Εξαμηνιαία εργασία

- Η εργασία θα πραγματοποιηθεί σε ομάδες των **2 ή 3 ατόμων**
- Καθορίζει στο **50% την τελική βαθμολογία** στο μάθημα
- Βασίζεται στη χρήση του **NS-3** network simulator
- Απαιτείται η χρήση προσωπικού υπολογιστή (κατά προτίμηση **Linux Ubuntu-based PC**) για την υλοποίηση
  - Windows will need Virtual Box/VMware

The NS-3 simulator

- NS-3 is a discrete event simulator, entirely written in **C++**
- Latest versions to be used for the project purposes **3.28-3.30 - more info: https://www.nsnam.org/wiki/Installation**
- Python bindings are also available for some modules but **C++ is strongly recommended**

Some things about ns-3

- ns-3 is designed as **a set of libraries** that can be combined together and also with other external software libraries.
- While some simulation platforms provide users with a single, integrated **graphical user interface environment** in which all tasks are carried out, ns-3 is more modular in this regard. Several external animators and data analysis and visualisation tools can be used with ns-3. However, **users should expect to work at the command line and with C++ (or Python) software development tools.**
- **ns-3 is primarily used on Linux-based systems**, although support exists for Cygwin (for Windows), and native Windows Visual Studio support is in the process of being developed.
- ns-3 is not an officially supported software product of any company. Support for ns-3 is done on a best-effort basis on the **ns-3-users mailing list (ns-3 google group).**

NS-3 modules

- NS-3 software is organised into a set of libraries/modules
  - Antenna
  - Internet
  - Wifi
  - Wimax
  - lte
  - Buildings
  - Openflow
  - Propagation
  - ++

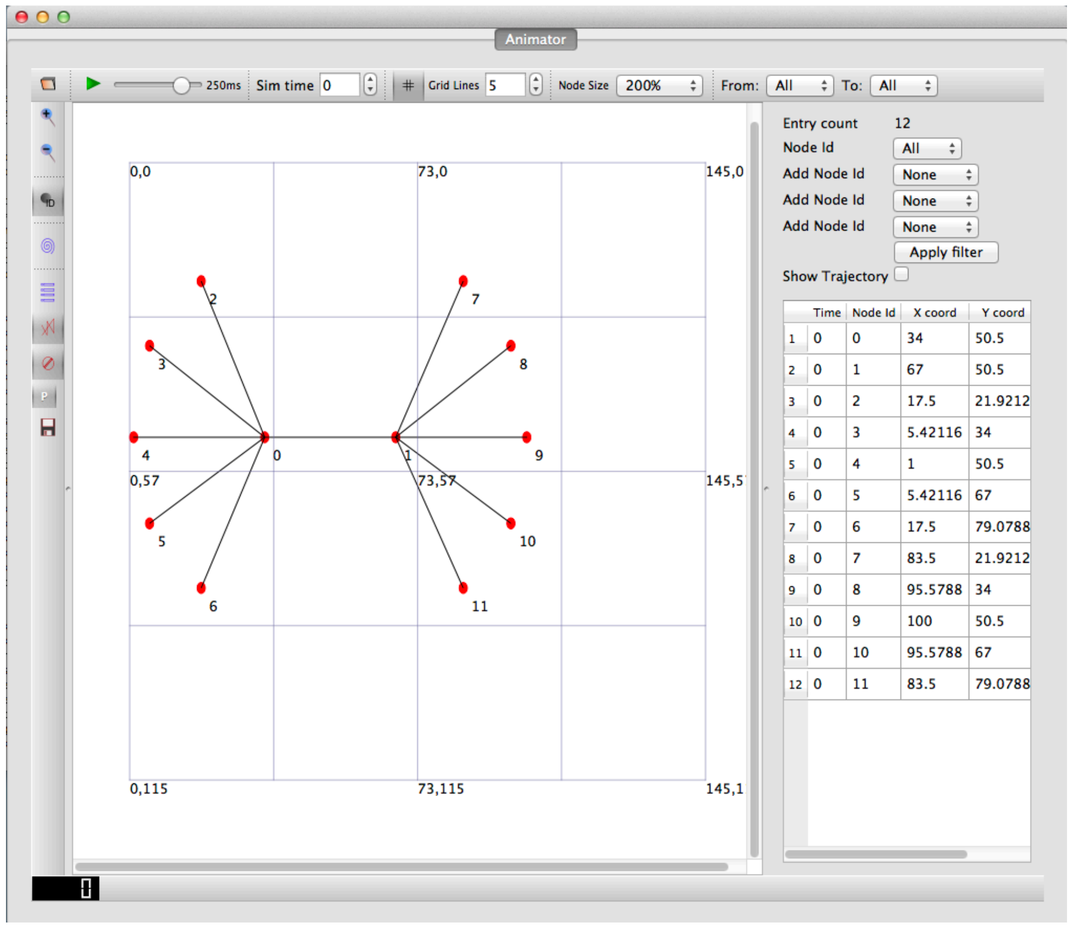
NS-3 visualisation tools

- Several external animators and data analysis and visualisation tools can be used with ns-3. However, **users should expect to work at the command line and with C++ and/or Python software development tools.**

- Recommended visualisation tools:

  - **Netanim** (NetAnim is an **offline** animator based on the Qt toolkit. It currently animates the simulation using an XML trace file collected during simulation)

  - **PyViz** (NS-3 PyViz is a **live** simulation visualizer, meaning that it uses no trace files. It can be most useful for debugging purposes, i.e. to figure out if mobility models are what you expect, where packets are being dropped, etc.)



