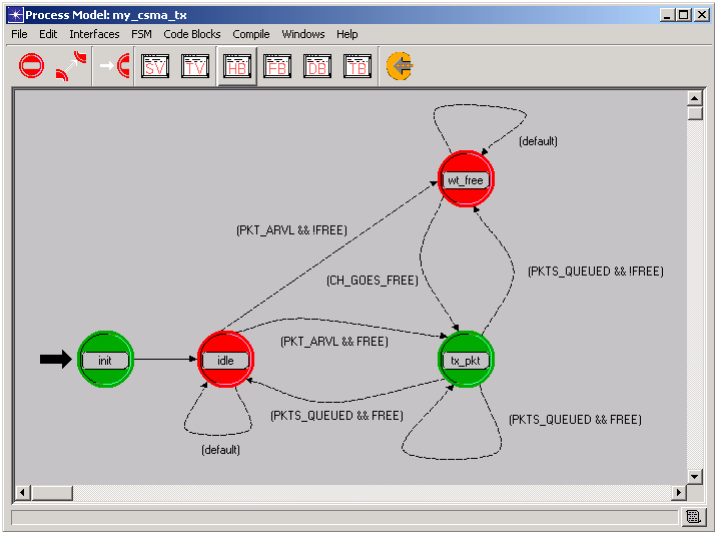
The performance of the Aloha random access protocol can be enhanced by adding a carrier sense capability. The carrier sense capability is employed in the classical CSMA protocol, which requires a source node to sense the channel and determine that it is free before committing to a transmission.

You can enhance the existing **<initials>_aloha_tx** process model so that the process waits until the channel is free before transmitting a packet.

In the Project Editor, choose **File > Recent Files > Process Models** and select the **<initials>_aloha_tx** model.

Modify the states and transitions so that the model appears as shown in the following figure.

The CSMA Process Model



1. Create a new state and name it **wt_free**.

2. Create a transition from **wt_free** to **tx_pkt**, and change the condition to **CH_GOES_FREE**.

3. Create a transition from the **wt_free** state back to itself and set the condition to **default**.

4. Create a transition from the **idle** state to **wt_free** and change the condition to **PKT_ARVL && !FREE**.

5. Add a transition from the **idle** state back to itself with a condition of **default**.

6. Change the condition on the transition from **idle** state to the **tx_pkt** state to **PKT_ARVL && FREE**.

7. Change the unconditional transition from **tx_pkt** to **idle** to conditional by setting the condition attribute to **default**.

8. Create a transition from **tx_pkt** back to itself, and set the condition to **PKTS_QUEUED && FREE**.

9. Finally, create a transition from **tx_pkt** to **wt_free** and set the condition to **PKTS_QUEUED && !FREE**.

Remember, you can move a condition label by left-clicking on the label and dragging it to a new position.

Editing the Header Block

You must change the header block so that the process verifies that the channel is free before transmitting. For the process to send a packet, it must first confirm that the channel is free by using the Kernel Procedure **op_stat_local_read()** to read the channel's **busy** statistic. If the channel is not free, the process enters the state **wt_free** until a "channel goes free" interrupt is received.

At the node level, the underlying statistic wire is triggered when the **busy** statistic changes to 0.0. The triggering is activated by enabling the wire's **falling edge trigger** attribute.

1. Add the following lines to the end of the process model header block.

```
/* input statistic indices */
#define CH_BUSY_STAT 0

/* Conditional macros */
#define FREE (op_stat_local_read (CH_BUSY_STAT) == 0.0)
#define PKTS_QUEUED (!op_strm_empty (IN_STRM))
#define CH_GOES_FREE (op_intrpt_type () == \
    OPC_INTRPT_STAT)
```

2. Save the header block.

3 Choose **File > Save As...** and rename the model **<initials>_csma_tx**.

4 **Compile** the model, then close the Process Editor.

Enhancing the Generic Transmitter Node Model

You can enhance the generic transmitter node model so that the bus receiver module delivers a falling edge statistic interrupt to the processor module whenever the receiver **busy** statistic changes from “busy” (1.0) to “free” (0.0).

