ANEXO D

CONSTRUCCIÓN DE TOPOLOGÍAS EN ns-3

Topología Point To Point

Este script está ubicado en la carpeta “examples” y se llama “first.cc”

ns-3 esta licenciado con licencia publica general GNU. Se puede ver las condiciones legales GNU en el encabezado de cada archivo de las distribuciones de ns-3. Frecuentemente se ve información acerca del copyright de las instituciones involucradas en el proyecto y el autor arriba del texto GPL.

```
/*
 * This program is free software; you can redistribute it and/or modify
 * it under the terms of the GNU General Public License version 2 as
 * published by the Free Software Foundation;
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA  02111-1307 USA
 */
```

Añadir módulos

Para ayudar a los usuarios lidiar con una gran numero de archivos “include” en el sistema, se agrupan “inlcludes” de acuerdo a grupos relativamente grandes. Se provee un solo achivo “include” que recursivamente cargará todos los archivos “include” usados en cada modulo. En vez de buscar exactamente que cabecera se necesita, y posiblemente tener que conseguir un cierto número de dependencias, se da la capacidad de dar un grupo de archivos en una gran granularidad. Este no es la mejor solución pero hace que escribir scripts sea mucho más fácil.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
```

Cada uno de los archivos “include” de ns-3 se guarda en un directorio llamado ns3; durante el proceso de compilación para ayudar evitar colisiones de nombre de archivo “include”. El archivo “ns3/core-module.h” corresponde al modulo ns-3 que se encontrará en el directorio scr/core de la distribución que se tenga descargado. Si se da un vistazo a este directorio se encontrarán un gran número de archivos de cabecera. Cuando se compila, el “Waf” guardará archivos de cabecera públicos en un directorio de ns-3 en el directorio apropiado build/debug o build/optimized dependiendo de la configuración. El “Waf” generará automáticamente también un archivo “include” del modulo para cargar todos los archivos de cabecera públicos.

Namespace Ns-3

La siguiente línea en el script es una declaración de namespace.

```
using namespace ns3;
```

El proyecto ns-3 implementa en un namespace llamado “ns-3”. Este agrupa todas las declaraciones relacionadas con ns-3 en un ámbito fuera del namespace global, el cual se espera que ayude con la integración con otro código. La declaración C++ “using” introduce el namespace de ns-3 dentro la región declarativa (global) actual. Esta es una manera elegante de decir que después de esta declaración, no se tendrá que digitar el operador de resolución de ambito “ns3::” antes de todo el código ns-3 para poder usarlo.

Logging

La siguiente línea de código es la siguiente,

```
NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");
```

Aquí se define un componente de registro con un nombre específico. Este macro debería usarse en cada archivo en el cual se quiere usar el macro NS_LOG. Este macro define un componente “log” nuevo el cual puede ser más adelante selectivamente habilitado o deshabilitado con las funciones “ns3::LogComponentEnable” y “ns3::LogComponentdisable” o con la “environment variable”.

Función Main

```
int
main (int argc, char *argv[])
{
```

Este es solo la declaración de la función “main” del programa. Como en cualquier programa de C++, se necesita definir una función principal que es la primera que se corre. No hay nada especial aquí. El script en ns-3 es solo en programa en C++.

Las siguientes dos líneas del script se usan para habilitar dos componentes de registro que se construyen dentro de las aplicaciones de Cliente y Servidor. Esto se ve en la aplicación como impresiones de mensajes cada vez que un paquete es enviado o recibido durante la simulación.

```
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);
```

“Topology Helpers”

“Node Container”

Las siguientes dos líneas crean los objetos “Node” dentro de ns-3 que representarán los computadores en la simulación.

```
NodeContainer nodes;
nodes.Create (2);
```

La primera línea solo declara un “Node Container” (Contenedor de Nodos) el cual se llama “nodes”. La segunda línea calla al método “Create” en los objetos “nodes” y le pide al contenedor crear dos nodos.

Los nodos por si solos no hacen nada. El siguiente paso en la construcción una topología es conectar los nodos dentro de la red. La forma más simple de red podría ser en enlace simple “point to point” entre dos nodos.

“PointToPointHelper”

Se va a construir un enlace “PointToPoint”, se hace uso de los objetos “topology helpers” para hacer el trabajo a bajo nivel requerido para crear el enlace. Es importante recordar que dos de las abstracciones son el “NetDevice” y el “Channel”. En el mundo real, estos términos meramente corresponden a tarjetas periféricas y cables de red. Típicamente estas dos cosas son íntimamente relacionadas y no se puede esperar, por ejemplo, que dispositivos Ethernet se conecten a medios inalámbricos. Por consiguiente se usa un “PontToPointHelper” para configurar y conectar los objetos “PoinToPointNetDevice” y “PontToPointHelperChannel” en el script.

