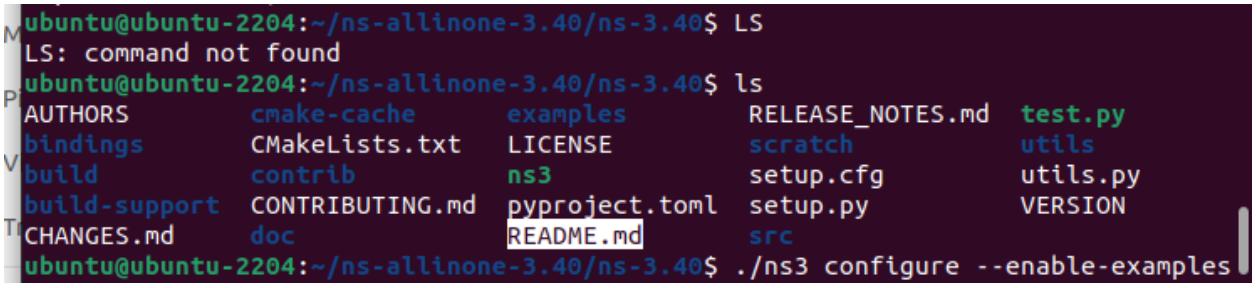
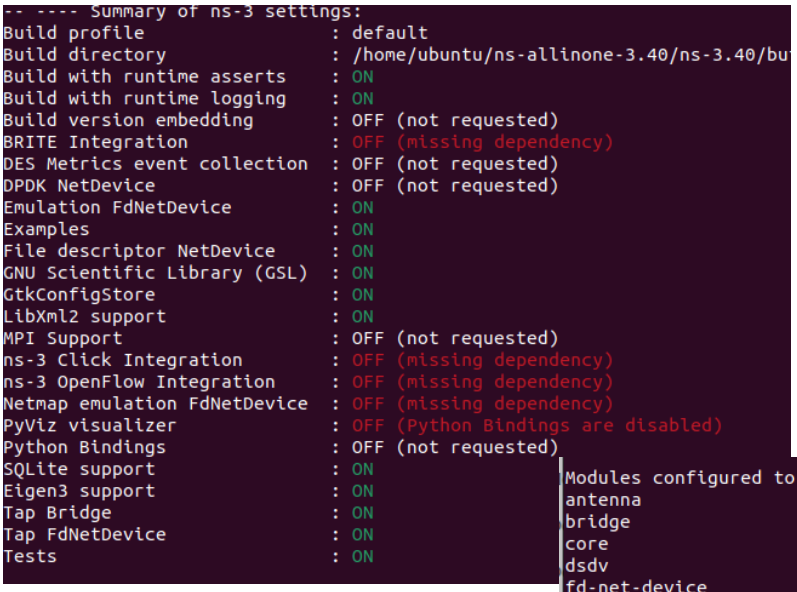
RETO 6