To enhance the generic transmitter node model so that it supports CSMA:

1. Select **File > Recent Files > Node Model** and select **<initials>_cct_tx**.

2. Right-click on the **statistic wire** and choose **Edit Attributes** from the pop-up menu. Set the **falling edge trigger** attribute to **enabled**. Click **OK**.

3. Open the **Attributes** dialog box for the **tx_proc** processor module and change the **process model** attribute to **<initials>_csma_tx**. Close the dialog box.

   ➥ The processor now uses a process model which acts on **channel busy** statistic interrupts delivered by the receiver module.

4. Choose **File > Save As...** rename the model **<initials>_cct_csma_tx**. Close the Node Editor.

Redefining the Network Model

Now that you have modified the appropriate models to support CSMA, you need to change the network model to use the new models. Instead of creating an entirely new model, you can duplicate the existing scenario (including the network model) and make the appropriate changes.

1. In the Project Editor, choose **Scenarios > Duplicate Scenario...** and name the new scenario **CSMA**.

The only change to the network model is to use the new CSMA transmitter nodes.

1. Add the **<initials>_cct_csma_tx** node model to your object palette and save the palette with the default name.

2. Right-click on one of the transmitter nodes and choose **Select Similar Nodes**.

   ➥ All 20 transmitter nodes are selected.

3. Right-click on any of the selected nodes and choose **Edit Attributes** from the pop-up menu.

4. Check **Apply changes to selected objects**.

5 Change the **model** attribute to **<initials>_cct_csma_tx**, then click **OK**.

➥ A dialog box appears to warn you that the change cannot be undone.

6 Click **Yes**.

➥ The node models are changed to **<initials>_cct_csma_tx.** The phrase “20 objects changed” appears in the message buffer.

Configuring CSMA Simulations

Configure a series of simulations for the CSMA model.

1. Save the project.

2. Choose **DES > Configure/Run Discrete Event Simulation (Advanced)**.

3. Right-click on the simulation set and choose **Edit Attributes**.

4. Change the **Seed** to **11**.

5. Click the Outputs tree node, then the Statistics Collection node, and change the **Scalar file** to **<initials>_cct_c**.

6. Set **Probe file** to **<NONE>**. Click **OK** to close the dialog box.

7. Save the simulation sequence file (you do not have to rename it because it was duplicated and renamed when you duplicated the scenario).

8. Execute the simulation. It may take a few minutes to run the 12 simulations. When they complete, close the Simulation Sequence Editor.

Analyzing the CSMA Results

View the results.

1. In the Project Editor, choose **File > New...**, then select **Analysis Configuration** from the pull-down menu. Click **OK**.

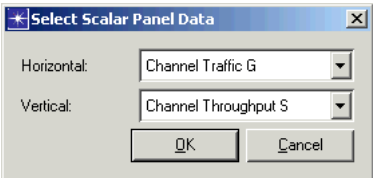
2. In the Analysis Configuration Editor, choose **File > Load Output Scalar File...**

3. Select **<initials>_cct_c** from the list of available files.

4. Click on the **Create a Graph of Two Scalars** toolbar button.

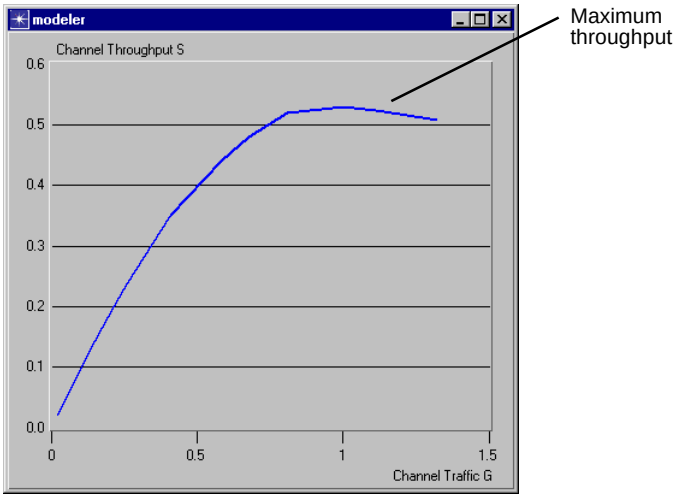
5. In the **Select Scalar Panel Data** dialog box, select the horizontal variable **Channel Traffic G** first, then select **Channel Throughput S** as the vertical variable, then click **OK**.

