

Paradigmas de programación

Francisco morales

Laboratorio Mesa

Documentación NS3

Introducción

En este documento, se explicará el proceso de instalación y configuración de Ns-3 en un entorno Linux, utilizando una máquina virtual. Comenzaremos con los pasos para descargar Ns-3 y preparar el entorno, incluyendo las herramientas necesarias. Luego, se analizará el programa `first.cc`, donde se explorará la interacción entre nodos dentro de una red simulada, mostrando cómo se establecen las conexiones y se gestionan los intercambios de datos. Finalmente, se concluirá con una reflexión sobre el paradigma orientado a eventos, abordando cómo este enfoque resulta adecuado para simular las interacciones en redes.

Procedimiento de instalación Ns3

1. Estar en alguna distribución de Linux. Existen varios métodos, el empleado en esta ocasión es una máquina virtual (Oracle Virtual Box)
2. El paso siguientes es descargar Ns3 desde:

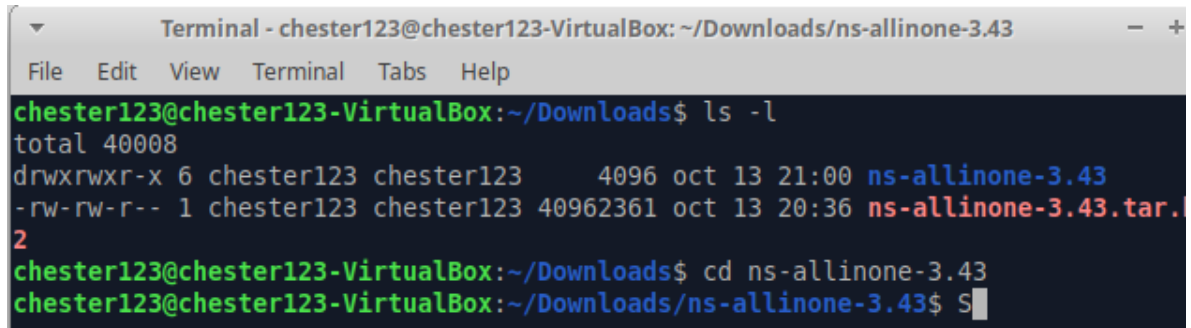
ns-3.43

ns-3.43 was released on October 9, 2024, due to contributions from [twenty-eight authors](#). This release is mainly a maintenance release and the API is generally consistent with the previous ns-3.42 release. See the file `CHANGES.md` for changed model behavior and build system aspects. The GPLv2 license identifiers in the headers of each file have been changed to use SPDX identifiers. The minimum GCC version supported is now version 10.1. Several improvements and bug fixes are listed in the [RELEASE_NOTES](#) and [CHANGES](#) files.

Download

The ns-3.43 release download is available from [this link](#). This download is a source archive that contains some additional tools (`bake`, `netanim`) in addition to the ns-3.43 source. The ns-3 source code by itself can also be checked out of our Git repository by referencing the tag `'ns-3.43'`.

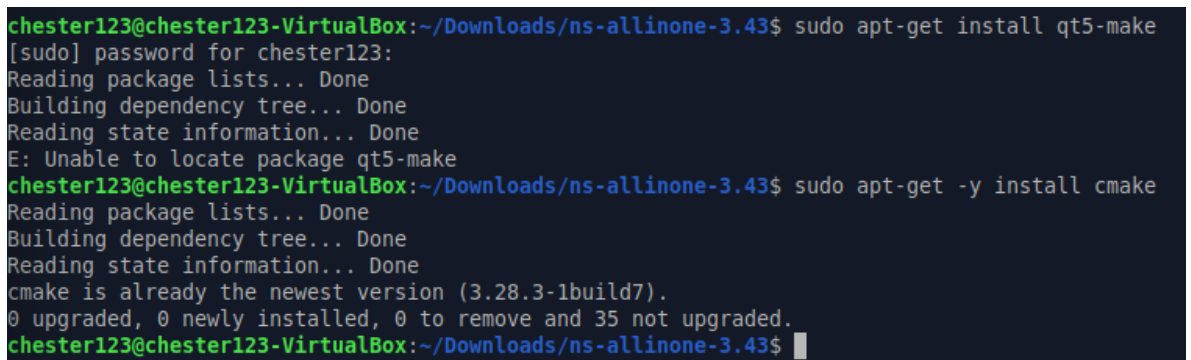
3. Siguiendo, tenemos que descomprimir el archivo descargado desde el terminal (una vez ya estemos ubicados sobre el) empleando el comando “*tar -xvf*” para luego ubicarnos sobre el archivo ya descomprimido:

A terminal window titled "Terminal - chester123@chester123-VirtualBox: ~/Downloads/ns-allinone-3.43". The terminal shows the command `ls -l` being executed, resulting in a directory listing for the current directory. The listing shows two files: `ns-allinone-3.43` (a directory) and `ns-allinone-3.43.tar.xz` (a file). The user then enters `cd ns-allinone-3.43` to navigate into the directory. The prompt changes to `~/Downloads/ns-allinone-3.43$`, and the user starts typing the command `tar -xvf`.

```
chester123@chester123-VirtualBox:~/Downloads$ ls -l
total 40008
drwxrwxr-x 6 chester123 chester123    4096 oct 13 21:00 ns-allinone-3.43
-rw-rw-r-- 1 chester123 chester123 40962361 oct 13 20:36 ns-allinone-3.43.tar.xz
chester123@chester123-VirtualBox:~/Downloads$ cd ns-allinone-3.43
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ S
```

(En esta imagen ya el archivo se encontraba descomprimido, entonces solo nos ubicamos sobre el)

4. Aquí, según la documentación usada es necesario instalar Qt5-qmake y Cmake.
- CMake es herramienta de configuración para compilar y gestionar dependencias en proyectos como Ns-3.
 - Qt5-qmake: es un generador de Makefiles para crear interfaces gráficas si necesitas visualizar o controlar Ns-3.
5. Para ello empleamos los siguiente comandos desde el terminal:

A terminal window showing the installation of qt5-make and cmake. The user runs `sudo apt-get install qt5-make`, which fails with the message "E: Unable to locate package qt5-make". The user then runs `sudo apt-get -y install cmake`, which succeeds, showing that cmake is already installed at version 3.28.3-1build7.

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ sudo apt-get install qt5-make
[sudo] password for chester123:
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
E: Unable to locate package qt5-make
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ sudo apt-get -y install cmake
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
cmake is already the newest version (3.28.3-1build7).
0 upgraded, 0 newly installed, 0 to remove and 35 not upgraded.
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$
```

(En nuestro caso, ya están instalados)

- Lo que sigue, es ejecutar el archivo de nombre *build.py* que se encuentra dentro del archivo descomprimido *ns-allinone-4.3.4*

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ ls
make build.py constants.py netanim-3.109 ns-3.43 __pycache__ README.md util.py
```

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ ./build.py --enable-examples --enable-tests
```

- Ahora, necesitamos ubicarnos en el directorio principal de Ns3.

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43$ cd ns-3.43
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$
```

- Como siguiente paso, copiaremos el programa *first.cc* de la carpeta *tutorials* a la carpeta *scratch* (en esta carpeta se almacenan los programas a ejecutar)

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$ cp examples/tutorial/first.cc scratch/
```

- Ahora, simplemente resta ejecutarlo:

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$ ./ns3 run scratch/first.cc
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$
```

Explicación código, enlace entre nodos.

Se incluyen las librerías necesarias para realizar el enlace simple.

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

Representan el punto de entrada de entrada en el programa. Y una instancia de CommandLine que se encarga de gestionar los argumentos de manera estructurada.

```
int  
main(int argc, char* argv[])  
{  
    CommandLine cmd(__FILE__);  
    cmd.Parse(argc, argv);
```

Usamos el método estático “SetResolution” de la clase “Time” para actualizar la “variable de resolución” a “nanosegundos”. Esto establece que todas las mediciones de tiempo en el programa se realizarán a esta resolución.

```
Time::SetResolution(Time::NS);
```

Estas dos líneas, permiten que, en la ejecución del programa, cuando se llamen a las aplicaciones a “udpEchoClientApplication” y “udpEchoServerApplication” las mismas impriman de manera automática mensajes (que determinan las mismas aplicaciones), y se clasifican en nivel de prioridad “INFO”. Es decir, que muestren lo que están haciendo.

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

Ahora, instanciamos la clase NodeContainer para crear dos nodos.

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

Creamos una instancia de `PointToPointHelper`. Objeto auxiliar para luego, crear la conexión entre los nodos. Y empleamos su método de establecimiento (Setter) Para el “Device”, “Channel”. Pasándoles los datos mostrados.

