

CSMA/CD



1. Overview



5. Enhancing the model with Deference (CSMA)



2. Getting Started



6. Enhancing the model with collision detection and backoff (Ethernet)



3. Building the Aloha model



7. Analyzing the Ethernet results



4. Analyzing the Aloha results



This chapter presents a detailed set of examples which illustrate the modeling and analysis of 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.



1. Overview



methods.

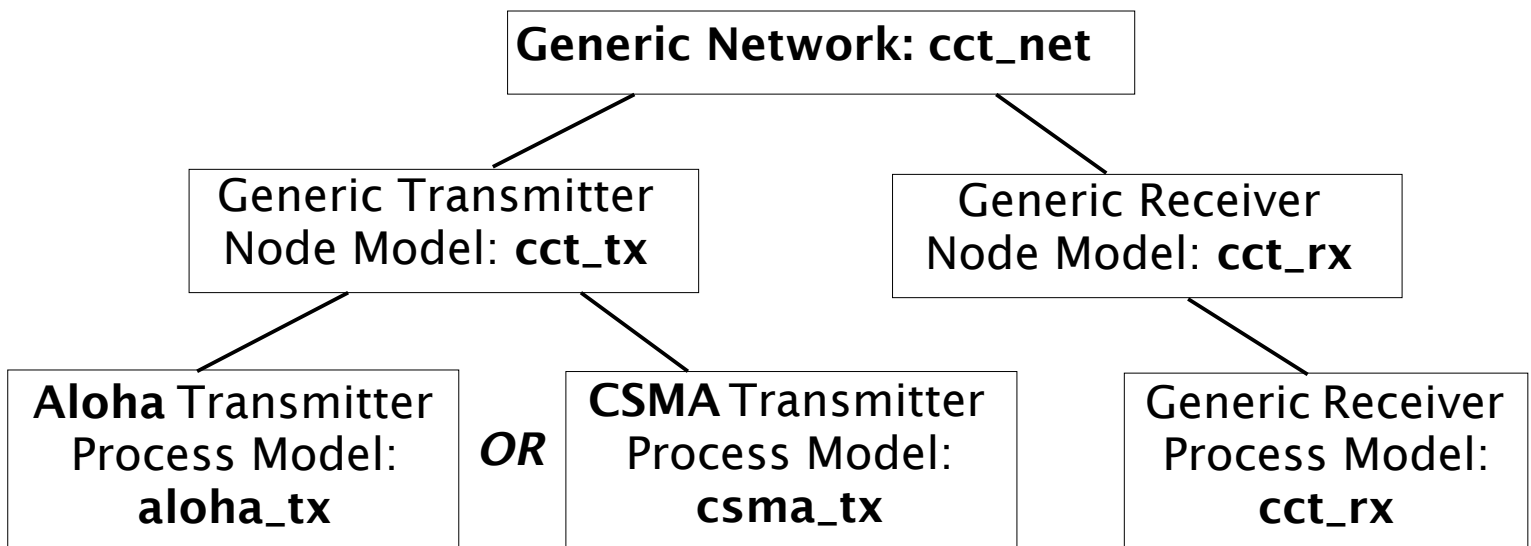
This lesson requires the construction of two models: an **Aloha** model and a **CSMA** model. The **Aloha** model will be designed first, since it is the simplest of the channel access

The task in this lesson is to design models which 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. Each method's performance will be compared against the others.

2. Getting Started

Before beginning the design of 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



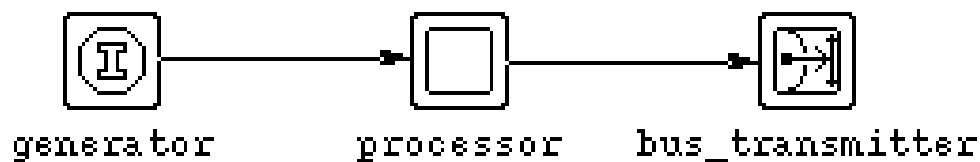
Designing the Generic Transmitter Node Model



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

The transmitter node must generate packets, process them, and send them on to the bus. This can be modeled simply using an ideal generator to generate packets, a 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

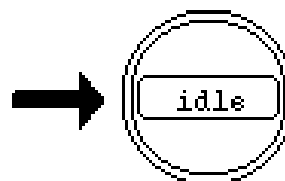


The Aloha transmitter process only has to receive packets from the generator and send them on to the transmitter.

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. Thus the process can begin the simulation in an unforced idle state where it waits for packets to arrive.

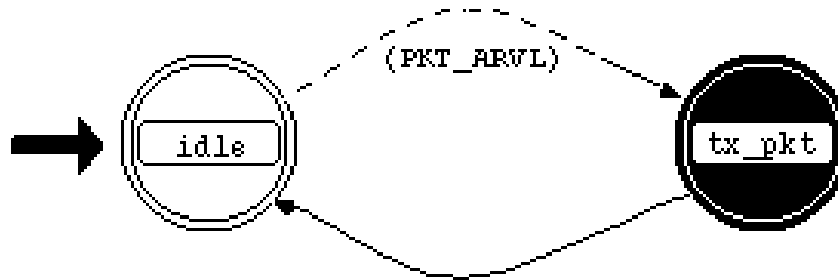
Because the FSM begins in an unforced state, the process needs to be activated with a begin simulation interrupt so that when the simulation starts, the FSM will execute the idle state and will be ready to transition with the first arriving packet.