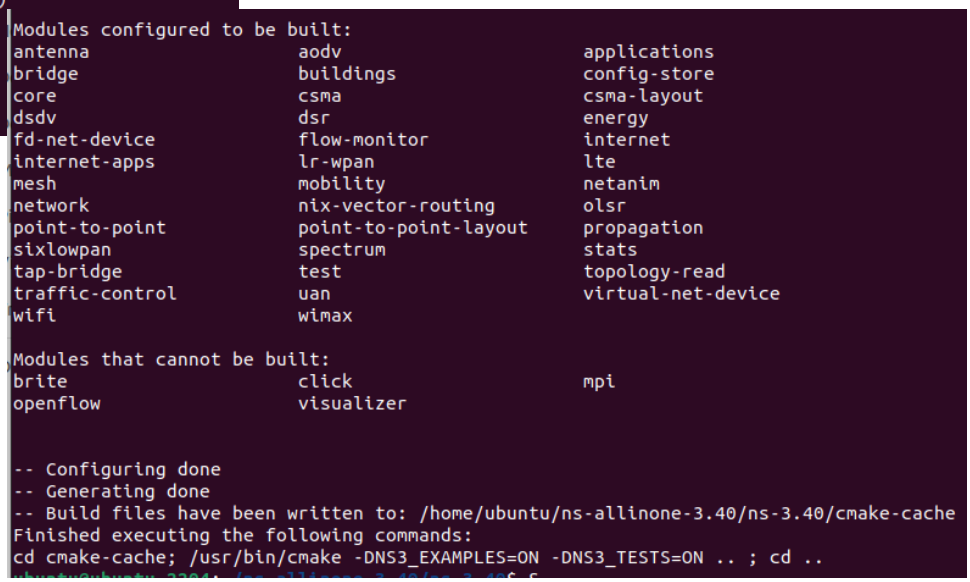
EVALUACION

- Se debe diseñar un ataque a una infraestructura “realista” :etsisi, red empresarial
- Se deberá **crear el fichero CC** de generación y simulación de un gemelo digital en NS3
- Se valorarán todos los “extra”: Mecanismo para detectar el **estado de saturación del router (con colores, etiquetas, un API, etc.); Modelo 3D del entorno** donde ocurre el ataque
- Se debe entregar e**l código CC y el fichero XML** resultado de la simulación, y una breve explicación del ataque diseño y los resultados



> ./ns3 configure –enable-exmaples – enable-tests





Esta todo OK, si no se ha podido completar la configuración de algunos módulos debido a la ausencia de dependencies : “these modules are all optional and require some extra dependencies, but are not needed for initial exploration of ns-3” (*ns-3 Installation Guide)*
$ ./ns3 build
$ ./test.py
Va a realizar muchos tests para comprobar que el build ha salido todo bien y está configurado correctamente. Tarda bastante, son 761 tests.

NS3- es un simulador de eventos de red para redes de internet, mantenido por la Universidad de Washington Consortium y ofrece una solución para simular el comportamiento de redes y protocolos, o para redes a gran escala y tiene una gran capacidad para monitorizar detalles del funcionamiento de la red como cwnd, en TCP es la ventana de bytes que se pueden transmitir en cualquier momento. (Introduction to NS3 Network Simulator 3, Serhat Arslan).

```
[758/761] PASS: Example src/bridge/examples/csma-bridge
[759/761] PASS: Example src/bridge/examples/csma-bridge-one-hop
[760/761] PASS: Example src/aodv/examples/aodv
761 of 761 tests passed (761 passed, 0 skipped, 0 failed, 0 crashed, 0 valgrind errors)
```

$./ns3 run hello-simulator
Para probar que todo funciona bien

```
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ ./ns3 run hello-simulator
[0/2] Re-checking globbed directories...
ninja: no work to do.
Hello Simulator
```

Voy a crear un Firstscript.cc para familiarizarme con los módulos y las clases siguiendo el tutorial 5.Conceptual overview . En el directorio **/ns-allinone-3.40/ns-3.40/** creo el script. En primer lugar, incluimos todas las librerías útiles básicas y declaramos que vamos a usar el scope nombre de ns3: **using namespace ns3; Cuando tengamos el script lo ponemos en el fichero scratch/ y ejecutamos.**

Funciones de LOG son para mostrar mensajes de Logging. También se puede usar el sistema Tracing que da un análisis detallado y la relación entre eventos es automática.

PEQUEÑOS APUNTES DE NS3 - DOCUMENTACIÓN OFICIAL

`Modules` **you will see a list of *ns-3* module documentation**. The concept of module here ties directly into the **module include files discussed above**. The ***ns-3* logging subsystem is in the `Using the Logging Module` section**.
Above statement by looking at the `Core` module, > `Debugging tools` book > `Logging` page.

[Logging page] **Optional to Logging output** messages is Tracing to get data out of your model. tutorial section **Using the Tracing System** for more details on our tracing system)
- *ns-3* takes the view of **multi-level approach to** message logging.