Select Scalar Panel Data Dialog Box



➥ Your graph should resemble the one in the following figure:

CSMA Protocol: Channel Throughput as a Function of Channel Traffic



The CSMA protocol achieves a maximum channel throughput of about 0.5.

Viewing Both Results on the Same Graph

Your goal is to compare the Aloha and CSMA protocols. The easiest way to do so is to display both traces on the same graph.

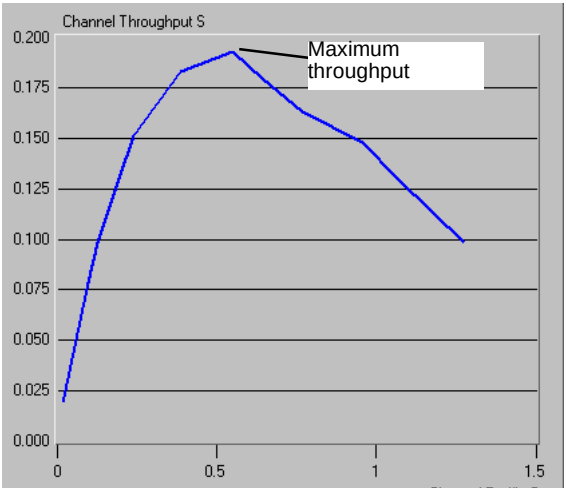
To view both results on a single graph panel, you will first create a scalar graph for the Aloha results, then create a vector graph that displays both results.

First, create a scalar graph for the Aloha results.

1. Choose **File > Load Output Scalar File...**
2. Select **<initials>_cct_a** from the menu.
3. Click on the **Create a Graph of Two Scalars** toolbar button.
4. Select the horizontal variable **Channel Traffic G** first, then select the vertical variable **Channel Throughput S** from the menu of available scalars that pops up. Click **OK**.

   ➥ Your graph should resemble the following one:

Aloha Protocol: Channel Throughput as a Function of Channel Traffic



The Aloha protocol achieves a maximum channel throughput of about 0.185.

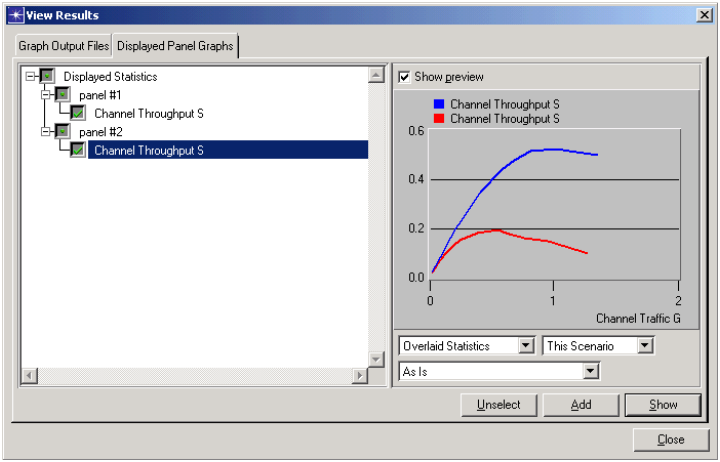
Now, create a vector graph that displays both results.

1. Choose **Panels > Create Vector Panel...**

2. Select the **Displayed Panel Graphs** tab, then open the **Displayed Statistics** treeview and select both displayed statistics.

3. Change the display mode to **Overlaid Statistics**.

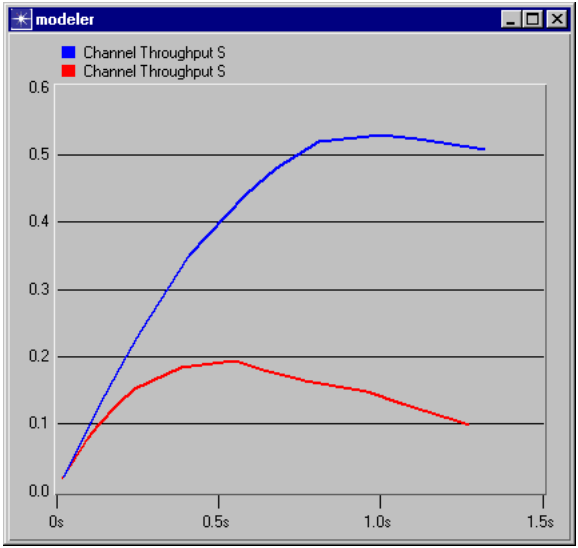
Selecting the “Overlaid Statistics” Filter