```
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
pointToPoint.SetChannelAttribute ("Delay", StringValue ("2ms"));
```

La primera linea crea la instancia del objeto “PointToPointHelper”. Desde una perspectiva de alto nivel. La segunda línea le dice al objeto “PointTopointHelper” que use el valor “5Mbps”

(cinco megabits por segundo) como velocidad de datos cuando este crea un objeto "PontToPointDevice".

Desde una perspectiva detallada, el string "DataRate" corresponde a lo que se llama un atributo de "PoinToPointNetDevice". Similar a lo de "DataRate", el atributo "Delay" que está asociado con el "PointToPointChannel". Y la tercera línea le dice al "PontToPointHelper" que use el valor de "2ms" (dos milisegundos) como el valor de retraso de transmisión de cada canal "point to point" que subsecuentemente se crea.

"NetDeviceContainer"

En este punto del script, se tiene un "NodeContainer" que contiene dos nodos. También se tiene un "PontToPointHelper" que está presto y listo para crear "PontToPointNetDevices" y objetos "PontToPointChannel" entre los nodos. Ahora se necesita tener una lista de los objetos "NetDevice", entonces se usa un "NetDeviceContainer" para administrar la tarea justo como se usó un "NodeContainer" para administrar los nodos que se crearon. Las siguientes dos líneas de código terminan la configuración de los dispositivos y el canal. La primeara línea declara el contenedor del dispositivos y la segunda realiza el trabajo pesado. El método "install" del "PointToPointHelper" toma un "NodeContainer" como parámetro. Internamente, un "NetDeviceContainer" se crea y los dos "PointToPointNetDevices" son conectados. Cuando los objetos son creados por el "PointTopointHelper", los "Atributes" previamente definidos en el ayudante son usados para inicializar los "Atributes" correspondiente los objetos creados.

```
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
```

Después de ejecutar la llamada "pointTopoint.Install (nodes)" se tendrán dos nodos, cada uno con un dispositivo de red point to point y un canal también point to point entre los nodos. Los dispositivos serán configurados a una tasa de transmisión de datos de cinco megabits por segundo sobre un canal el cual tiene un delay de transmisión de dos milisegundos.

"InternetStackHelper"

Ahora se necesita instalar el Stack de protocolos en los nodos. Las dos líneas de código siguientes hacen esta tarea. El "InternetStackHelper" es un ayudante de topología que se encarga de instalar el stack de protocolos en los nodos. El método "install" toma un objeto "NodeContainer" como parámetro. Cuando este se ejecuta, este instalará un Stack de internet (TCP,UDP,IP,etc.) en cada unos de los nodos del contenedor.

```
InternetStackHelper stack;
stack.Install (nodes);
```

“Ipv4AddressHelper”

A continuación se necesita asociar a los dispositivos de los nodos a las direcciones IP. Existe un ayudante de topología que maneja la asignación de direcciones IP. El único API visible al usuario es para fijar la dirección IP base y la máscara de red para usarse cuando se ejecute la asignación real de direcciones IP (lo cual es hecho a bajo nivel dentro del ayudante). En las dos líneas de código siguientes se declaran un objeto ayudante de direcciones y le dicen que debería empezar a asignar las direcciones de la red “10.1.1.0” usando la máscara 255.255.255.0 para definir los bits asignables. Por defecto las direcciones asignadas empezarán en uno y se incrementarán de uno un uno, de manera que la primera dirección asignada de esta base será 10.1.1.1, seguida de 10.1.1.2, etc. El sistema de ns-3 a bajo nivel recuerda todas las direcciones asignadas y generará un error si accidentalmente se asigna la misma dirección dos veces (lo cual es un muy difícil de depurar).

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
```

La siguiente línea de código ejecuta la asignación, en ns-3 se hacen asociaciones entre direcciones IP y los dispositivos usando un objeto “Ipv4Interface”.

```
Ipv4InterfaceContainer interfaces = address.Assign (devices);
```

Ahora se tiene una red point to point construida, con stacks de protocolos y direcciones IP asignadas. Los que se necesita en este punto es que la aplicación genere tráfico.