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.

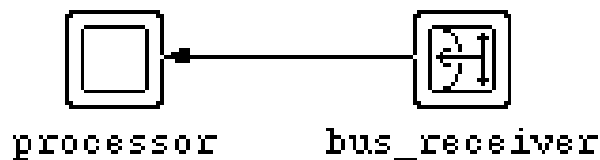
Designing the Generic Receiver Node Model



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. It consists only of a bus receiver and a processor module.

Conceptual Generic Receiver Node Model



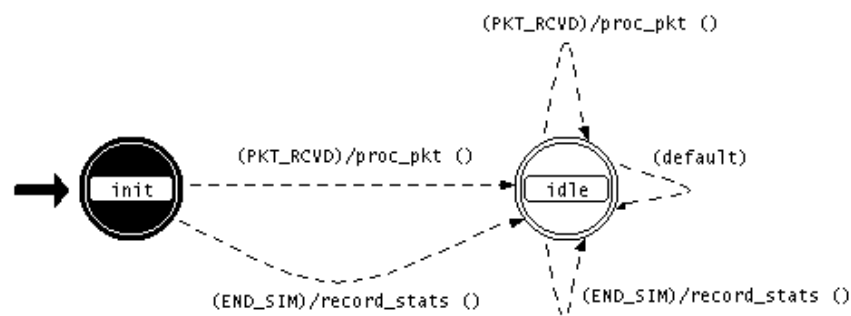
Designing the Generic Receiver Node Process Model



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 below. 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



3. Building the Aloha Model



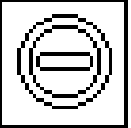
The **Aloha** process and node models will be created first. These models will then serve as the basis for an enhanced model that will be used to represent the **CSMA** system.

Building the Aloha model entails several steps, which are outlined below:

- Creating the Aloha transmitter node process model
- Creating a generic transmitter node model
- Creating a generic receiver node 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 to build is the transmitter process model:

- 1) Start OPNET if it is not already running.
- 2) Choose **New** from the **File** menu, select **Process Model** from the pull-down list, then click **OK**.
- 3) Using the **Create State** action button, place two states in the workspace. 
- 4) For the initial state, change the **name** attribute to **idle**. Leave the **status** as **unforced**.
- 5) For the other state, change the **name** attribute to **tx_pkt** and the **status** to **forced**.



Next, add the appropriate transitions between the states:

- 1) Draw the two transitions as shown.



- 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 ideal generator, so the macro does not need to determine which input stream received the packet.

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

- 1) Open the **header block** and type in definitions shown below. Save the changes, then close the edit pad when you are finished.

```
/* Input stream from ideal 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;
```

The symbolic constants **IN_STRM** and **OUT_STRM** in the header block will be used in calls to Kernel Procedures which 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 temporary variables and state variables:

- 1) Open the **temporary variable block** and enter the following declarations. Save the changes and close the edit pad when you are finished.

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

This establishes a temporary variable packet pointer. Since the process model acquires packets from the generator stream and immediately transmits them, it is not necessary to retain the packet pointer between process invocations.

- 1) Open the **state variable block** and enter the following information. Click **OK** to close the dialog box when you are finished.

Type	Name	Comments
int	max_packet_count	Number of packets to process.

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 idle state in its enter executives block:

- 1) Double-click on the top of the **idle** 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 (OPC_IMA_INTEGER, "max packet count",  
    &max_packet_count);
```

- 2) Save your changes, then close the text edit pad when you are finished.

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:

```
/* 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 ("Simulation ended when max packet count reached.",
        "", "", "");
}
```

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 the event-driven **PKT_ARVL** conditional is always applicable.

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

- 1) Choose **Simulation Attributes** from the **Interfaces** menu.
- 2) Enter an attribute into the dialog box table as shown:

Simulation Attributes			
Attribute Name	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:

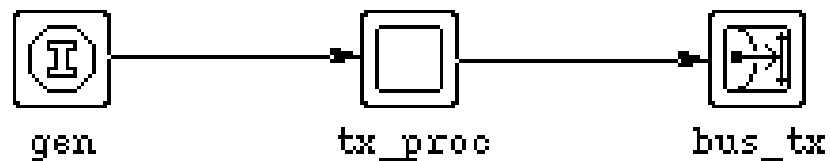
- 1) Choose **Process Interfaces** from the **Interfaces** menu:
- 2) Change the initial value of the **begsim intrpt** attribute to **“enabled”**.
- 3) Change the **Status** of all the attributes to **hidden**.

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

- 4) **Compile** the process model. Supply the name **<initials>_aloha_tx**.
- 5) When the process model is finished compiling, **close** the Process Model Editor.

Creating the Generic Transmitter Node Model

- 1) Select **New** from the **File** menu, and select **Node Model** from the resulting dialog box, then click **OK**.
- 2) Using the appropriate action buttons, create one packet generator module, one processor module, and one bus transmitter module. (Be sure to use a **bus** transmitter.)



- 3) For each module, open the attribute dialog box and change the **name** attribute as shown above.
- 4) Connect the modules with packet streams as shown above.

Note: Open the packet stream's dialog boxes to see that **stream 0** is automatically selected for input and output of the processor, 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 args** attribute, you must promote it so its value can be set more easily at simulation time.

- 1) Open the generator's attribute dialog box and set the **interarrival pdf** attribute to **exponential**.
- 2) Left-click on the **interarrival args** attribute, then click on the **Promote** button to promote the attribute.

(gen) Attributes	
Attribute	Value
name	gen
interarrival pdf	exponential
interarrival args	promoted

- 3) Close the attribute dialog box.

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

- 1) Open the processor's attribute dialog box and set the **process model** attribute to **<initials>_aloha_tx**.
- 2) Close the dialog box when you are finished.

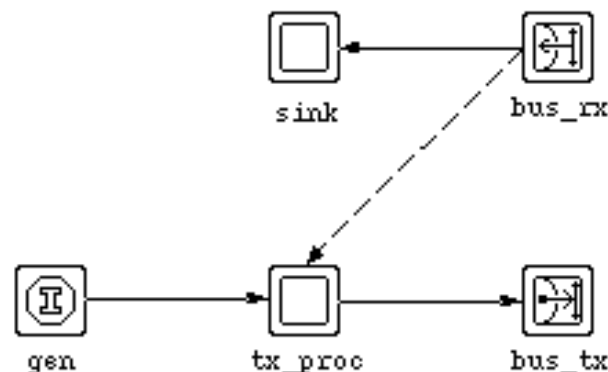
Enhancing the generic transmitter node model



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

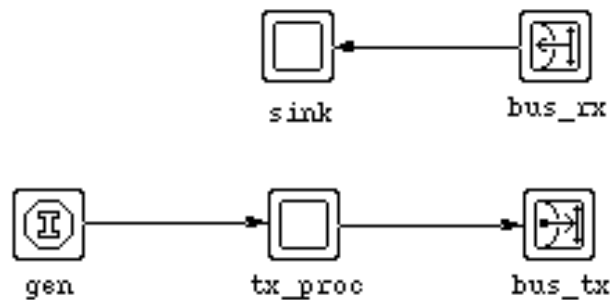
The enhancements will consist of a bus receiver module (to support the eventual full duplex capability of the CSMA protocol), and a sink processor to accept and destroy packets received by the receiver module. The enhancements also include an inactive (disabled) statistic wire which, when enabled in 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

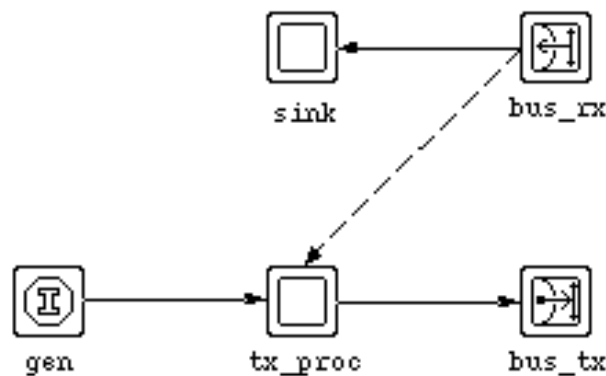


Add the following features to the node model:

- 1) Using the appropriate action buttons, add one processor module and one bus receiver module.



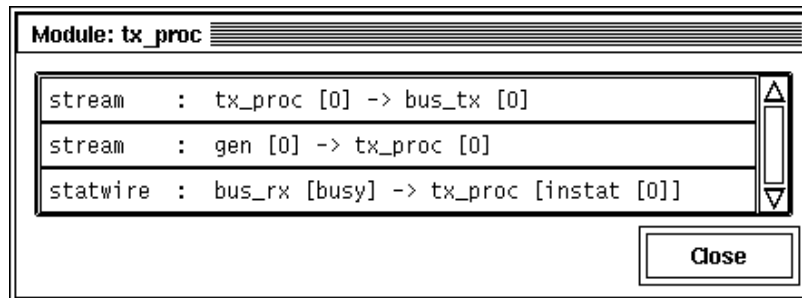
- 2) Change the **name** of the new processor module to **sink** and the **name** of the bus receiver to **bus_rx**.
- 3) Connect the new modules with a packet stream as shown.
- 4) Using the **Create Statistic Wire** action button, connect the **bus_rx** module with the **tx_proc** module.



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

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

- 1) Select **Show Module Connectivity** from the **Objects** menu. Place the cursor over the **tx_proc** module. The objects should be connected as follows:



If necessary, correct the streams by changing the **src stream** and **dest strm** attributes of the packet streams.

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

- 1) Select **Node Interfaces** from the **Interfaces** menu.
- 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.interarrival args**.

If you would like, add a comment to describe the process. When you are finished, click **OK** to save the changes. Finally, **save** the model as **<initials>_cct_tx** and **close** the editor.

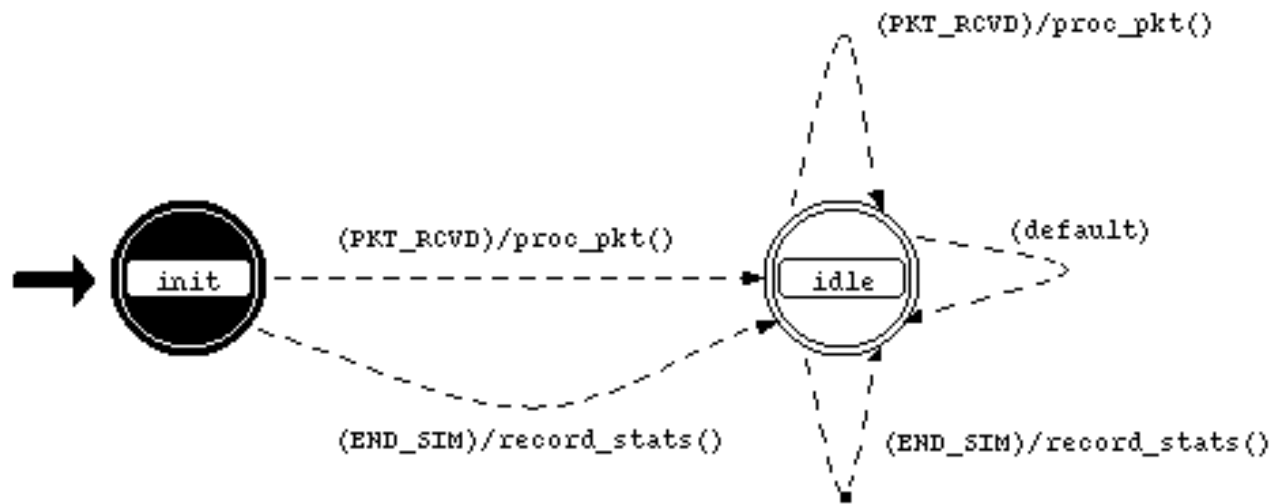
Creating the Generic Receiver Node Process Model



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) Select **New** from the **File** Menu, and select **Process Model** from the resulting dialog box and click **OK**.
- 2) Using the **Create State** action 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 transition states as shown:



- 1) For the first transition between the states, change the **condition** attribute to **PKT_RCVD** and the **executive** attribute to **proc_pkt()**.
- 2) For the second transition between the states, change the **condition** attribute to **END_SIM** and the **executive** attribute to **record_stats()**.
- 3) For the first transition from **idle** back to itself, change the **condition** attribute to **PKT_RCVD** and the **executive** attribute to **proc_pkt()**.
- 4) For the second transition from **idle** back to itself, change the **condition** attribute to **default**.
- 5) For the third 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 action button, open the **header block** and type in the definitions shown below.