4 Click **Show.**

➥ The graph of the two scalars should resemble the following graph:

Aloha and CSMA Protocols Compared



Renaming the Traces

Because both the Aloha and CSMA simulation executables used the same label in the **op_stat_scalar_write()** Kernel Procedure, both traces are called **Channel Throughput S**. These traces can be renamed to be more informative.

1 Display the **Edit Graph Properties** dialog box by right-clicking on the multiple vector graph and selecting **Edit Graph Properties** from the pop-up menu. Be sure to click on the graph and not on the panel (the area around the graph).

➥ Notice the pull-down menu of active traces in the top section of the dialog box.

Active Traces Pull-down Menu

2 Click and hold the pull-down menu to see the list of active traces. Both are named **Channel Throughput S**. Which is the CSMA trace and which the Aloha trace? In this pull-down menu, traces are listed in the order in which they were added to the multi-trace graph. The first trace listed is the CSMA trace.

3 Change the label for the CSMA trace:

3.1 Make sure the pull-down menu shows the first trace.

3.2 Change **Custom Title** to **CSMA Channel Throughput S**.

3.3 Click on the **Apply** button at the bottom of the dialog box.

4 Change the label for the Aloha trace:

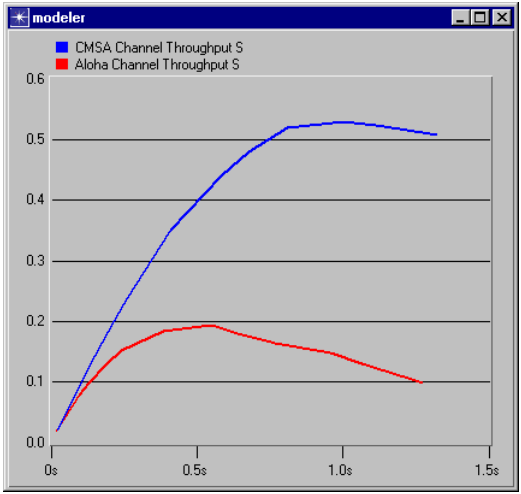
4.1 Select the second active trace listed in the pull-down menu (it is still called **Channel Throughput S**).

4.2 Change **Custom Title** to **Aloha Channel Throughput S**.

5 Click **OK**.

➥ The graph should appear as follows:

Renamed Traces Overlaid



The CSMA protocol is shown to be superior to the Aloha protocol at all channel traffic loads.

The theoretical channel throughput S as a function of channel traffic G in a 1-persistent CSMA channel with negligible propagation delay is given by $\mathbf{S=G(1+G)e^{-G}/(G+e^{-G})}$. This formula predicts a maximum throughput of approximately 0.5 at a channel traffic of approximately 1.0. Although the simulations are brief and limited, the results support this prediction.

6 Close the graphs and the Analysis Configuration Editor.

Adding Collision Detection and Backoff

If a node has full-duplex capability, it can both transmit and monitor a connected bus link at the same time. This capability can be modeled using the Ethernet protocol.

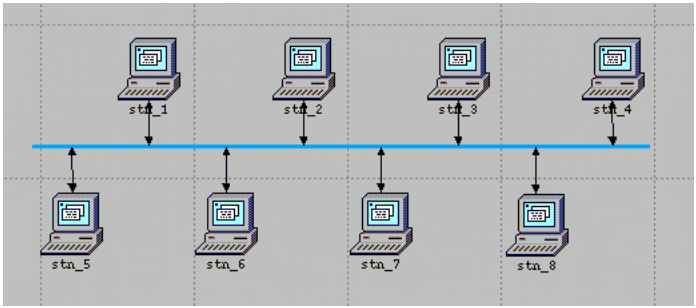
A node with full-duplex capability can both transmit and ‘listen’ on the line to determine whether a collision condition exists. This operational mode is commonly referred to as Carrier-Sense Multiple Access with Collision Detection (or CSMA/CD). This is practiced by the commercial protocol Ethernet, and is accurately modeled by an OPNET-supplied example model.

