INTRODUCTION TO C-PROGRAMMING

Outline

INTRODUCTION TO NS3

Logging should be preferred for debugging information, warnings, error messages, or any time you want to easily get a quick message out of your scripts or models.
ns-3 takes the view that all of these verbosity levels are useful and we provide a selectable, multi-level approach to message logging. Logging can be disabled completely, enabled on a component-by-component basis, or enabled globally; and it provides selectable verbosity levels. The ns-3 log module provides a straightforward, relatively easy to use way to get useful information out of your simulation.

ASCII Tracing

ns-3 provides helper functionality that wraps the low-level tracing system to help you with the details involved in configuring some easily understood packet traces. If you enable this functionality, you will see output in a ASCII files - thus the name. For those familiar with ns-2 output, this type of trace is analogous to the out.tr generated by many scripts.

```
AsciiTraceHelper ascii;
pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("myfirst.tr"));
```

The outside call, to EnableAsciiAll(), tells the helper that you want to enable ASCII tracing on all point-to-point devices in your simulation; and you want the (provided) trace sinks to write out information about packet movement in ASCII format.

Parsing Ascii Traces

Each line in the file corresponds to a trace event. In this case we are tracing events on the transmit queue present in every point-to-point net device in the simulation. The transmit queue is a queue through which every packet destined for a point-to-point channel must pass. Note that each line in the trace file begins with a lone character (has a space after it). This character will have the following meaning:

1. +: An enqueue operation occurred on the device queue;
2. -: A dequeue operation occurred on the device queue;
3. d: A packet was dropped, typically because the queue was full;
4. r: A packet was received by the net device

PCAP Tracing

The ns-3 device helpers can also be used to create trace files in the .pcap format. The acronym pcap (usually written in lower case) stands for packet capture, and is actually an API that includes the definition of a .pcap file format. The most popular program that can read and display this format is Wireshark (formerly called Ethereal). However, there are many traffic trace analyzers that use this packet format. We encourage users to exploit the many tools available for analyzing pcap traces. In this tutorial, we concentrate on viewing pcap traces with tcpdump.

title

The code used to enable pcap tracing is a one-liner.
pointToPoint.EnablePcapAll ("myfirst");

Notice that we only passed the string "myfirst," and not "myfirst.pcap" or something similar. This is because the parameter is a prefix, not a complete file name. The helper will actually create a trace file for every point-to-point device in the simulation. The file names will be built using the prefix, the node number, the device number and a ".pcap" suffix. In our example script, we will eventually see files named "myfirst-0-0.pcap" and "myfirst-1-0.pcap" which are the pcap traces for node 0-device 0 and node 1-device 0, respectively.

Using the Logging Module

1. There are currently seven levels of log messages of increasing verbosity defined in the system.
2. LOG_ERROR - Log error messages (associated macro: NS_LOG_ERROR);
3. LOG_WARN - Log warning messages (associated macro: NS_LOG_WARN);
4. LOG_DEBUG - Log relatively rare, ad-hoc debugging messages (associated macro: NS_LOG_DEBUG);
5. LOG_INFO - Log informational messages about program progress (associated macro: NS_LOG_INFO);
6. LOG_FUNCTION - Log a message describing each function called (two associated macros: NS_LOG_FUNCTION, used for member functions, and NS_LOG_FUNCTION_NOARGS, used for static functions);
7. LOG_LOGIC - Log messages describing logical flow within a function (associated macro: NS_LOG_LOGIC);
8. LOG_ALL - Log everything mentioned above (no associated macro).

Key Abstractions

Node

n Internet jargon, a computing device that connects to a network is called a host or sometimes an end system. Because ns-3 is a network simulator, not specifically an Internet simulator, we intentionally do not use the term host since it is closely associated with the Internet and its protocols. Instead, we use a more generic term also used by other simulators that originates in Graph Theory - the node.

In ns-3 the basic computing device abstraction is called the node. This abstraction is represented in C++ by the class Node. The Node class provides methods for managing the representations of computing devices in simulations. You should think of a Node as a computer to which you will add functionality. One adds things like applications, protocol stacks and peripheral cards with their associated drivers to enable the computer to do useful work. We use the same basic model in ns-3.