Sim time 0 | Grid Lines 5 | Node Size 200% | From: All | To: All

Entry count 12
Node Id All
Add Node Id None
Add Node Id None
Add Node Id None
Apply filter
Show Trajectory

	Time	Node Id	X coord	Y coord
1	0	0	34	50.5
2	0	1	67	50.5
3	0	2	17.5	21.9212
4	0	3	5.42116	34
5	0	4	1	50.5
6	0	5	5.42116	67
7	0	6	17.5	79.0788
8	0	7	83.5	21.9212
9	0	8	95.5788	34
10	0	9	100	50.5
11	0	10	95.5788	67
12	0	11	83.5	79.0788

NS-3 conceptual overview
Key abstractions

- **Node**: 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.

- **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.

- **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.

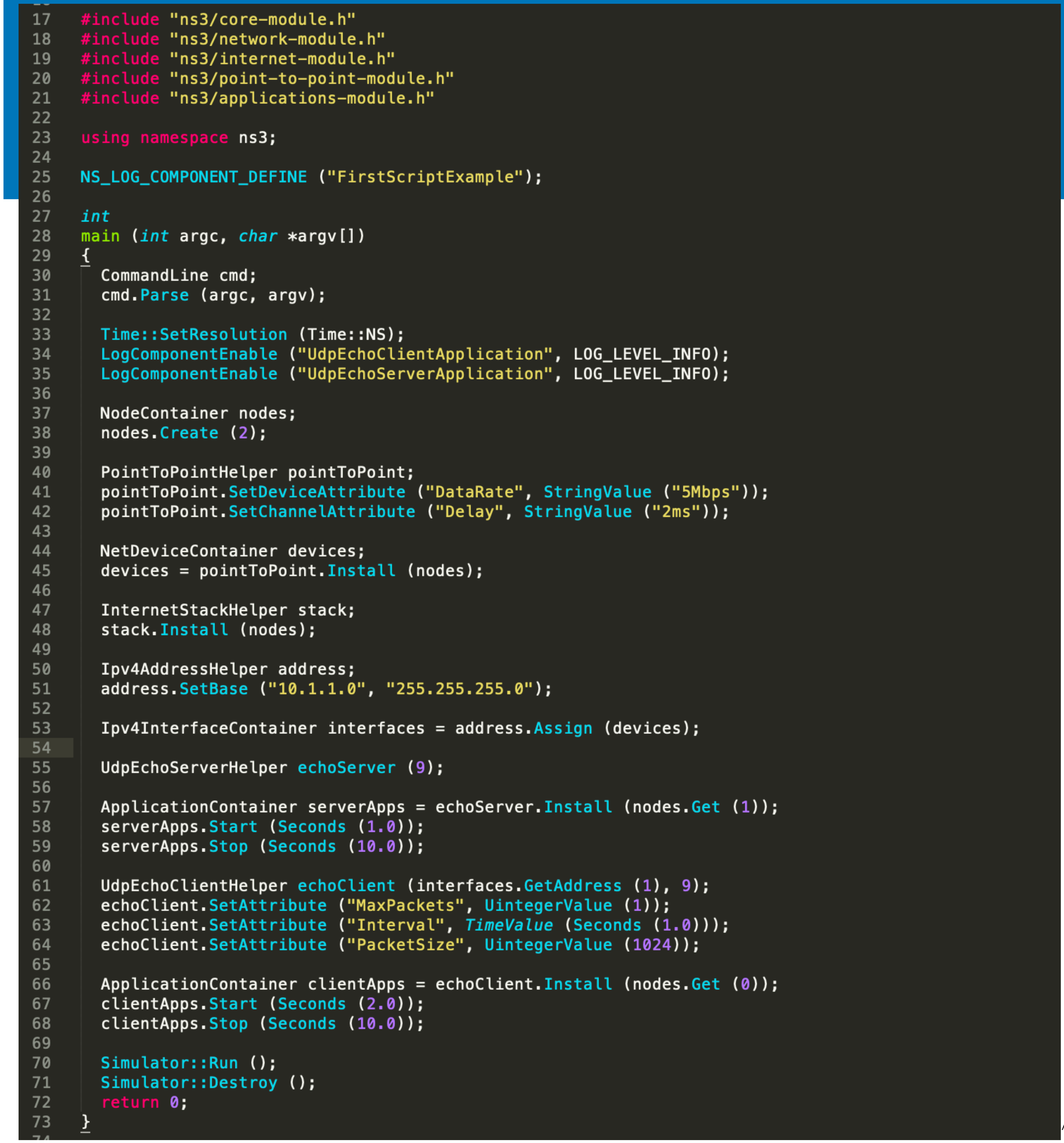
  - A Channel specialization may model something as simple as a wire.

  - 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.