Because Ethernet is a fairly sophisticated model, you will not build it yourself. Instead, the section provides a guided tour of the standard Ethernet process, node, and network models.

The ethcoax_net Network Model

The **ethcoax_net** network model consists of a multi-tap bus network populated by eight nodes. The nodes employ the node model **ethcoax_station_adv**.

The ethcoax_net Network Model

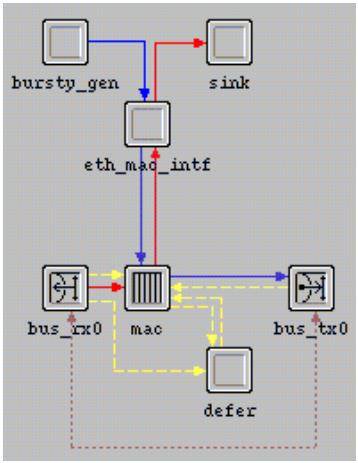


The ethcoax_station_adv Node Model

The **ethcoax_station_adv** node model is significantly more complicated than the Aloha or CSMA node models. It has four processor modules, a queue module which performs the bulk of the channel access processing, and a pair of bus receiver and transmitter modules.

The **ethcoax_station_adv** node model provides part of the functionality associated with the OSI Data Link Layer called the Media Access Control (MAC) sublayer. The functions of the individual modules are discussed in the following paragraphs.

The ethcoax_station_adv Node Model



The **bus_tx** and **bus_rx** modules serve as the bus link interface. These modules are set to transmit and receive at a data rate of 10 Mbits/second, the standard data rate used in an Ethernet network.

The **sink** processor represents higher layers and simply accepts incoming packets which have been processed through the **mac** process.

The **defer** processor independently monitors the link's condition and maintains a deference flag which the **mac** process reads over a statistic wire to decide whether transmission is allowed.

The **bursty_gen** module represents higher layer users who submit data for transmission. It uses an ON-OFF pattern for traffic generation.

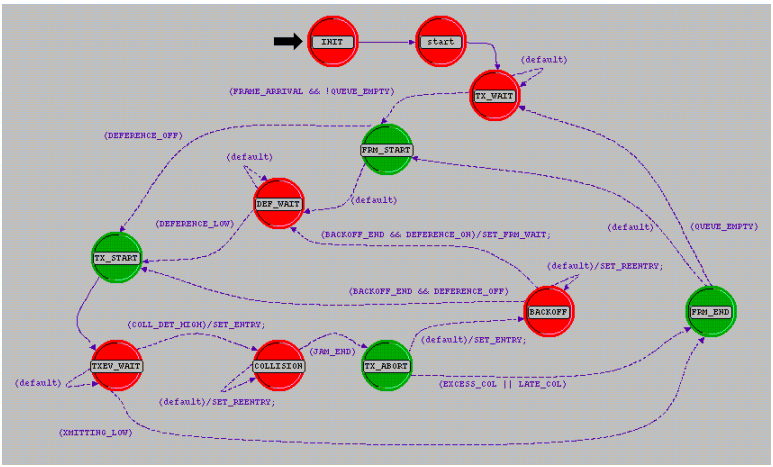
The **mac** process handles both incoming and outgoing packets. Incoming packets are decapsulated from their Ethernet frames and delivered to a higher level process. Outgoing packets are encapsulated within Ethernet frames and when the deference flag goes low, a frame is sent to the transmitter. This process also monitors for collisions, and if one occurs, the transmission is appropriately terminated and rescheduled for a later attempt.