Logging can be disabled completely, enabled on a component-by-component basis, or enabled globally; The ***ns-3*** **log module provides** a straightforward, relatively easy to use way to get **useful information out of your simulation.**

- **Error messages are logged to the "operator console"** (which is typically `stderr` in Unix- based systems). In other systems, **warning messages** may be output as well as more detailed informational messages. In some cases, logging facilities are used to output **debug messages** which can quickly turn the output into a blur.

LOG_LEVEL_TYPE

1. LOG_ERROR — Log error messages (associated macro: NS_LOG_ERROR);
2. LOG_WARN — Log warning messages (associated macro: NS_LOG_WARN);
3. LOG_DEBUG - Log relatively rare, ad-hoc debugging messages (associated macro: NS_LOG_DEBUG);
4. LOG_INFO — Log informational messages about program progress
5. 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);
6. LOG_LOGIC – Log messages describing logical flow within a function (associated macro: NS_LOG_LOGIC);
7. LOG_ALL — Log everything mentioned above (no associated macro).

LOG_LEVEL_TYPE that, if used, enables logging of **all the levels above it in addition to it's level.** (As a consequence of this, LOG_ERROR and LOG_LEVEL_ERROR and also LOG_ALL and LOG_LEVEL_ALL are functionally equivalent.) For example, enabling LOG_INFO will only enable messages provided by NS_LOG_INFO macro, while enabling LOG_LEVEL_INFO will also enable messages provided by NS_LOG_DEBUG, NS_LOG_WARN and NS_LOG_ERROR macros.

[tutorial 5.] Inside the main
Time::SetResolution(Time::NS);
You can change the resolution exactly once. The mechanism enabling this flexibility is somewhat memory hungry, so once the resolution has been set explicitly we release the memory, preventing further updates. (**If you don't set the resolution explicitly, it will default to one nanosecond, and the memory will be released when the simulation starts.)**

Time resolution es la unidad minima de tiempo que un sistema simulador puede medir o representar → la minima fraccion detimpo distinguible entre dos eventos.

The next two lines of the script are used to **enable two logging components** that are **built into the Echo Client and Echo Server applications:**
LogComponentEnable("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable("UdpEchoServerApplication", LOG_LEVEL_INFO);

TOPOLY HELPERS

- Ahora vamos a crear los nodos (hosts), netdevices, y channels, basicamente la topologia de red, utilizando las clases Node, NodeDevice, Channels, ademas de utilizar tambien las clase de las distintas topologias, ya que NS3 facilita de esta forma el poder crear la topologia de red sin escribir mucho. -

Creamos los nodos de la red y luego la topologia.
NodeContainer nodes;

PointToPointHelper pointToPoint;

A continuacion configuramos la Vtransmision y el Tiempo de propagacion del enlace.
“DataRate”, “5Mbps” ; “Delay”, “2ms”;

APPLICATION LAYER
In this script we use **two specializations of the core *ns-3* class** `Application` called `UdpEchoServerApplication` and `UdpEchoClientApplication`. Just as we have in our previous explanations, we use **helper objects** to help configure and manage the underlying objects.

helper can’t do anything useful unless it is provided with a port number that the client also knows about. we require the port number as a parameter to the constructor. The constructor, in turn, simply does a `SetAttribute` with the passed value. If you want, you can set the “Port” `Attribute` to another value later using `SetAttribute`.
the `UdpEchoServerHelper` object has an `Install` method. It is the execution of this method that actually causes the underlying echo server application to be instantiated and attached to a node

There is a typical case where `Simulator::Stop` is absolutely necessary to stop the simulation: when there is a self-sustaining event. Self-sustaining (or recurring) events are events that always reschedule themselves. As a consequence, they always keep the event queue non-empty.

There are many protocols and modules containing recurring events, e.g.:

- FlowMonitor - periodic check for lost packets
- RIPng - periodic broadcast of routing tables update
- etc.

