Práctica 4

a) Instalación del Simulador NS-3 (Versión 3.44)

✅ Requisitos previos

Antes de instalar, comentar que lo haremos usando Ubuntu.

Instalación de dependencias (en Ubuntu)

Ejecutamos lo siguiente en el terminal:

```
sudo apt update
sudo apt install -y gcc g++ python3 python3-dev python3-setuptools git merc
urial \ gir1.2-goocanvas-2.0 python3-gi python3-gi-cairo python3-pygraphviz
python3-pip \ python3-pybind11 cmake libc6-dev libsqlite3-dev libgtk-3-dev v
tun lxc \ openmpi-bin openmpi-common openmpi-doc libopenmpi-dev
```

Descarga y descompresión del código fuente

Descargamos el archivo `ns-allinone-3.44.tar.bz2` desde el campus virtual, lo descomprimimos y compilamos el archivo desde su directorio.

Configuración y compilación

Entramos al directorio del simulador y configuramos:

```
cd ns-3.44
./ns3 configure --enable-examples --enable-tests
./ns3 build
```

```
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$ ./ns3 configure --enable-exampl
es --enable-tests
Warn about uninitialized values.
-- Using default output directory /home/lucian/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44/build
-- Proceeding without cmake-format
-- find_external_library: SQLite3 was found.
CMake Warning at build-support/macros-and-definitions.cmake:853 (find_package):
  By not providing "FindBoost.cmake" in CMAKE_MODULE_PATH this project has
  asked CMake to find a package configuration file provided by "Boost", but
  CMake did not find one.

  Could not find a package configuration file provided by "Boost" with any of
  the following names:

    BoostConfig.cmake
```

```
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$ ./ns3 build
[  0%] Building CXX object src/fd-net-device/CMakeFiles/raw-sock-creator.dir/helper/creator-utils.cc.o
[  0%] Building CXX object src/fd-net-device/CMakeFiles/raw-sock-creator.dir/helper/encode-decode.cc.o
[  0%] Building CXX object src/core/CMakeFiles/core.dir/model/example-as-test.cc.o
[  0%] Building CXX object src/core/CMakeFiles/core.dir/model/time.cc.o
[  0%] Building CXX object src/core/CMakeFiles/core.dir/helper/random-variable-stream-helper.cc.o
[  0%] Building CXX object src/fd-net-device/CMakeFiles/raw-sock-creator.dir/helper/raw-sock-creator.cc.o
[  0%] Building CXX object src/core/CMakeFiles/core.dir/model/unix-fd-reader.cc.o
[  0%] Building CXX object src/core/CMakeFiles/core.dir/model/int64x64-128.cc.o
```

Esto tardó varios minutos, ya que se compila todo el simulador.

Se podrías usar el comando `./test.py` pero ejecuta más de 700 pruebas, lo que podría llevar mucho tiempo así que hemos obviado esta parte.

El comando `./ns3` actúa como un wrapper para compilar y ejecutar los scripts.

Sustituye al uso directo del compilador (por ejemplo `g++`) seguido de la ejecución manual del binario. También podríamos decir que automatiza la selección del entorno, la compilación incremental y la ejecución del código.

b) Ejecución de la primera simulación

Una vez compilado NS3, ejecutamos un modelo de ejemplo incluido con el simulador con `./ns3 run first`.

```
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$ ./ns3 run first
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
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$
```

Esto ejecuta el ejemplo llamado `first`, que está en `examples/tutorial/first.cc`.