The eth_mac_v2 Process Model

The **eth_mac_v2** process model manages the transmission and reception of packets. These tasks have been decomposed into three basic functions:

- encapsulating and queuing outgoing packets
- decapsulating and delivering incoming packets
- managing an ongoing transmission

The eth_mac_v2 Process Model

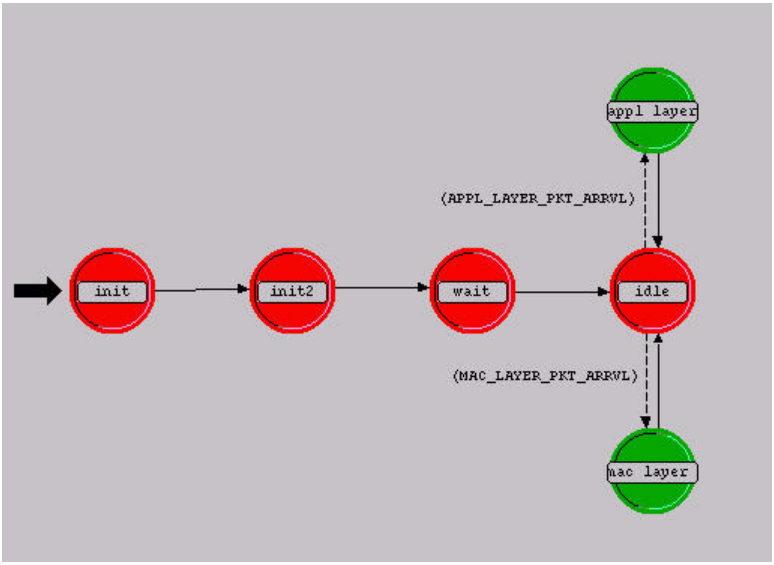


The ethernet_mac_interface Process Model

The **ethernet_mac_interface** process converts packets representing the application data into ethernet form for the **mac** processor.

The ethernet_mac_interface process takes packets from a traffic source, assigns a valid destination address (if random assignment is specified for traffic destination), and sends them to the mac processor. It also accepts packets from the mac processor and forwards them on to the higher layer traffic sink process.

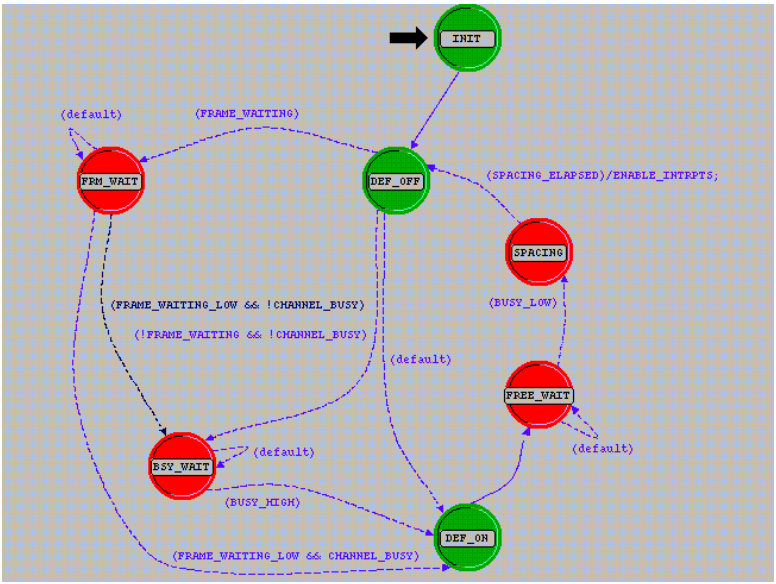
The ethernet_mac_interface Process Model



The eth_defer_v2 Process Model

The **eth_defer_v2** process determines whether the deference flag should be raised or lowered. The deference flag is read by the **eth_mac_v2** process to decide whether a transmission is permissible or whether the channel must be deferred to another user.

The eth_defer_v2 Process Model



Executing the ethcoax_net Simulation

Load the pre-defined Ethernet model and run a simulation.

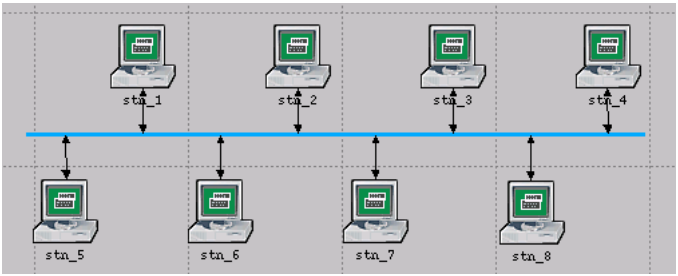
1. Go to **<opnet_dir> \ <release> \ models \ std \ tutorial_req \ modeler**.