El script que hemos creado siguiendo el tutorial 5 de la documentacion: consiste en crear una red muy sencilla básica utilizando los helpers de ns3 para construir. Utilizamos NodeContainerHelper, NetDeviceContainerHelper, PointToPointHelper, Ipv4AdressHelper, InternetStackHelper and Ip4InterfaceHelper. Cada una de estas clases crea objetos que hacen posible montar la red que se va a emular sin escribir mucho código. Se crean los nodos de la red, la NIC y el enlace que une a las dos estaciones en una comunicación de punto a punto. Internet es el protocolo que se utiliza para obtener toda la pila de protocolos: UDP/TCP, IP. Después se crea un objeto aplicación servidor y cliente para que el nodo 1 actúe como servidor y el nodo 0 como cliente. Se definen los parámetros de la simulación para que dure 20s y se envíe y reciba un solo paquete por el puerto 9.

Asi las librerias que hemos incluido se compilaran con
ns-3.40$/.ns3 myfirst.cc

```
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ ./ns3 run myfirst.cc
[0/2] Re-checking globbed directories...
[2/2] Linking CXX executable ../build/examples/tutorial/ns3.40-myfirst-default
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
```

BUS NETWORK
En el segundo ejemplo tenemos el script second.cc ene l que implementamos una red como la del primer script pero en esta añadimos una LAN con 3 equipos conectados a un bus de comunicacion que utiliza el protocolo CSMA (Carrier Sense Multiple Access), es decir, control de acceso al medio. De normal se suele utilizar el CSMA/CD que es deteccion de portadora por colision. Aqui simplificamos. Vamos a incluir las librerias necesarias “helpers”, declaramos el namespace, y un script log. En el main vamos a declarar dos variables: verbose, booleano que permitira encender o apagar el registro de los eventos en un log, y nCsma = 3, numero de nodos en el canal.

CommandLine cmd;

md.AddValue("nCsma", "Number of \"extra\" CSMA nodes/devices", nCsma);
cmd.AddValue("verbose", "Tell echo applications to log if true", verbose);
cmd.Parse(argc, argv);
--
La aplicación cliente se instala en el Nodo0 y el servidor en el ultimo de los nodos de csma

Decisión del ejemplo	¿Qué se pretende mostrar?	¿Por qué ayuda?
Servidor en `csmaNodes.Get(nCsma)` (último host de la LAN)	1. Que el paquete recorra **dos redes distintas** (PPP + CSMA). 2. Que haya **un salto IP** y se vea el reenvío en n1.	‣ Se observa ARP en la LAN, encapsulado PPP en el enlace, tablas de rutas, etc. ‣ Da un ejemplo minimalista de “cliente en la WAN → servidor en la LAN”.
Cliente en n0 (punto-a-punto)	1. Simular al “usuario externo” que llega a la red local. 2. Mantener un solo cliente para simplificar la demo.	‣ El rol del router (n1) queda claro: un interfaz hacia Internet (PPP) y otro hacia la LAN (CSMA).

* El servidor se pone en el “último” nodo CSMA **para obligar al tráfico a pasar por el router n1 y así mostrar, en un ejemplo muy compacto, la interacción entre dos redes diferentes (PPP y CSMA), ARP, routing y captura de paquetes en cada extremo**. Si tu objetivo es otro (por ejemplo medir rendimiento solo en el enlace PPP)
--
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
what happens is that each node behaves as if it were an OSPF router that communicates instantly and magically with all other routers behind the scenes. Each node generates link advertisements and communicates them directly to a global route manager which uses this global information to construct the routing tables for each node.
--
le dice al simulador que grabe todo el tráfico que pase por las **NetDevices** creadas con ese helper en archivos **.pcap**. → captura los paquetes que pasan por el nodoPtP(1), osea los paquetes que salen y entran en la red bus.