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

Ahora, creamos una instancia de “`NetDeviceContainer`”, una clase que permite conectar los nodos instalando dispositivos de red configurados con los parámetros de “`pointToPoint`”. Estos dispositivos de red, conocidos como `NetDevices`, son los que facilitan la comunicación entre los nodos de acuerdo con la velocidad y el retardo definidos en el canal.

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

Las Siguietes líneas de código siguen un procedimiento similar, se instancias clases cuyo objetivo es dotar a los nodos con capacidades, como conectarse internet y tener una dirección ir única para cada nodo.

```
InternetStackHelper stack;  
stack.Install(nodes);  
  
Ipv4AddressHelper address;  
address.SetBase("10.1.1.0", "255.255.255.0");  
  
Ipv4InterfaceContainer interfaces = address.Assign(devices);
```

Aquí, asignamos al nodo numero 2 como el receptor en nuestra simulación. Además, que comenzará a funcionar desde el segundo 1, hasta el segundo 10.

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

En estas líneas, configuramos un cliente UDP Echo usando “UdpEchoClientHelper”. Al crear “echoClient(interfaces.GetAddress(1), 9);”, establecemos que el cliente se dirigirá a la dirección IP del nodo servidor (“interfaces.GetAddress(1)”) y usará el puerto 9 para enviar sus paquetes, que es donde el servidor está escuchando. Luego, con los atributos “MaxPackets”, se define la cantidad de paquetes a enviar; aquí se establece en uno, lo que indica que solo enviará un paquete. ‘Interval’ establece el intervalo de tiempo entre envíos. Aunque el valor es de un segundo, solo afecta si se enviaran múltiples paquetes, lo cual no ocurre en este caso. Finalmente, “PacketSize” define el tamaño del paquete en bytes, establecido en 1024 bytes, indicando el tamaño del único paquete que el cliente enviará. En conjunto, estas configuraciones hacen que el cliente envíe un solo paquete de 1024 bytes al servidor en el nodo “n1”, al puerto 9, y, si el servidor responde, el cliente recibirá el eco del mensaje enviado.

```
UdpEchoClientHelper echoClient(interfaces.GetAddress(1), 9);  
echoClient.SetAttribute("MaxPackets", UIntegerValue(1));  
echoClient.SetAttribute("Interval", TimeValue(Seconds(1.0)));  
echoClient.SetAttribute("PacketSize", UIntegerValue(1024));
```

Se establece que el nodo número 1, será el emisor de la simulación. Además, se le asigna un tiempo de inicio en 2 segundos hasta los 10 segundos.

```
ApplicationContainer clientApps = echoClient.Install(nodes.Get(0));
clientApps.Start(Seconds(2.0));
clientApps.Stop(Seconds(10.0));
```

Se ejecuta la simulación. Cuando ya no haya ordenes que completar se ejecuta Destroy(), que la termina.

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

Siendo la salida:

```
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$ ./ns3 run scratch/first.cc
At time +2s client sent 1024 bytes to 10.1.1.2 port 9
At time +2.00369s server received 1024 bytes from 10.1.1.1 port 49153
At time +2.00369s server sent 1024 bytes to 10.1.1.1 port 49153
At time +2.00737s client received 1024 bytes from 10.1.1.2 port 9
chester123@chester123-VirtualBox:~/Downloads/ns-allinone-3.43/ns-3.43$
```

Análisis del paradigma orientado a eventos considerando la simulación.

El programa tiene un paradigma orientado a eventos, lo cual se refleja en su flujo de ejecución. En lugar de un avance continuo, se establece una secuencia de eventos cronológicamente ordenados que comienza en el segundo 1 y termina en el segundo 10, avanzando directamente de evento en evento. A diferencia de un bucle simple, que podría evaluar cada instante de tiempo, este enfoque permite que el programa se mueva

exclusivamente hacia los momentos donde hay acciones predefinidas, evitando los intervalos sin actividad y maximizando la eficiencia

Este tipo de paradigma es particularmente útil en simulaciones como esta, donde los cambios ocurren en momentos discretos y bien delimitados, en lugar de un flujo constante. Cada evento programado sucede a intervalos específicos, como el inicio del servidor y el envío de paquetes por parte del cliente. Este enfoque orientado a eventos asegura que el simulador solo avance en el tiempo hacia los puntos clave, optimizando el procesamiento y reduciendo la carga de recursos, ya que no se detiene en cada nanosegundo, sino únicamente en aquellos momentos en los que algo significativo debe ocurrir.