```
/* 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;
```

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 delivered by the Simulation Kernel.

The global variable **subm_pkts** is established 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.

Define the following state variable:

- 1) Using the appropriate action button, open the **state variables block** and enter the following code:

state variables		
Type	Name	Comments
int	rcvd_pkts	Received packet counter.

The generic receiver process uses the **rcvd_pkts** variable to keep track of the number of valid received packets.

Next, open the **function block** and enter the following code:

```
/* This function gets the received packet, destroys it, */
/* and then logs the incremented received packet total. */
void proc_pkt ()
{
    Packet* in_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 packets */
    ++rcvd_pkts;
}

/* This function writes the end-of-simulation channel */
/* traffic and channel throughput statistics */
void record_stats ()
{
    /* Record final statistics */
    op_stat_scalar_write ("Channel Traffic G",
        (double) subm_pkts / op_sim_time ());
    op_stat_scalar_write ("Channel Throughput S",
        (double) rcvd_pkts / op_sim_time ());
}
```

As defined in the function block on the previous page, 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. Next, initialize the following variable in the init state:

- 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;
```

This state initializes the state variable used to count received packets.

Finally, you can define the process interfaces and compile the model:

- 1) Select **Process Interfaces** under the **Interfaces** menu.
- 2) Change the initial value of the **endsim intrpt** attribute to **enabled**.
- 3) Change the **Status** of all the attributes to **hidden**.

Attribute Name	Status	Initial Value
begsim intrpt	hidden	disabled
doc file	hidden	nd_module
endsim intrpt	hidden	enabled
failure intrpts	hidden	disabled

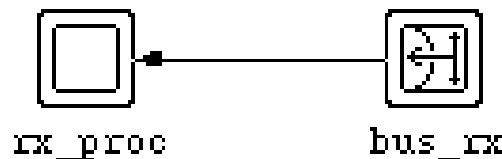
If you wish, add a comment to describe the process. When you are finished, click **OK** to save your changes, then compile the model:

- 4) Click on the **Compile Process Model** action button.
- 5) Supply the filename **<initials>_cct_rx** and click **OK**.
- 6) **Close** the process model editor.