“Applications”

Otra de la abstracciones del sistema ns-3 es la “Application”. En este script se usan dos especializaciones de la clase “Application” llamadas “UpdEchoServerApplication” y “UdpEchoClientApplication”.

“UpdEchoServerApplication”

Las líneas de código siguientes se usan para iniciar una aplicación UDP de eco en el servidor en uno de los nodos que previamente se crearon. En la primera línea de código se declara el “UpdEchoServerApplication”. Como es usual, esta no es una aplicación en sí misma, en realidad es un objeto usado para ayudar a crear la aplicación real. Uno de las convenciones es pasar los Atributos requeridos a los constructores de ayudantes. En este caso, el ayudante no puede hacer nada útil a menos que se la pase un número de puerto como parámetro al constructor. El constructor, en cambio, simplemente fija los atributos con el valor pasado. Si se quiere se puede fijar el “Atribute” “Port” con otro valor después usando el método “SetAttribute”.

```
UdpEchoServerHelper echoServer (9);
```

```
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1));
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

Similar a muchos de los otros objetos ayudantes, el objeto “UdpEchoServerHelper” tiene un método “Install”. Es la ejecución de este método que realmente permite que la instancia de la aplicación fundamental de eco del servidor sea creada y agregada al nodo. Interesantemente, el método “Install” toma un “NodeContainer” como parámetro justo como los otros métodos “Install” que se han visto. Ahora “EchoServer.Install” va a instalar una “UpdEchoServerApplication” en el nodo encontrado en el número de índice “1” del “NodeContainer” que se usa para administrar los nodos. El método “Install” retornará un contenedor que tiene los punteros a todas las aplicaciones (una en este caso, dado que se paso un “NodeContainer” conteniendo un solo nodo) creadas por el ayudante.

Las aplicaciones requieren un tiempo para empezar a generar tráfico y pueden tener un tiempo opcional para parar. En ns-3 se tienen los dos. Estos tiempos son fijados usando los métodos “Start” y “Stop” del “ApplicationContainer”. Estos métodos toman parámetros de tiempo. En este caso, se usa una secuencia de conversión de C++ explicita para tomar el doble 1.0 de C++ y convertirlo en un objeto de tiempo ns-3 usando un llamado a segundos. Hay que tener en cuenta que las reglas para la conversión pueden ser controladas por el autor, y C++ tiene sus propias reglas, entonces no siempre se puede asumir que los parámetros serán correctamente convertidos. Las dos líneas siguientes harán que la aplicación de eco en el servidor empiece en el segundo uno dentro de la simulación y pare en el segundo diez dentro de la simulación. Por consiguiente la simulación durará al menos diez segundos.

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

“UpdEchoClientApplication”

La aplicación de eco del cliente se configura con un método substancialmente similar al método que usa el servidor. Hay una “UdpEchoClientApplication” fundamental que es administrado por un “UdpEchoClientHelper”.

```
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9);
echoClient.SetAttribute ("MaxPackets", UintegerValue (1));
echoClient.SetAttribute ("Interval", TimeValue (Seconds (1.)));
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));

ApplicationContainer clientApps = echoClient.Install (nodes.Get (0));
clientApps.Start (Seconds (2.0));
```