2. Open the **ethcoax_net** Project.

   ➥ The ethcoax_net model opens in the workspace.

3. Choose **File > Save As...** and save the project as **<initials>_ethcoax_net** in your default model directory.

   The ethcoax_net Model



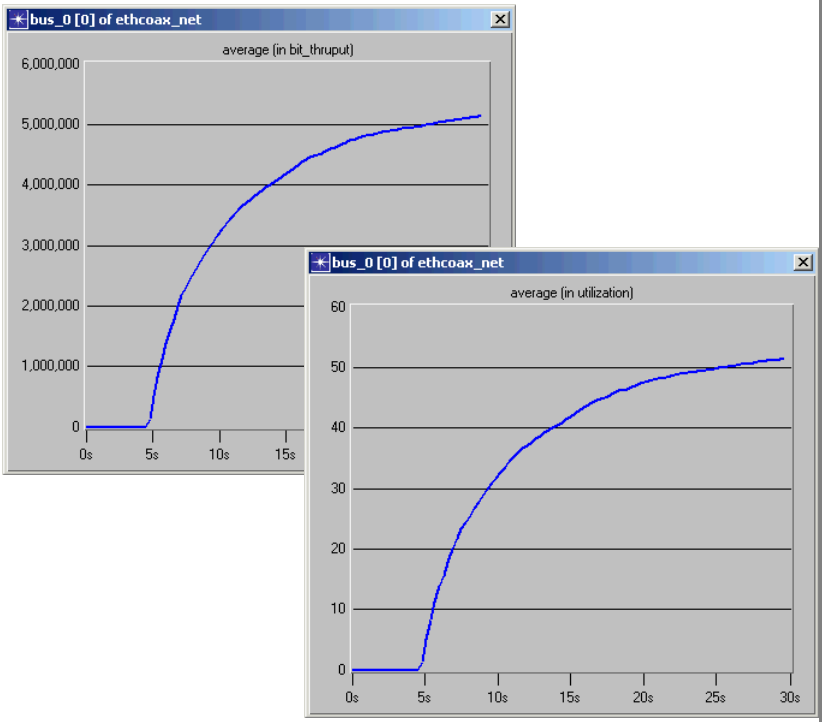
4. Choose **DES > Run Discrete Event Simulation**.

Analyzing the Ethernet Results

Because the default Ethernet model collects results differently from the Aloha and CSMA simulations, you need to use a slightly different approach to view the results from this simulation.

1. In the Project Editor, choose **DES > Results > View Statistics**
2. Select **Object Statistics > ethcoax_net > bus_0 [0] > utilization**.
3. Change the filter type from **As Is** to **average**, then click **Show**.
4. Click the **Unselect** button in the **View Results** dialog box, then select **Object Statistics > ethcoax_net > bus_0 [0] > bit_thruput**.
5. Click **Show**.
6. Place the graphs so you can see both clearly and consider the results. The graphs should resemble the following ones.

bit_thruput and utilization Graphs



Though the general trend lines are the same, the graphs have radically different ordinate bounds.

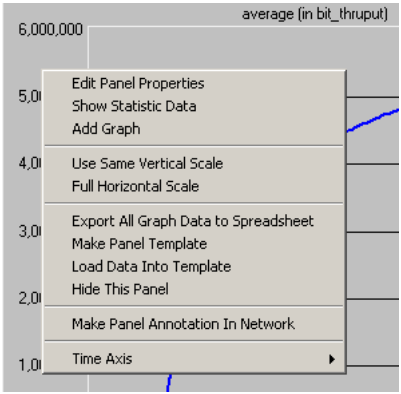
The **bit_thruput** statistic measures the average number of bits successfully received by the receiver per unit time. By definition, this statistic only counts the bits associated with collision-free packets and can reach a maximum value of no more than

10 Mbits/second, the data rate assigned to the channel. As you can see, the throughput after 30 seconds of simulation stabilizes near 5.1 Mbits/second. To get a more accurate reading of the actual throughput, you can view the vector graph as a set of data points.