Despues de compilar el archivo se ejecuta el binario generado y simula un red mostrando la salida por consola con los logs del simulador.

c) Crear nuestro propio script a partir de `first.cc`

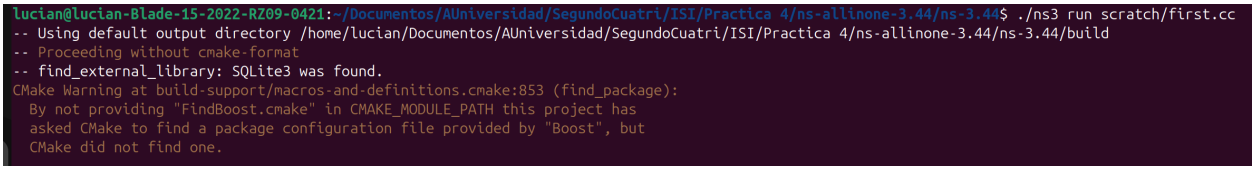
Lo haremos directamente mediante la línea de comandos por comodad:

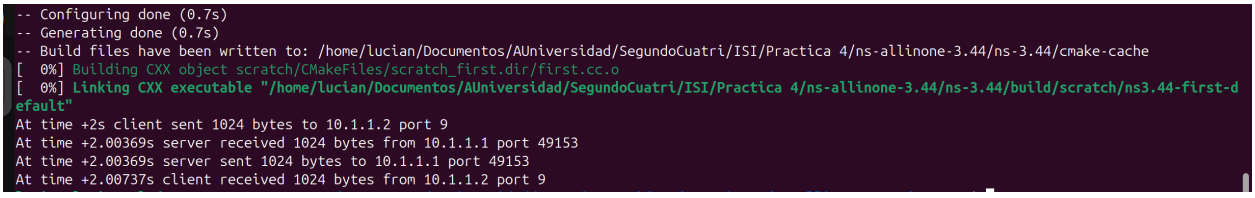
```
cp examples/tutorial/first.cc scratch/
```

Lo colocaremos en el directorio `scratch` como se indica en la práctica.

Una vez realiazada la copia ejecutaremos el siguiente comando y comprobaremos la salida.

```
./ns3 run scratch/first.cc
```





De hecho podremos comprobar que aparece la misma salida que en el apartado b).

El script first.cc como se comentaba antes establece un canal con una tasa de transmisión fija, se asignan direcciones IP a los nodos y se establece una comunicación UDP entre dos de ellos mediante la clase `UdpEchoClientApplication`.

El script completo sin editar es el siguiente:

```
/*
 * SPDX-License-Identifier: GPL-2.0-only
 */

#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"

// Default Network Topology
//
//       10.1.1.0
// n0 -------------- n1
//    point-to-point
//

using namespace ns3;

NS_LOG_COMPONENT_DEFINE("FirstScriptExample");

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

   Time::SetResolution(Time::NS);
   LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
   LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);

   NodeContainer nodes;
   nodes.Create(2);

   PointToPointHelper pointToPoint;
```

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

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

    InternetStackHelper stack;
    stack.Install(nodes);

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

    Ipv4InterfaceContainer interfaces = address.Assign(devices);

    UdpEchoServerHelper echoServer(9);

    ApplicationContainer serverApps = echoServer.Install(nodes.Get(1));
    serverApps.Start(Seconds(1));
    serverApps.Stop(Seconds(10));

    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));
    clientApps.Stop(Seconds(10));

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

d) Validación con Wireshark

NS3 entre otras cosas, puede capturar tráfico en formato `.pcap` para abrirlo con Wireshark.

En el manual se indica que añadiendo la siguiente línea al script antes de `Simulator::Run()` .

```
47
48      Ipv4InterfaceContainer interfaces = address.Assign(devices);
49
50      pointToPoint.EnablePcapAll("first");
51
52      UdpEchoServerHelper echoServer(9);
53
54      ApplicationContainer serverApps = echoServer.Install(nodes.Get(1));
```

Esta línea indicará al simulador que genere archivos de traza formato `.pcap` para las interfaces de red creadas con el helper.