```
clientApps.Stop (Seconds (10.0));
```

Para el cliente de eco, sin embargo, se necesitan definir diferentes atributos. Los dos primeros atributos se fijan durante la construcción del “UdpEchoClientApplication”. Se pasan los parámetros que se usan (internamente para el ayudante) para configurar los atributos de dirección remota y del puerto.

Es importante recordar que se usa un “Ipv4InterfaceContainer” para mantenerse al tanto de las direcciones IP que se asignan a los dispositivos. La interface cero en el contenedor de interfaces va a corresponder a la dirección IP del nodo cero en el contenedor de nodos. La interface uno en el contenedor de interfaces corresponde a la dirección IP del nodo uno en el contenedor de nodos. Entonces en la primera línea de código (de más arriba), se están creando el ayudante y diciéndole que configure la dirección remota del cliente para que sea la dirección IP asignada al nodo en el cual reside el servidor. También se le dice que arregle los paquetes para que sean enviados por el puerto nueve.

El atributo “MaxPackets” le dice al cliente el máximo número de paquetes que se permiten enviar durante la simulación. El atributo “Interval” le dice al cliente que tanto tiempo esperar entre paquetes, y el atributo “PacketSize” le dice al cliente que tan grandes deberían ser los payloads de los paquetes. Con esta combinación particular de atributos, se le está diciendo al cliente que envíe un paquete de 1024 bytes.

Justo como en el caso del servidor de eco, se le dice al cliente de eco que empieza y para, pero aquí se le dice al cliente que empiece un segundo después de que el servidor este habilitado (a los dos segundos dentro de la simulación).

“Simulator”

Ahora solo se necesita correr la simulación. Esto se hace utilizando la función global “Simulator::Run”.

```
Simulator::Run ();
```

Cuando esta función es llamada, el sistema empezará a buscar a través de la lista de eventos programados y los ejecutará. Primero se correrá el evento al segundo 1.0, el cual habilita la aplicación de eco de servidor (este evento puede, a su vez, programar muchos otros eventos). Después se correrá el evento para t=2.0 segundos el cual iniciará la aplicación de echo del Cliente. De nuevo, esto puede programar otros eventos más. La implementación del evento de inicio en la aplicación de eco del Cliente iniciará la fase de transferencia de datos del simulador enviando un paquete al servidor.

```
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (10.0));
```

```
...
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (10.0));
```

El acto de enviar el paquete al servidor deparará una cadena de eventos que serán automáticamente programados invisiblemente para el usuario y el cual ejecutará los mecanismos de eco de paquetes de acuerdo a varios parámetros diferentes de tiempo que se han configurado en el script.

Eventualmente, dado que solo se envía un paquete (MaxPackets Attribute se fijo en uno), la cadena de eventos disparada por una sola solicitud de eco de cliente disminuirá y la simulación se volverá inactiva. Una vez esto pase, los eventos restantes serán los eventos de parada para el servidor y para el cliente. Cuando estos se ejecutan, no hay mas eventos para procesar y la función “Simulator::Run” retorna. En este punto la simulación esta completa.

Ahora lo que queda es limpiar. Esto es llevado a cabo llamando a la función global “Simulator::Destroy”. Finalmente las líneas restantes del script hacen esto.

```
  Simulator::Destroy ();
  return 0;
}
```

“Building The Script”

Ahora se necesita pasar el script dentro del directorio “scratch” y él automáticamente será compilado si se corre el “Waf”.

```
cd ../..
cp examples/tutorial/first.cc scratch/myfirst.cc
```

```
./waf
```

Deberían verse mensajes reportando que el script se compilo satisfactoriamente.

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
[614/708] cxx: scratch/myfirst.cc -> build/debug/scratch/myfirst_3.o
[706/708] cxx_link: build/debug/scratch/myfirst_3.o -> build/debug/scratch/myfirst
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (2.357s)
```

Ahora ya se puede correr el script (si se compila el programa en el directorio “scratch” se tiene que correrlo fuera del directorio scratch“)

```
./waf --run scratch/myfirst
```

Se debería ver la siguiente salida

```
Waf: Entering directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
Waf: Leaving directory `/home/craigdo/repos/ns-3-allinone/ns-3-dev/build'
'build' finished successfully (0.418s)
Sent 1024 bytes to 10.1.1.2
Received 1024 bytes from 10.1.1.1
Received 1024 bytes from 10.1.1.2
```

Aquí se ve que el sistema se asegura que el archivo ha sido compilado y después lo corre. Se ve que el componente de registro en el cliente de eco indica que se ha enviado un paquete de 1024 bytes hacia el servidor de eco en la dirección IP 10.1.1.2. También se puede ver que el componente de registro en el servidor de eco ha recibido los 1024 bytes desde la dirección 10.1.1.1. El servidor de eco silenciosamente genera el eco de vuelta hacia el cliente, se puede ver el paquete ha sido recibido de vuelta en el cliente.