Creating the Generic Receiver Node Model

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

- 1) Select **New** from the **File** Menu, and select **Node Model** from the resulting dialog box.
- 2) Using the appropriate action buttons, create one processor module and one bus receiver module. (Be sure to use a **bus** receiver.)



- 3) For each module, open the attribute dialog box and change the **name** attribute as shown.
- 4) Connect the modules with a packet stream as shown.

Note: 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**.

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

- 1) Select **Node Interfaces** from the **Interfaces** menu.
- 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**.

Attribute Name	Status	Initial Value
TIM source	hidden	none
altitude	hidden	0.0
condition	hidden	enabled
phase	hidden	0.0

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

- 1) **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 table below lists pipeline stage 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) Select **New** from the **File** Menu, and select **Link Model** from the resulting pull-down menu.
- 2) In the **Link Types** table, change the **Supported** value to **no** for the **ptsimp** and **ptdup** types.

Link Type	Supported	Palette Icon
ptsimp	no	
ptdup	no	
bus	yes	bus_1k
bus tap	yes	bus_tap

➔ This link model will only support the bus and bus tap types.

If you wish, add a comment to describe the link. When you are finished, click **OK** to close the dialog box, then **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.

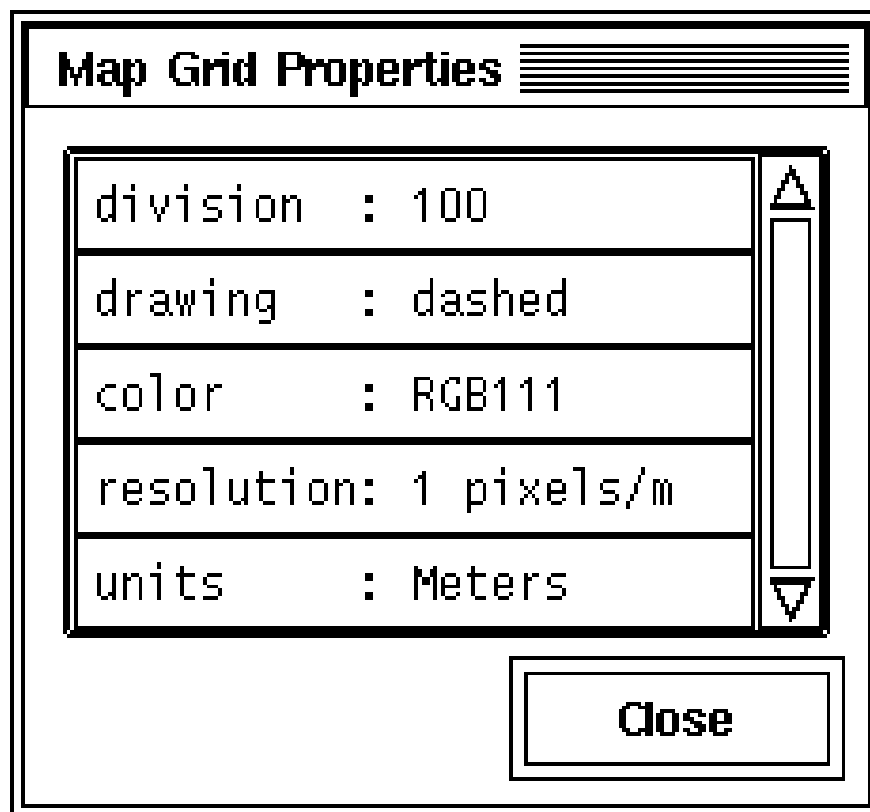
The analytical Aloha model assumes that packets are always introduced into the network at exponentially distributed interarrival times. However, this tutorial's network model will contain a finite number of nodes which each buffer packets until the previous outstanding transaction completes. To closely follow the analytical model's assumptions, a relatively large number of transmitter nodes must exist on the bus.

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

- 1) Select **New** from the **File** Menu, and select **Project** from the resulting pull-down menu. Name the project **<initials>_cct_network** and the scenario **aloha**, then click **OK**.
- 2) **Quit** the Startup Wizard
- 3) Open the **Object Palette**, then create a subnet node anywhere in the workspace. Name the subnet **cct_net**.

The next step is to change the properties within the subnet to a more usable scale:

- 1) Double-click on the subnet icon to move to the subnet view.
- 2) Select **Grid Properties** from the **View** menu.
- 3) Change the **units** attribute to **Meters**.
- 4) Change the **resolution** attribute to **1 pix/m**.
- 5) Change the **division** attribute to **100**.