Para la ejecución haremos el siguiente comando:

```
./ns3 run scratch/first
```

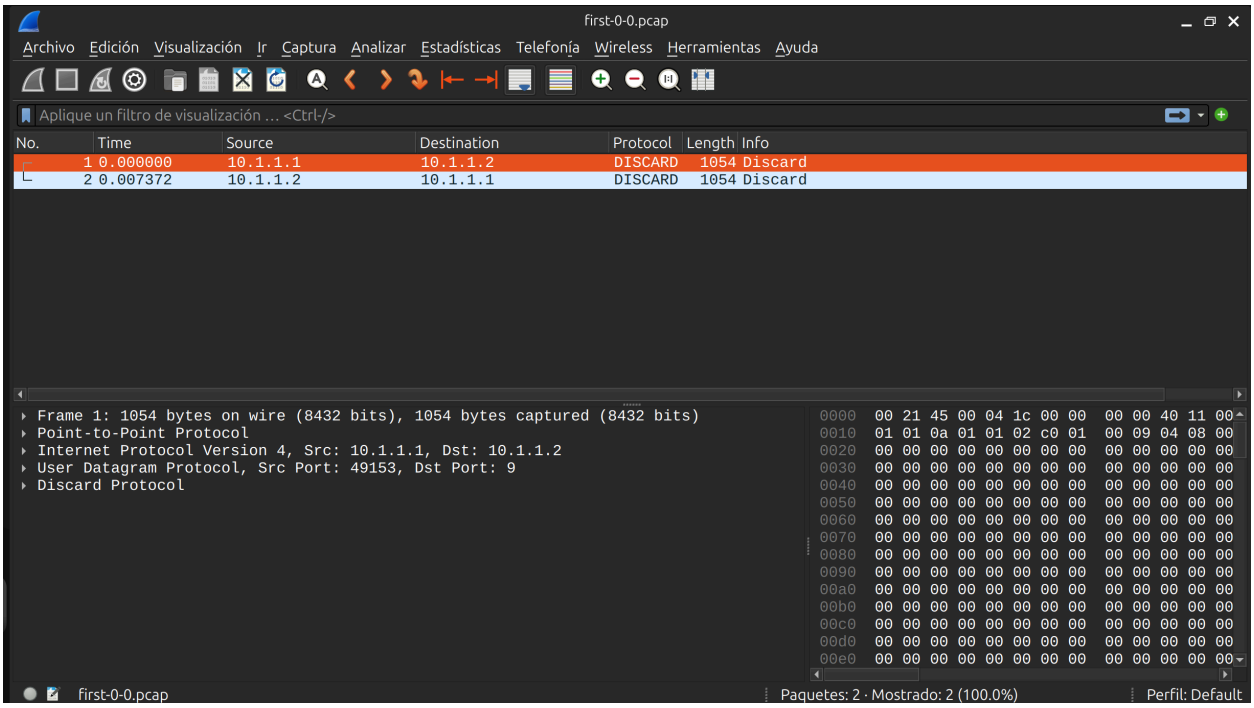
Pero a diferencia de antes se generarán varios archivos `.pcap`.

```
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$ ./ns3 run scratch/first
[  0%] Building CXX object scratch/CMakeFiles/scratch_first.dir/first.cc.o
[  0%] Linking CXX executable "/home/lucian/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44/build/scratch/ns3.44-first-d
efault"
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
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$ ls
AUTHORS   build-support  CMakeLists.txt      CONTRIBUTING.md  first-0-0.pcap  LICENSES     pyproject.toml    scratch     src           utils
bindings  CHANGES.md     CODE_OF_CONDUCT.md  doc              first-1-0.pcap  ns3          README.md         setup.cfg   test.py       utils.py
build     cmake-cache    contrib             examples         LICENSE         __pycache__  RELEASE_NOTES.md  setup.py    testpy-output  VERSION
lucian@lucian-Blade-15-2022-RZ09-0421:~/Documentos/AUniversidad/SegundoCuatri/ISI/Practica 4/ns-allinone-3.44/ns-3.44$
```

Al ejecutar `ls` en nuestro directorio podremos ver que se generaron:

- first-0-0.pcap
- first-1-0.pcap

Cada nodo creado por el helper tiene su propia interfaz, entonces NS-3 crea un archivo pcap para cada interfaz que forma parte de la conexión.