Application

In ns-3 there is no real concept of operating system and especially no concept of privilege levels or system calls. We do, however, have the idea of an application. Just as software applications run on computers to perform tasks in the ?real world,? ns-3 applications run on ns-3 Nodes to drive simulations in the simulated world.

In ns-3 the basic abstraction for a user program that generates some activity to be simulated is the application. This abstraction is represented in C++ by the class Application. The Application class provides meth- ods for managing the representations of our version of user-level applications in simulations. Developers are ex- pected to specialize the Application class in the object-oriented programming sense to create new applications. In this tutorial, we will use specializations of class Application called UdpEchoClientApplication and UdpEchoServerApplication. As you might expect, these applications compose a client/server application set used to generate and echo simulated network packets

Channel

In the real world, one can connect a computer to a network. Often the media over which data flows in these networks are called channels. When you connect your Ethernet cable to the plug in the wall, you are connecting your computer to an Ethernet communication channel. In the simulated world of ns-3, one connects a Node to an object representing a communication channel. Here the basic communication subnetwork abstraction is called the channel and is represented in C++ by the class Channel.

The Channel class provides methods for managing communication subnetwork objects and connecting nodes to them. Channels may also be specialized by developers in the object oriented programming sense. A Channel specialization may model something as simple as a wire. The specialized Channel can also model things as complicated as a large Ethernet switch, or three-dimensional space full of obstructions in the case of wireless networks.

We will use specialized versions of the Channel called CsmaChannel, PointToPointChannel and WifiChannel in this tutorial. The CsmaChannel, for example, models a version of a communication subnetwork that implements a carrier sense multiple access communication medium. This gives us Ethernet-like functionality.

Net Device

It used to be the case that if you wanted to connect a computer to a network, you had to buy a specific kind of network cable and a hardware device called (in PC terminology) a peripheral card that needed to be installed in your computer. If the peripheral card implemented some networking function, they were called Network Interface Cards, or NICs. Today most computers come with the network interface hardware built in and users don?t see these building blocks.

A NIC will not work without a software driver to control the hardware. In Unix (or Linux), a piece of peripheral hard- ware is classified as a device. Devices are controlled using device drivers, and network devices (NICs) are controlled using network device drivers collectively known as net devices. In Unix and Linux you refer to these net devices by names such as eth0.

NetDevice

In ns-3 the net device abstraction covers both the software driver and the simulated hardware. A net device is "installed" in a Node in order to enable the Node to communicate with other Nodes in the simulation via Channels. Just as in a real computer, a Node may be connected to more than one Channel via multiple NetDevices.

The net device abstraction is represented in C++ by the class NetDevice. The NetDevice class provides methods for managing connections to Node and Channel objects; and may be specialized by developers in the object-oriented programming sense. We will use the several specialized versions of the NetDevice called CsmaNetDevice, PointToPointNetDevice, and WifiNetDevice in this tutorial. Just as an Ethernet NIC is designed to work with an Ethernet network, the CsmaNetDevice is designed to work with a CsmaChannel; the PointToPointNetDevice is designed to work with a PointToPointChannel and a WifiNetNevice is designed to work with a WifiChannel.

Topology Helpers

In a real network, you will find host computers with added (or built-in) NICs. In ns-3 we would say that you will find Nodes with attached NetDevices. In a large simulated network you will need to arrange many connections between Nodes, NetDevices and Channels. Since connecting NetDevices to Nodes, NetDevices to Channels, assigning IP addresses, etc., are such common tasks in ns-3, we provide what we call topology helpers to make this as easy as possible. For example, it may take many distinct ns-3 core operations to create a NetDevice, add a MAC address, install that net device on a Node, configure the node's protocol stack, and then connect the NetDevice to a Channel. Even more operations would be required to connect multiple devices onto multipoint channels and then to connect individual networks together into internetworks. We provide topology helper objects that combine those many distinct operations into an easy to use model for your convenience.


THANK YOU