- **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 Channels via multiple NetDevices.

first.cc



```
#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"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("FirstScriptExample");

int
main (int argc, char *argv[])
{
  CommandLine cmd;
  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 (1.0));
  serverApps.Stop (Seconds (10.0));

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

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

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

first.cc

```
#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"
```

- To help the high-level script users deal with the large number of include files present in the system, **we group includes according to relatively large modules**. A single include file is included that will recursively load all of the include files used in each module.

```
using namespace ns3;
```

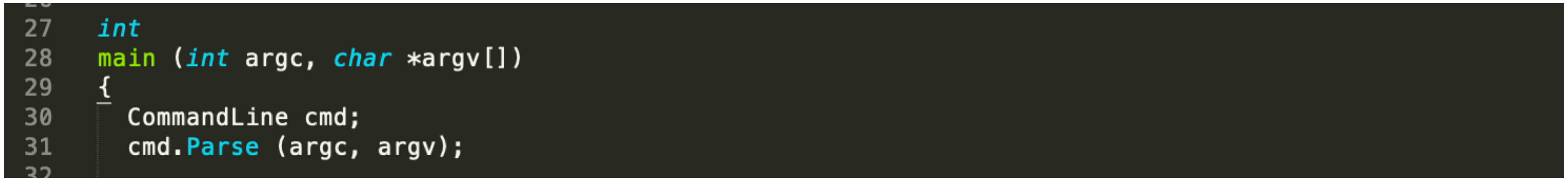
- The ns-3 project is implemented in a C++ namespace called ns-3. This groups all ns-3-related declarations in a scope outside the global namespace, which we hope will help with integration with other code. The C++ **using** statement introduces the ns-3 namespace into the current (global) declarative region.

- This is a fancy way of saying that after this declaration, **you will not have to type ns3:: scope** resolution operator before all of the ns-3 code in order to use it.

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

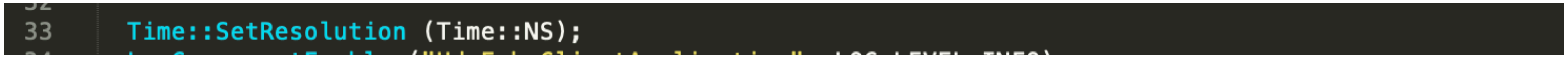
- This line declares a logging component called ***FirstScriptExample*** that allows you to enable and disable console message logging by reference to the name.

first.cc



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

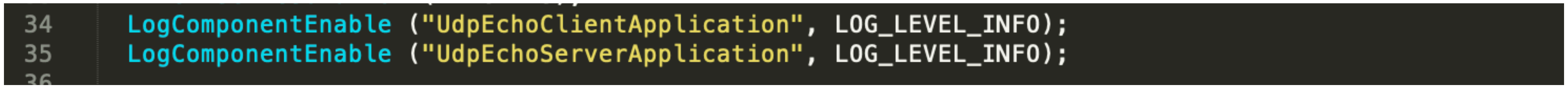
- This is just the declaration of the main function of your program (script). Just as in any C++ program, you need to define a main function that will be the first function run. There is nothing at all special here. Your ns-3 script is just a C++ program



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

- The resolution is the smallest time value that can be represented (as well as the smallest representable difference between two time values).

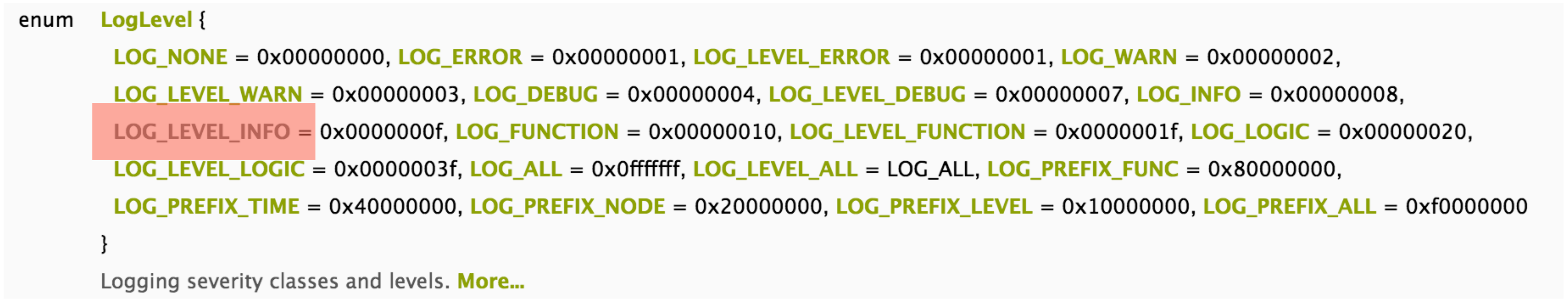
first.cc



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

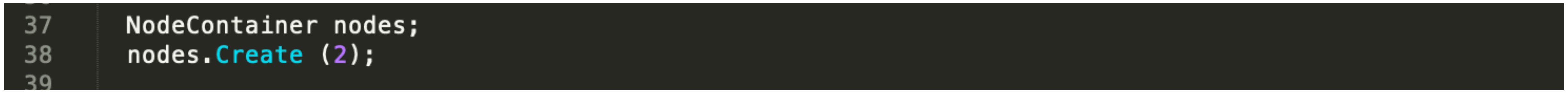
- There are a number of levels of logging verbosity/detail that you can enable on each component.

- These two lines of code enable debug logging at the ***INFO*** level for echo clients and servers. This will result in the application printing out messages as packets are sent and received during the simulation.



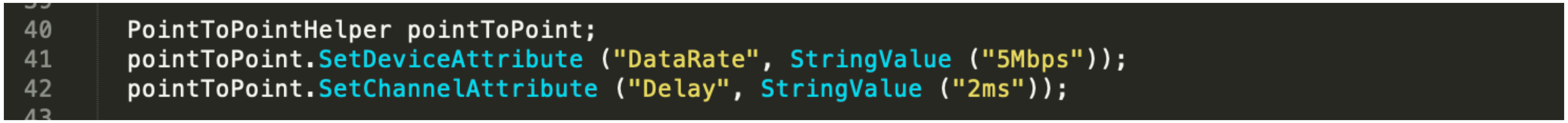
```