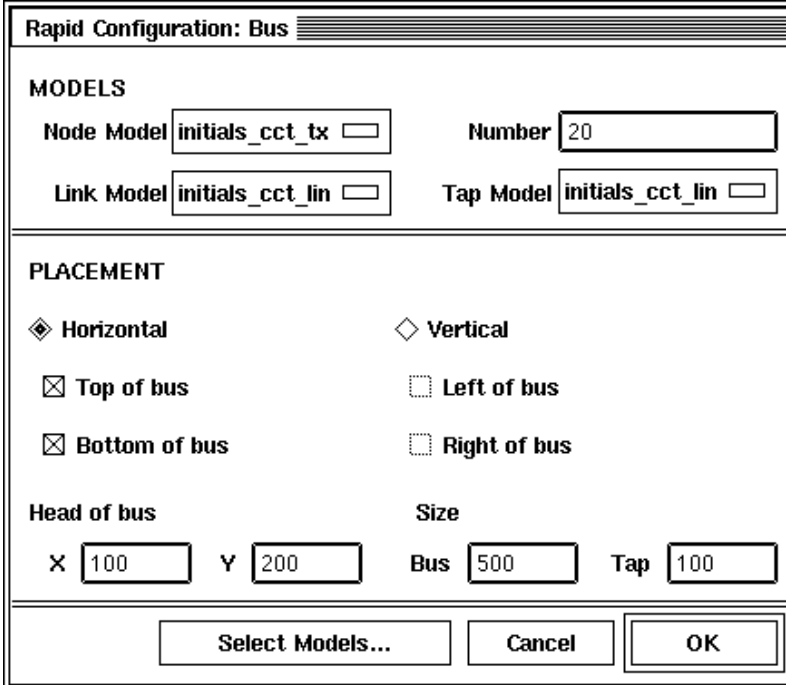
- 6) Left-click the **Close** button.

In order to easily build your network, you need a custom palette which contains the necessary objects for your network. To create the palette:

- 1) In the **Object Palette** window, 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.
- 3) Click on the **Node Models** button, then add **<initials>_cct_tx** and **<initials>_cct_rx** from the list of available node models. **Close** the dialog box when you are finished.
- 4) Click on the **Link Models** button, then add **<initials>_cct_link** from the list of available link models. **Close** the dialog box when you are finished.
- 5) Save the CML by clicking on the **Save** 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** CML is ready for use.

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

- 1) Click on the **Rapid Configuration** action button.
- 2) Select **bus** from the menu of available configurations, then click **OK**.
- 3) In the **Rapid Configuration** dialog box, set the following values:



The image shows a dialog box titled "Rapid Configuration: Bus". It is divided into two main sections: "MODELS" and "PLACEMENT".

MODELS

- Node Model:** A text box containing "initials_cct_tx".
- Number:** A text box containing "20".
- Link Model:** A text box containing "initials_cct_lin".
- Tap Model:** A text box containing "initials_cct_lin".

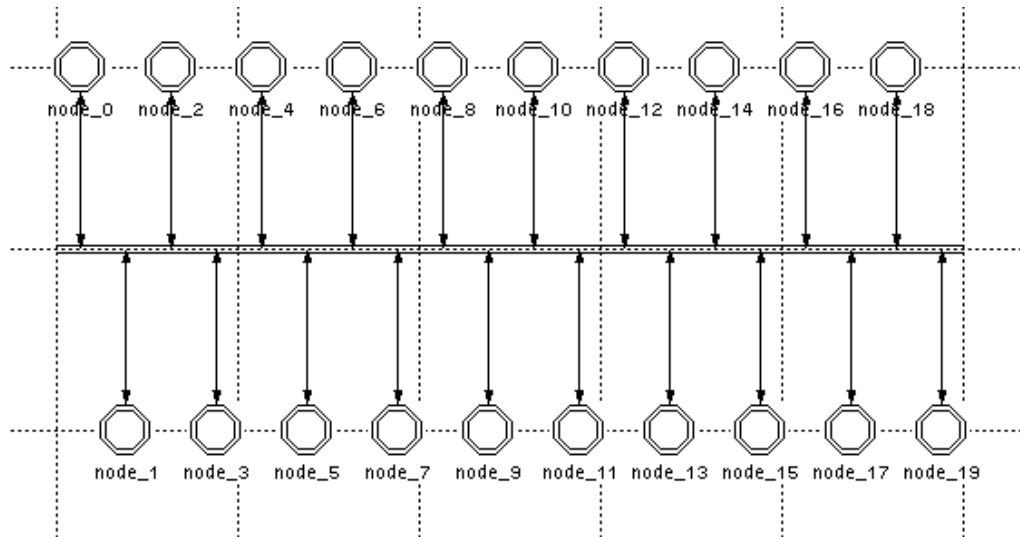
PLACEMENT

- Horizontal:** A radio button that is selected.
- Vertical:** A radio button that is not selected.
- Top of bus:** A checked checkbox.
- Left of bus:** An unchecked checkbox.
- Bottom of bus:** A checked checkbox.
- Right of bus:** An unchecked checkbox.
- Head of bus:**
 - X:** A text box containing "100".
 - Y:** A text box containing "200".
- Size:**
 - Bus:** A text box containing "500".
 - Tap:** A text box containing "100".