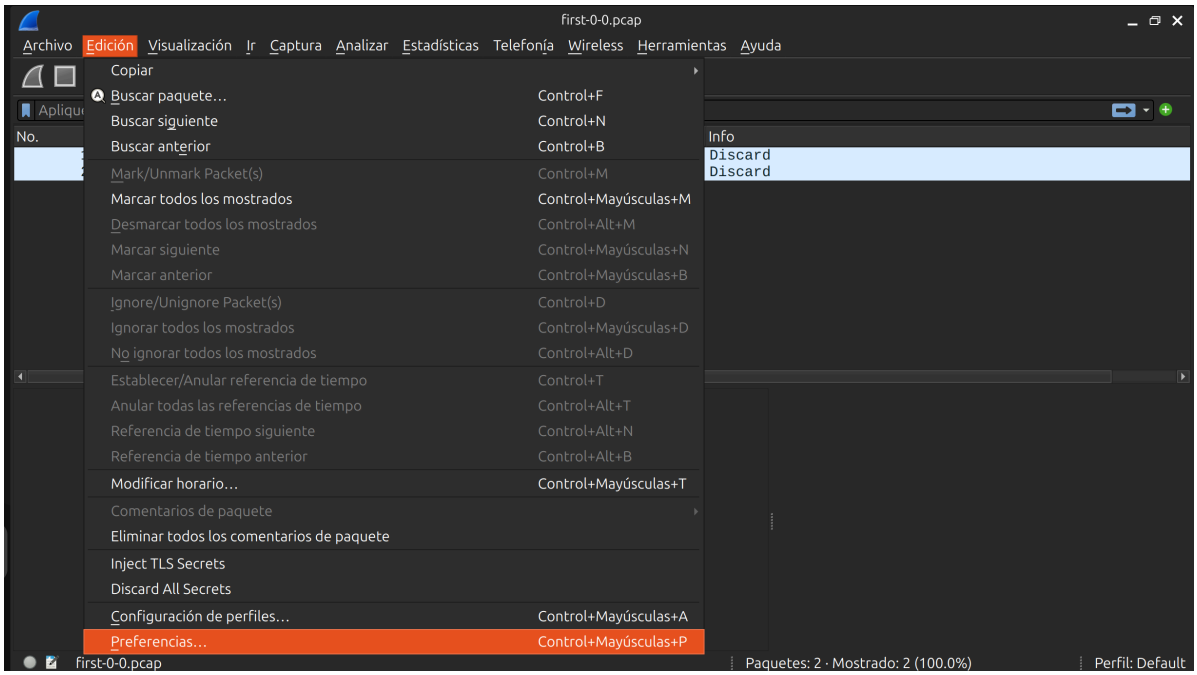
En ambos casos aparecen como DISCARD, ya que por conveción el puerto 9 se asocia al servicio Discard. En el script de NS-3 se está usando el puerto 9 para el *Echo Server*, de modo que Wireshark, al ver paquetes dirigidos a ese puerto, muestra el nombre “Discard Protocol”.