enum   LogLevel {
         LOG_NONE = 0x00000000, LOG_ERROR = 0x00000001, LOG_LEVEL_ERROR = 0x00000001, LOG_WARN = 0x00000002,
         LOG_LEVEL_WARN = 0x00000003, LOG_DEBUG = 0x00000004, LOG_LEVEL_DEBUG = 0x00000007, LOG_INFO = 0x00000008,
         LOG_LEVEL_INFO = 0x0000000f, LOG_FUNCTION = 0x00000010, LOG_LEVEL_FUNCTION = 0x0000001f, LOG_LOGIC = 0x00000020,
         LOG_LEVEL_LOGIC = 0x0000003f, LOG_ALL = 0x0fffffff, LOG_LEVEL_ALL = LOG_ALL, LOG_PREFIX_FUNC = 0x80000000,
         LOG_PREFIX_TIME = 0x40000000, LOG_PREFIX_NODE = 0x20000000, LOG_PREFIX_LEVEL = 0x10000000, LOG_PREFIX_ALL = 0xf0000000
       }
       Logging severity classes and levels. More...
```

first.cc



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

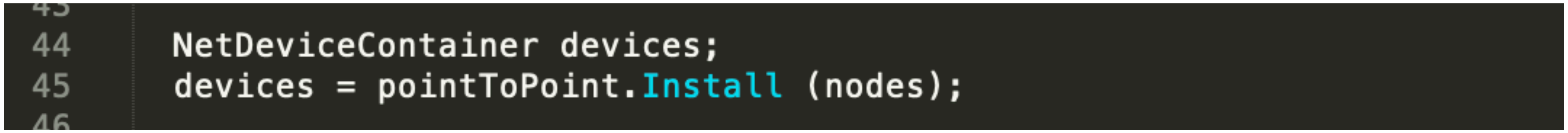
- Creation of two nodes using the ***NodeContainer*** class
- The first line above just declares a NodeContainer which we call nodes. The second line calls the create method on the nodes object and asks the container to create two nodes.



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

- The first line instantiates a ***pointToPoint*** helper
    - ***helpers are a higher level API overlaid over the ns-3 core***
- The 2nd and 3rd lines tells the ***pointToPoint*** object to use the value “5Mbps” (five megabits per second) as the “DataRate” when it creates a ***pointToPointNetDevice***
- From a more detailed perspective, the string “DataRate” corresponds to what we call an ***attribute*** of the ***pointToPointNetDevice***

first.cc



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

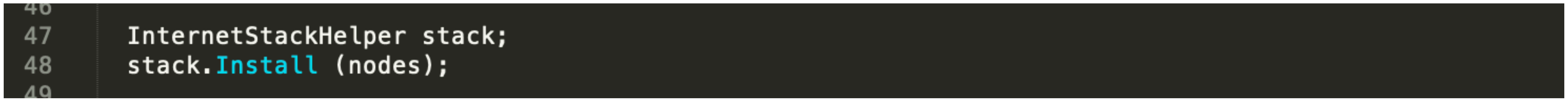
- At this point in the script, we have a NodeContainer hat contains two nodes. We have a PointToPointHelper that is initiated and ready to make ***pointToPointNetDevices*** and **wire** *pointToPointChannels* objects between them.

- Just as we used the ***NodeContainer*** topology helper object to create the ***Nodes*** for our simulation, we will ask the ***pointToPointHelper*** to do the work involved in **creating, configuring and installing our devices for us**.

- We will need to have a list of all of the NetDevice objects that are created (***devices***), so we use a NetDeviceContainer to hold them just as we used a NodeContainer to hold the nodes we created.

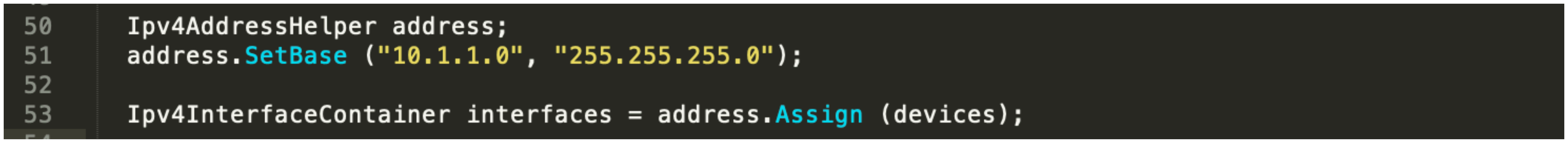
first.cc

- We now have nodes and devices configured, but we don’t have any protocol stacks installed on our nodes.



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

- The Install method takes a NodeContainer as a parameter. When it is executed, it will install an Internet Stack (TCP, UDP, IP, etc.) on each of the nodes in the node container.



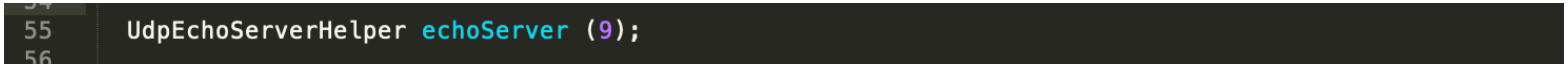
```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");