At the bottom of the dialog box are three buttons: "Select Models...", "Cancel", and "OK".

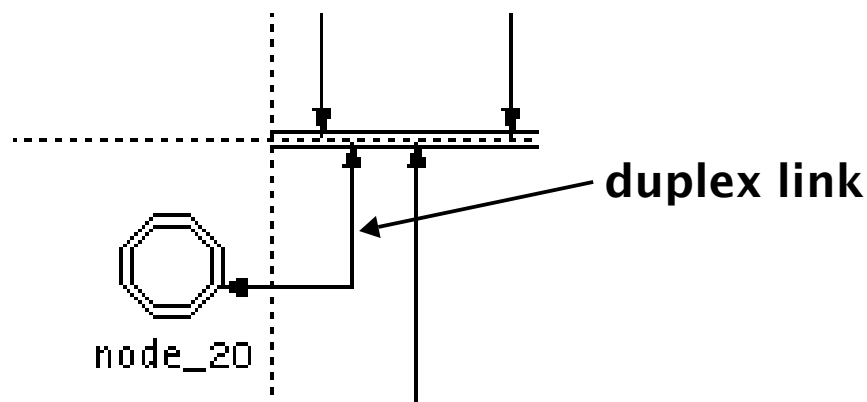
- 4) Click **OK** when all the values are entered.
- ➔ The network on the following page is drawn in the workspace.

Rapid Configuration creates the following bus network:



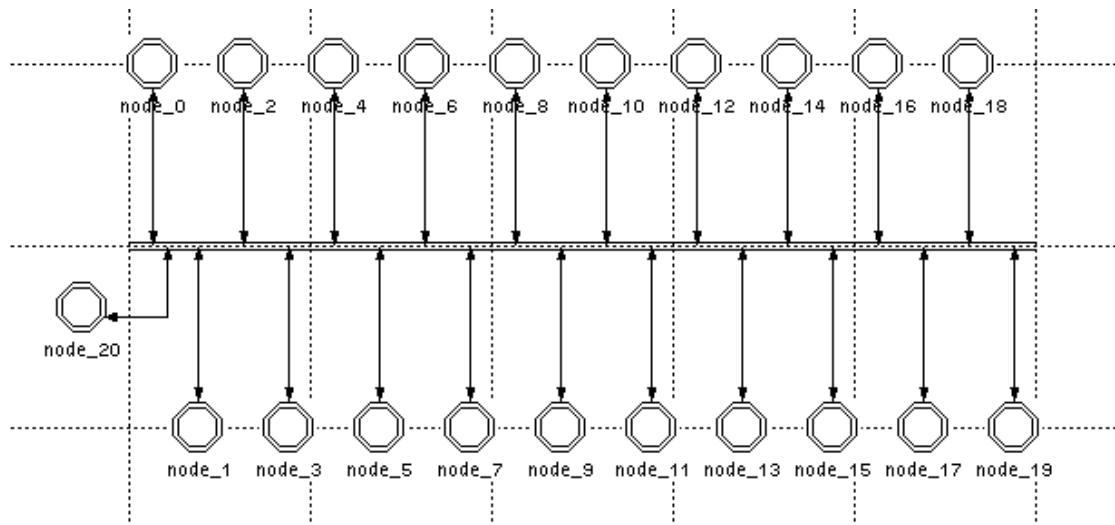
This network still needs a receiver node. To add this node and connect it to the network:

- 1) Click and drag the receiver node **<initials>_cct_rx** from the palette into the left side of the tool area.
- 2) Click on the **<initials>_cct_link** tap link in the palette, then draw a tap from the bus to the receiver node.



Note: You cannot draw a tap from the node to the bus.

The completed bus model looks like this:



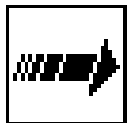
Finally, you can **save** the model (but do not exit the Project Editor) and close the object palette.

Executing the Aloha Simulation

Because this simulation produces scalar results, an output scalar file must be specified where these results accumulate from successive simulations. This must be done in the Simulation Configuration Editor, which can be reached by selecting **Configure Simulation (advanced)** from the **Simulation** menu in the Project Editor.

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 hence, different levels of throughput. Conclusions will be drawn from the results of twelve simulations, each with a different interarrival time value.

- 1) Select **Configure Simulation (advanced)** from the **Simulation** menu.
 - ➡ The Simulation Configuration Editor opens with a single simulation set object in the workspace.
- 2) Right-click to open the attribute dialog box for the simulation set.



scenario

There are a number of things that need to be changed in the simulation set:

- 1) Set the following attributes in the simulation set:

Attribute	Value
Scalar file	<initials>_cct_a
Seed	531
Duration	20000 seconds

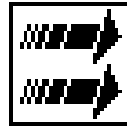
You need to add the **max packet count** attribute and the **interarrival args** attribute to the attribute table:

- 1) Click on the **Add** button just under the attribute table.
- 2) In the **Add Attribute** dialog box, add **max packet count** and **cct_net.*.gen.interarrival args**. Click **OK**.
- 3) Under **max packet count**, add the value **1000**.
- 4) Under **cct_net.*.gen.interarrival args**, add the values **1000, 200, 150, 100, 80, 50, 35, 30, 25, 20, 18, 15**.