1 Verify that the **bit_thruput** panel is the active window.

2 Right-click in the panel border and choose **Show Statistic Data.**

Selecting "Show Statistic Data"



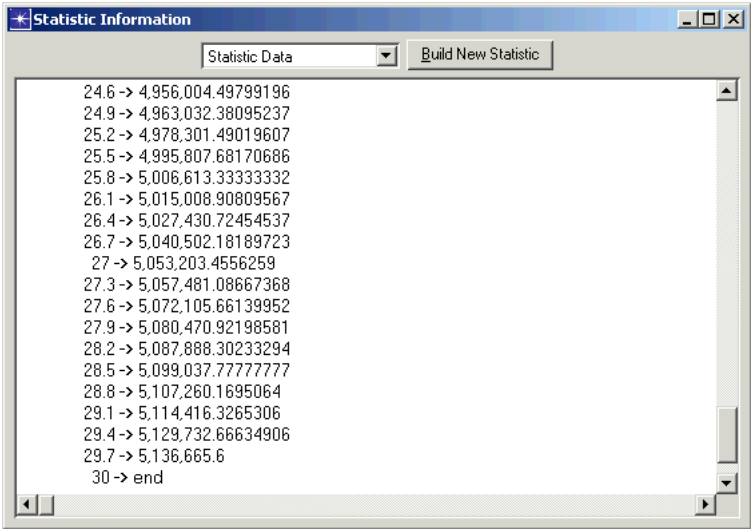
➥ A window opens, showing the sequence of ordinate and abscissa pairs for the vector, in ASCII format.

3 From the pull-down menu at the top of the **Statistic Information** dialog box, select **Statistic Data**.

4 Scroll to the bottom of the editing pad to read the final value of the vector.

The ASCII data should resemble that shown:

Bit Throughput at End of Simulation



At the end of the simulation, the receiver's **bit_thruput** statistic is indeed almost exactly 5.1 Mbits/second. When divided by the channel capacity, this raw level of bit throughput results in a normalized channel throughput of 0.51 (that is, channel utilization of 51 percent).

When you are finished viewing the data, close the **Statistic Information** dialog box.

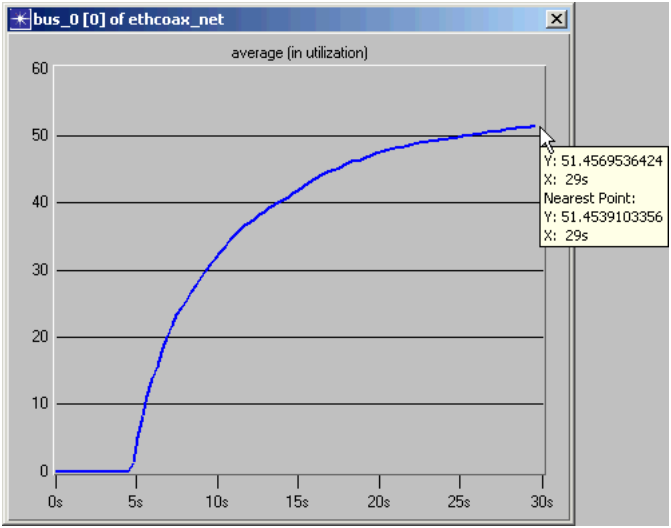
You can also see these values on the utilization graph:

1. Click on the **average (in utilization)** graph to activate that window.

2. Move the cursor to the far right of the vector and let the cursor come to rest. The tooltip shows the final value of the channel utilization.

   You should see an average channel utilization of about 51 percent. This value is the percentage of the 10 Mbits/second channel that the probed transmitter uses.

Average of Utilization Graph



This indicates that even when channel traffic is a relatively high 51 percent, the Ethernet protocol is able to carry essentially all of the submitted load. This also demonstrates the superiority of the carrier-sensing, collision-detection, and backoff strategies used by Ethernet over the less sophisticated methods used by the pure Aloha and CSMA protocols.

Congratulations! You have completed all of the Modeler tutorial lessons. By now, you should be able to build your own network model, collect statistics, run a simulation, and analyze the results on your own.

If you purchased additional modules, such as Multi-Vendor Import or ACE, continue with the tutorials that illustrate these capabilities. Return to the main tutorial menu and choose the desired tutorial from the list of available lessons.

From time to time, you may have questions about Modeler. Consult the documentation (**Help > Product Documentation**) first. You can also contact OPNET's Technical Support by choosing **Help > Web - Support Center**.

Good luck model building!