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

- Next we need to associate the devices on our nodes with IP addresses
- An address helper is declared, which should start allocating IP addresses from the network 10.1.1.0 using the mask 255.255.255.0 to define the allocatable bits
- The 3rd line performs the actual address assignment in automated way.

first.cc

- Now we have a point-to-point network built, with stacks installed and IP addresses assigned. **What we need at this point are applications to generate traffic**

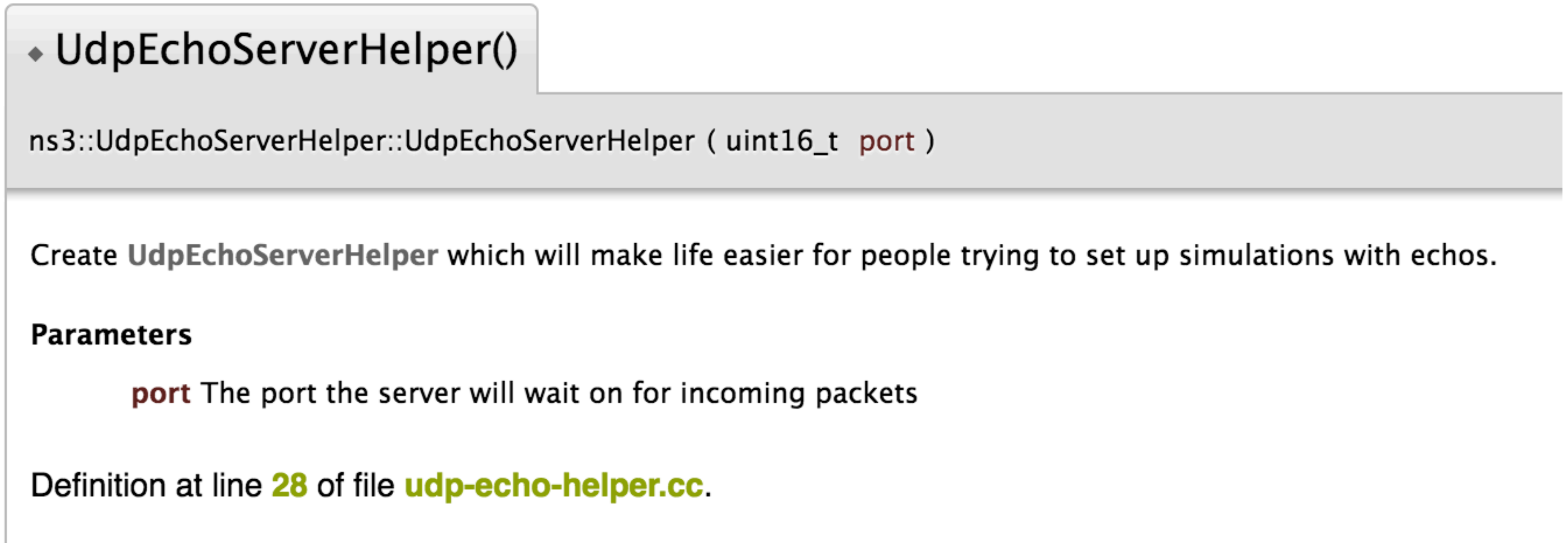
- In this script we use two specialisations of the core ns-3 class **Applications** called ***UdpEchoServerApplication*** and ***UdpEchoClientApplication***, via the respective helpers



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

- The first line of code in the above snippet declares the ***UdpEchoServerHelper.*** As usual, this isn't the application itself, <u>it is an object used to help us create the actual applications</u>. One of our conventions is to place required **Attributes** in the helper constructor.