(To enter multiple values for the interarrival args attribute, select it in the table, then click the **Values...** button and enter one value per row in the resulting dialog box.)

- 5) Click **OK** when you are finished.

Note that the icon for the simulation set object now looks like the one shown below, indicating that the sequence contains more than one run (one run for each value of **interarrival args**):



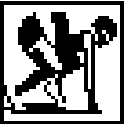
- 1) **Save** the simulation sequence.

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 which may already exist in a similarly named file, the output scalar file **<initials>_cct_a** must be deleted if it exists.

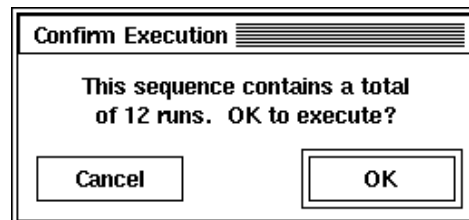
- 1) Select **Delete Models** from the **File** menu.
 - ➡ A list of deletable file types appears.
- 2) Select the **Output Scalars** item.
 - ➡ A list of available **output scalar files** appears.
- 3) If the list contains the output scalar file **<initials>_cct_a**, select the entry to delete the file.
- 4) **Close** the open dialog boxes.

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

- 1) Click on the **Execute Simulation Sequence** action button.



- 2) Left-click **OK** at 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.



- ➔ 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 such as that shown below:

```
Simulation terminated by process (csma_tx) at modul
Simulation ended when max packet count exceeded.
-----
Simulation Completed.
SIM_T (748.074), REAL_T (76.0) sec.
EV (86800), AV_EV/S (1142), AV_EFF (9.84563)
```

- 3) When the simulations are complete, **close** the editor.

4. Analyzing the Aloha Results



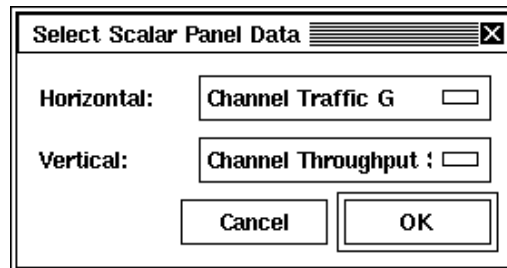
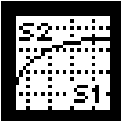
Aloha channel performance can be measured according to the number of successfully received packets as a function of the packets submitted, regardless of whether the packets are original or retransmitted. In this network, **channel throughput** is a typical measurement of network performance.

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 in the Analysis Configuration Editor.

- 1) In the Project Editor, choose **View Results (Advanced)** from the **Results** menu.
 - ➔ The Analysis Configuration Editor opens.
- 2) Select **Load Output Scalar File** from the **File** menu.
- 3) Select **<initials>_cct_a** from the list of available files.

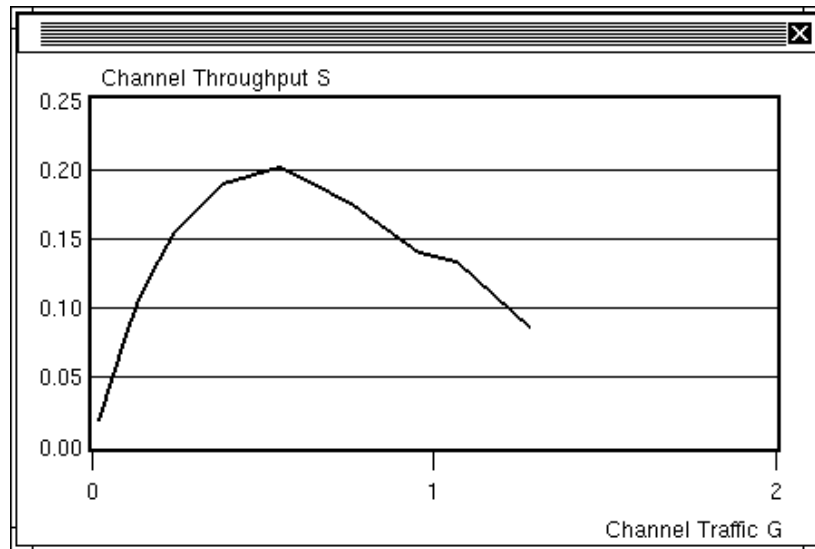
Draw the scalar panel:

- 1) Click on the **Create Scalar Panel** action 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 pops up.



- 3) Click **OK**.
- ➡ The graph of the scalar panel appears in the workspace (the graph is shown on the next page).

The scalar graph panel should resemble the one below:



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 $S_{max} = 1/2e^{-1} \approx 0.18$. 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**.

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

It should be noted again that 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.