2. ATAQUE SNIFFING Y ESCUHA PASIVA EN WIFI

Simulación del ataque de sniffing (escucha pasiva en Wi-Fi):

- **Objetivo**: Un atacante en una red Wi-Fi intercepta los paquetes que viajan por el aire y los analiza.

- **En la simulación**: El "atacante" actuará como un nodo Wi-Fi que está escuchando pasivamente el tráfico, mientras que el cliente y el servidor están comunicándose en la misma red.

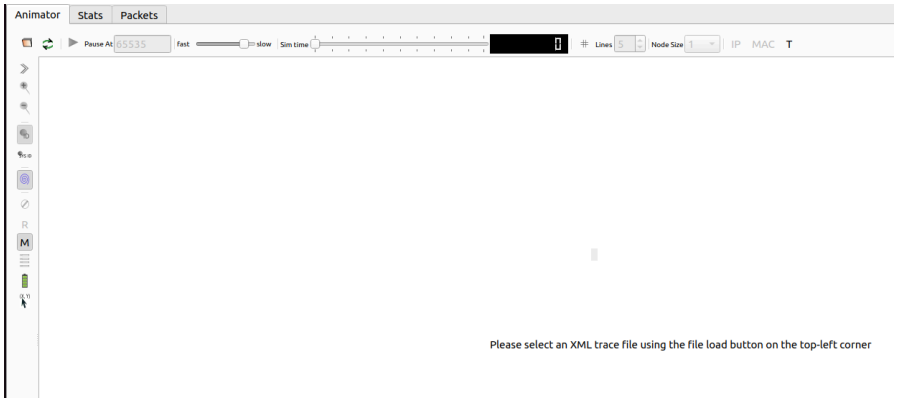
```
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/myAttack.cc
[0/2] Re-checking globbed directories...
ninja: no work to do.
NS_ASSERT failed, cond="m_ptr", msg="Attempted to dereference zero pointer", +0.064856000s 0
file=/home/ubuntu/ns-allinone-3.40/ns-3.40/src/core/model/ptr.h, line=748
NS_FATAL, terminating
terminate called without an active exception
Command 'build/scratch/ns3.40-myAttack-default' died with <Signals.SIGABRT: 6>.
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ S
```

Me daba error, el cual me costó bastante encontrar y arreglar hasta que se me ocurrió por divinidad diría que en el error aparece como “un intento de referenciar un puntero zero”, que entiendo que querra decir vacío. Pero al final, se me ocurrió que no había configurado el movimiento del sniffer. Entendiendo que era una funcionalidad adicional y no esencial, pero debe ser que al ser parte de una red wifi, se debe configurar este aspecto. En la documentación, al menos en el apartado de topología wifi, no he encontrado nada.

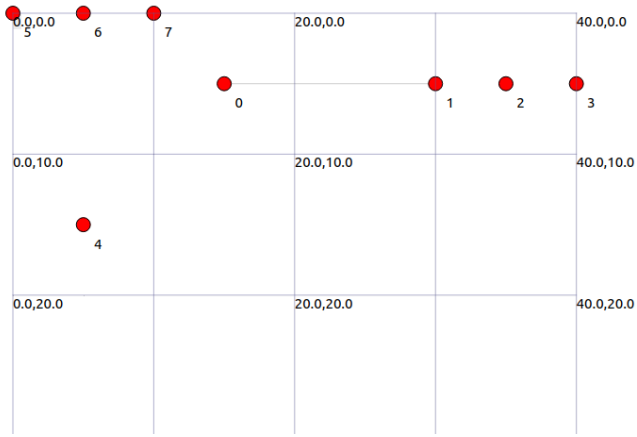
Una vez funciona todo y se compila correctamente, lo he ejecutado y se puede ver el intercambio del paquete de la simulación

```
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/myAttack.cc
[0/2] Re-checking globbed directories...
ninja: no work to do.
At time +2s client sent 1024 bytes to 10.1.2.3 port 9
At time +2.01227s server received 1024 bytes from 10.1.3.3 port 49153
At time +2.01227s server sent 1024 bytes to 10.1.3.3 port 49153
At time +2.02556s client received 1024 bytes from 10.1.2.3 port 9
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$
```

Se genera un .xml para la animación: “Siffer.xml” y se puede comenzar la simulación con el siguiente comando: *$ ./netanim-3.109/NetAnim* Se abrirá una ventana:



Abrimos el archivo xml en el directorio donde hemos ejecutado la simulación y vemos como transcurre la animación.