- In this case, **the helper can't do anything useful unless it is provided with a port number that the client also knows about.**



◆ UdpEchoServerHelper()

ns3::UdpEchoServerHelper::UdpEchoServerHelper (uint16_t port)

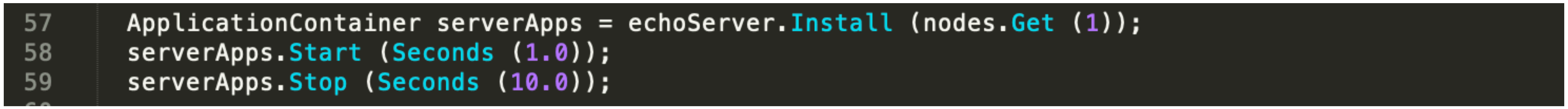
Create UdpEchoServerHelper which will make life easier for people trying to set up simulations with echos.

Parameters

port The port the server will wait on for incoming packets

Definition at line 28 of file udp-echo-helper.cc.

first.cc



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

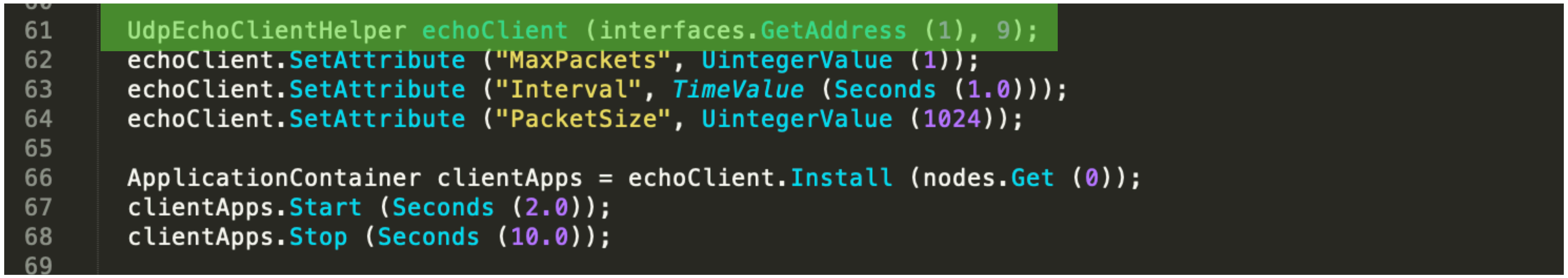
- Similar to many other helper objects, the ***UdpEchoServerHelper*** object has an ***Install*** method. It is the execution of this method that actually causes the underlying echo server application to be instantiated and attached to a node.

- ***Install*** on Node containers will return an ApplicationContainer that holds pointers to all of the applications (one in this case since we passed a NodeContainer containing one node)

- Applications require a time to “start” generating traffic and may take an optional time to “stop”. We provide both.

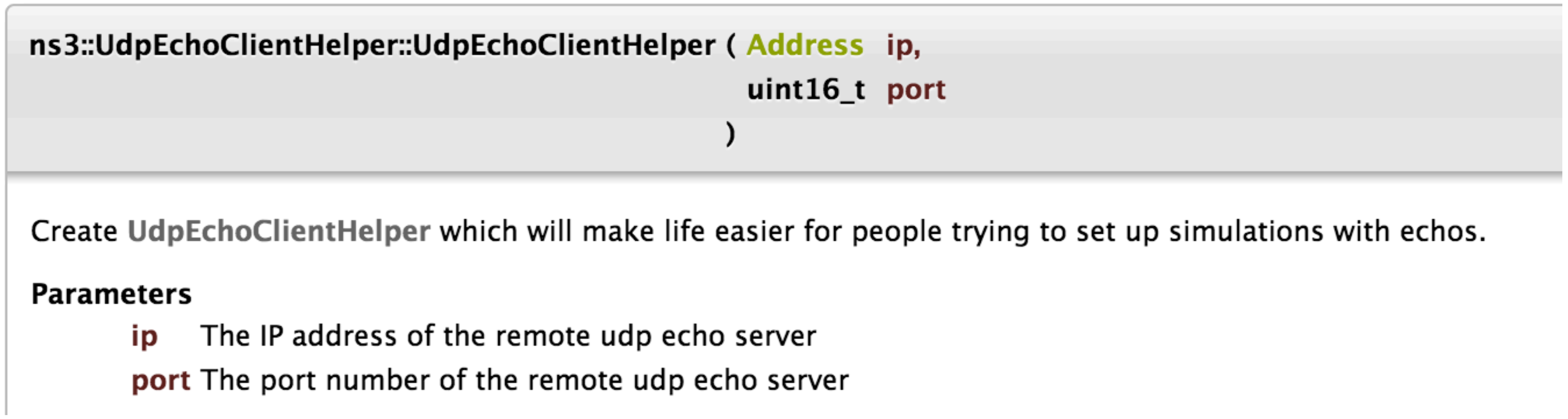
- The simulation will last nine seconds

first.cc



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

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

- The echo client application must be set up in a method similar to that for the server.
- For the client, however, we need to **set five different attributes**:
  - The first two are set during construction of the ***UdpEchoClientHelper.***
  - We pass parameters that are used (internally to the helper) to set the “RemoteAddress” and “RemotePort” *Attributes.* The first interface in the ***Interfaces*** container corresponds to the IP address of the first node in the ***Nodes*** container.