e) Validación de cabeceras IP en Wireshark

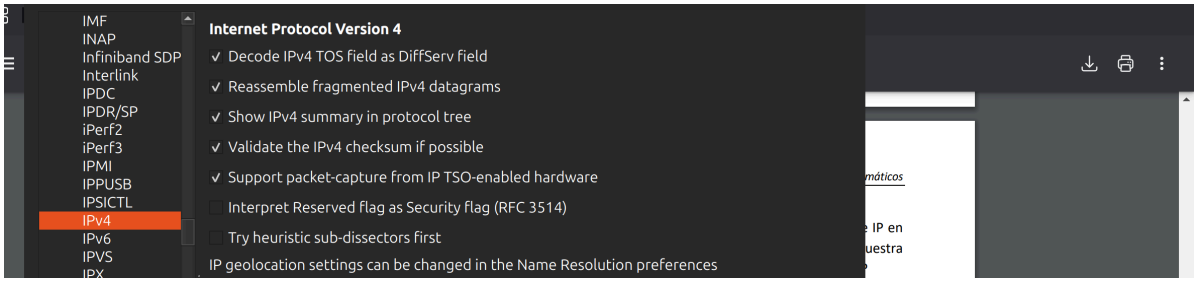
1. **Configuración en Wireshark:**

   Abriremos uno de los archivos .pcap generados con Wireshark (el 0-0).

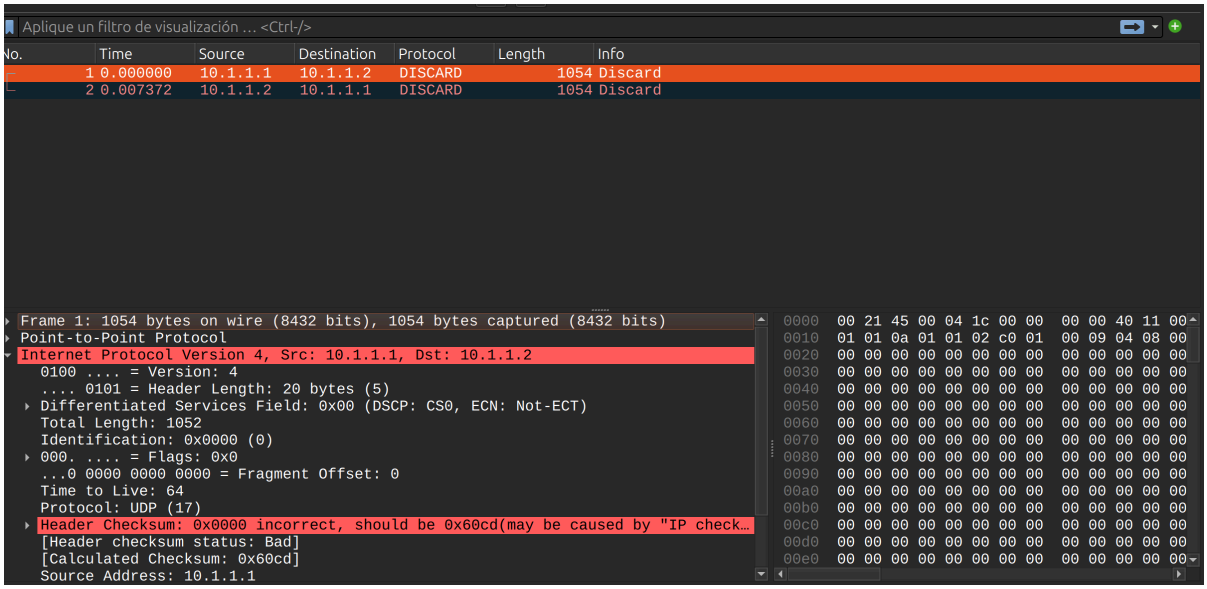
   Vamos a **Edit → Preferences → Protocols → IPv4**



Activamos la opción **"Validate IP checksum if possible"** .



2. Wireshark nos muestra un error en la cabecera IP, indicando que el **checksum es incorrecto.**



Esto se debe a que en NS-3 por razones de optimización, **el cálculo del checksum IP no se realiza de manera real** en la simulación, ya que se trata de una aproximación en un entorno simulado. El fallo de validación en Wireshark es lo esperado y no afecta a la validez de la simulación. Si se quisiera activar el cálculo real de los checksums, habría que configurar NS-3 para que procese los headers de forma completa (usando, por ejemplo, `GlobalValue::Bind()` en algunos casos), lo cual puede conllevar una sobrecarga innecesaria en la simulación.