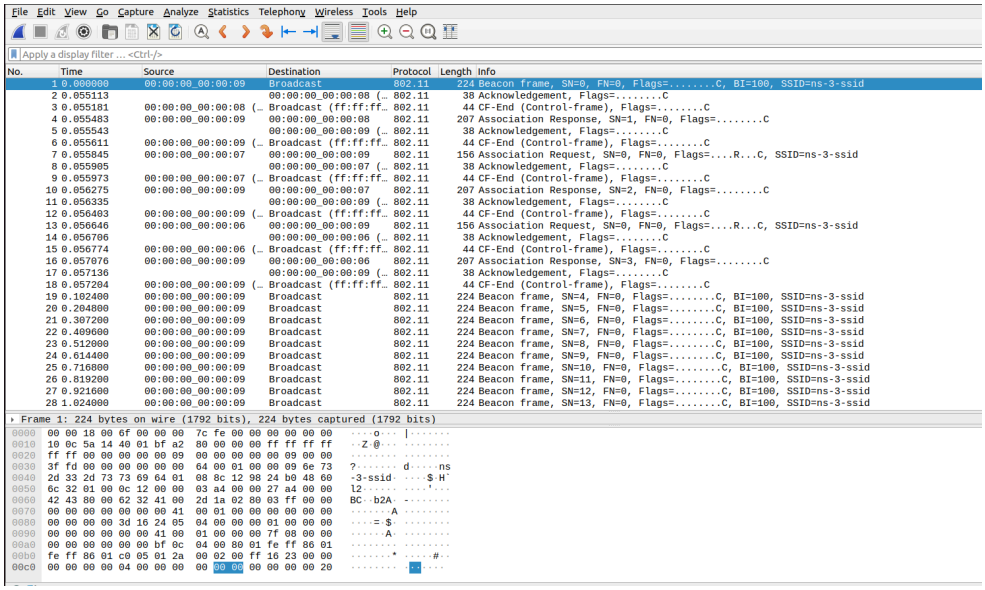
Como observamos en la animación, vemos la disposición de la red que he diseñado en el propio codigo [myAttack.cc]

La animación mostrara el intercambio del paquete entre el nodo 5 y el nodo 3 de la otra red.

El nodo 4 es el sniffer, el cual no tiene asignado ninguna dirección ip, ni siquiera está en la misma red wifi. Sin embargo puede interceptar el tráfico porque está en modo promiscuo y puede escuchar si está suficientemente próximo a la red wifi objetivo.

```
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$ ./ns3 run scratch/myAttack.cc
[0/2] Re-checking globbed directories...
ninja: no work to do.
At time +2s client sent 1024 bytes to 10.1.2.3 port 9
At time +2.01227s server received 1024 bytes from 10.1.3.3 port 49153
At time +2.01227s server sent 1024 bytes to 10.1.3.3 port 49153
At time +2.02556s client received 1024 bytes from 10.1.2.3 port 9
ubuntu@ubuntu-2204:~/ns-allinone-3.40/ns-3.40$
```

Los archivos .pcap generados al final de la simulación se pueden examinar con wireshark como la siguiente captura del sniffer:



Como podemos observar al capturar los paquetes de la red y vemos que podemos identificar el nombre de la wifi a el que las estaciones están contenidas SSID = ns3-ssid, que el ssid que hemos definido en el .cc

También ha interceptado todos los Beacon Frames, tramas que se intercambian en la red wifi y las direcciones mac de cada dispositivo.