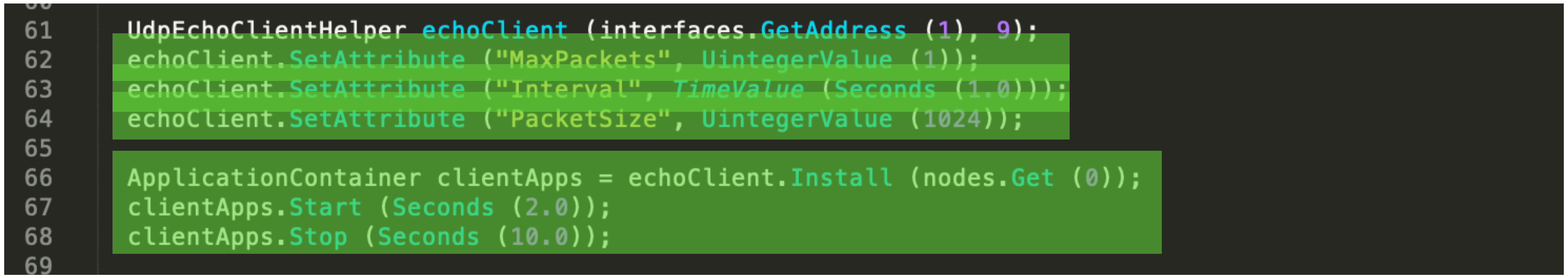
```
ns3::UdpEchoClientHelper::UdpEchoClientHelper ( Address  ip,
                                                uint16_t port
                                              )
```

Create UdpEchoClientHelper which will make life easier for people trying to set up simulations with echos.

Parameters

ip	The IP address of the remote udp echo server
port	The port number of the remote udp echo server

first.cc



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

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

- The “**MaxPackets**” attribute tells the client the maximum number of packets we allow it to send during the simulation.

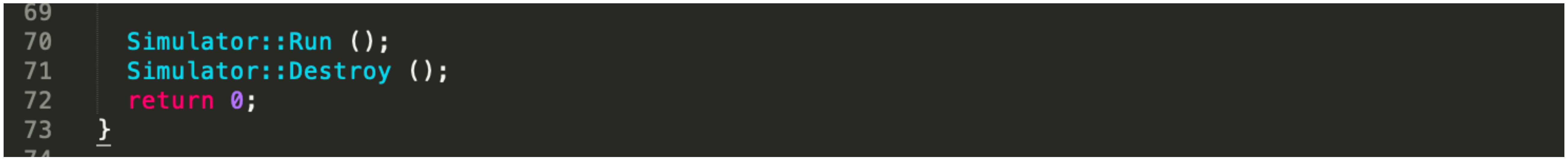
- The “**Interval**” tells the client how long to wait between packets,

- and the “**PacketSize**” tells the client how large its packet payloads should be.

- With **this particular combination of Attributes, we are telling the client to send one 1024-byte packet.**

- Just as in the case of the echo server, we tell the echo client to start and stop, but here we start the client one second after the server is enabled (at two seconds into the simulation)

first.cc



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

- What we finally need to do is to actually run the simulation. This is done using the global function ***Simulator::Run***
- When we previously called the methods serverApps.Start (Seconds(1.0));, etc. we actually scheduled events in the simulator at the respective times.
- When ***Simulator::Run*** is called, **the system will begin looking through the list of scheduled events** and execute them
- The act of sending the packet to the server will **trigger a chain of other background events that will be automatically scheduled** and which will perform the mechanics of the packet echo according to the various timing parameters that we have set in the script.
- Eventually, since we only send one packet (remember application set up earlier), the chain of events triggered by that single client echo request will complete and **the simulation will go idle**.
- When there are no further events to process ***Simulator::Run*** returns. The simulation is then complete. ***Simulator::Destroy()*** is used for clean up (memory allocation, destroy objects, etc.).

Εξαμηνιαία εργασία

- Will be implemented using NS-3, version 3.28-3.30
- Lies in the thematic area of **Wireless HetNets**
- Main objectives:
    1. The creation of a **EPC-LTE deployment (20%)**
    2. The creation of a **Wi-Fi deployment (10%)** sharing the same transport infrastructure with the LTE deployment
    3. In the context of those 2 deployments: **(20%)**
        1. Prove the **end-to-end connectivity** of the two wireless network deployments (i.e., confirm the exchange of packets between both LTE/Wi-Fi network hosts and the Internet)
        2. **Create constant/bursty traffic** models between both LTE/Wi-Fi network hosts and the Internet
            - Simulate **VoIP/Web/FTP download services** exploiting parameters related to packet size, interval, time distribution, etc.
        3. You should also be able to:
            a. **Monitor and export specific network KPIs** using one ore more of the available tools (e.g., LTE module tracing on PDCP layer, Flow Monitor tool, PCAP tracing)

            b. **Understand and discuss the results** (i.e., how specific network KPIs vary over time with UE mobility/with data traffic patterns/associated BSs-APs/available BS-AP resources)

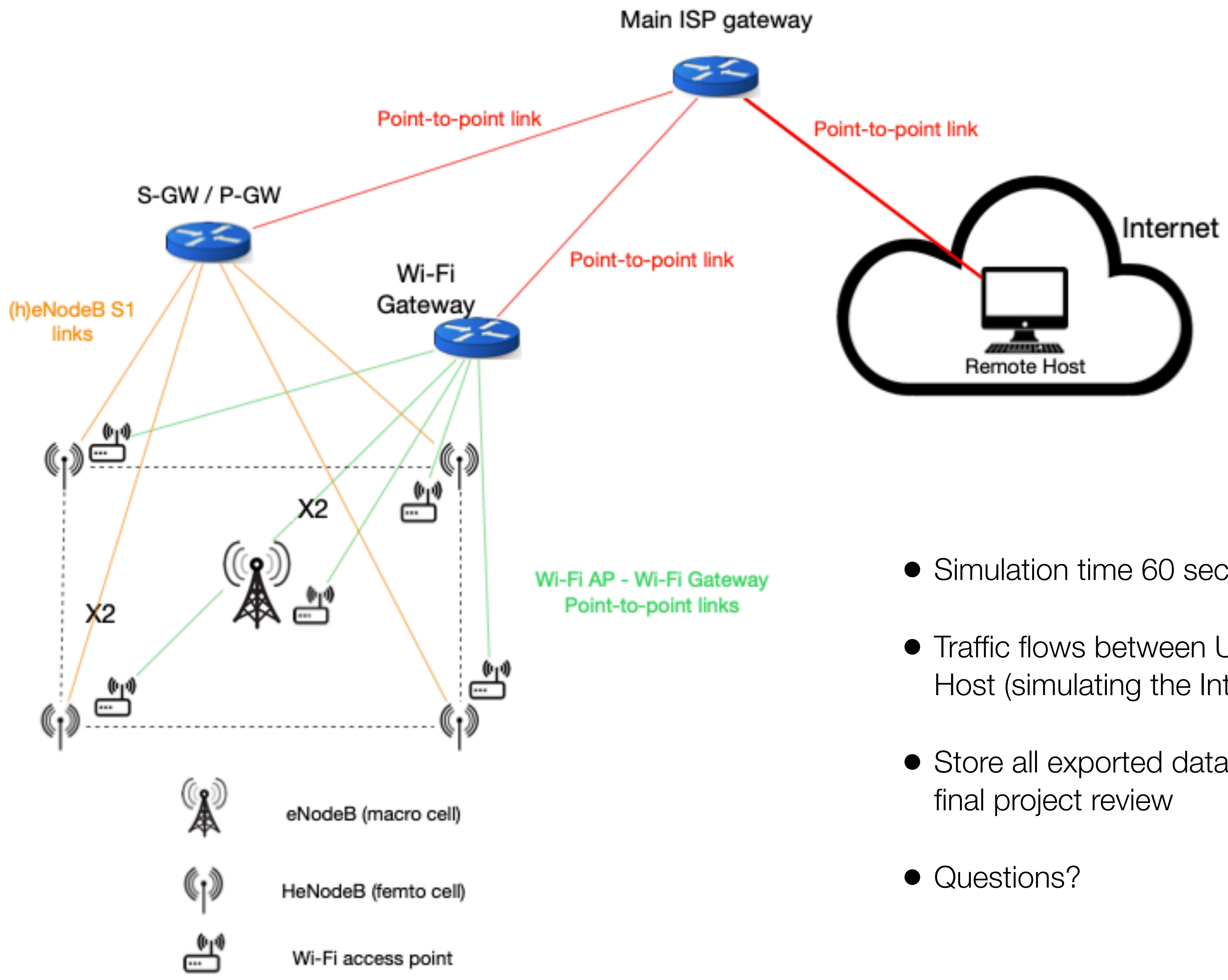
EPC - LTE set-up

- Set-up an **EPC network**
  - **1 x MME, 1 x PGW, 1 x SGW** (PGW/SGW may be one entity depending on the NS-3 version)
- Set-up an **E-UTRAN access network**
  - 4 HeNodeBs/femto cells at the corners of the rectangular area (100 x 200 m)
  - 1 eNodeB/macro cell at the center of the rectangular area
  - All 5 E-UTRAN eNodeBs are X2-interface connected with each other
- Create **10 static and 10 mobile LTE UEs** in a pre-defined rectangular area
  - Use locations for static UEs/mobility models for mobile UEs of your choice
- Create traffic flows between the UEs and the remote host
  - Simulate **VoIP/Web/FTP download services** exploiting parameters related to packet size, interval, time distribution, etc.

Wi-Fi set-up

- Set up a Wi-Fi network that connects to the same transport infrastructure with the LTE **via a Wi-Fi Gateway**
- Install **5 Wi-Fi APs** at the same locations with the E-UTRAN base stations
  - Plan your network selecting the channels of operation of the Wi-Fi APs
- Create **10 static and 10 mobile Wi-Fi hosts**
  - Use locations for static Wi-Fi hosts/mobility models for mobile Wi-Fi hosts of your choice
- Create traffic flows between the Wi-Fi hosts and a remote host
- Simulate **VoIP/Web/FTP download services** exploiting parameters related to packet size, interval, time distribution, etc.

Overview



- Simulation time 60 seconds
- Traffic flows between UEs/WiFi sta nodes and Remote Host (simulating the Internet)
- Store all exported data into files that will be used for the final project review
- Questions?