Networks Simulation
Corso di Tecnologie di Infrastrutture di Reti

Carlo Augusto Grazia

Department of Engineering Enzo Ferrari
University of Modena and Reggio Emilia

Modena, 26 April 2017
Overview

- Motivations
  - What is Simulation
  - Why does it matter?
  - What is Emulation

- Different Simulators
  - Why ns-3 is better?

- Inside ns-3

- Demo/Tutorial
Simulation in a nutshell

Exogenous inputs to system (the environment) → System under study (has deterministic rules governing its behavior) → Observer

“real” life

Pseudo random inputs to system (models environment) → Computer program simulates deterministic rules governing behavior → Observer

“simulated” life
Why simulation

- Real-system not *available*:
  - complexity (e.g. huge networks);
  - cost (e.g. space communications, satellite);
  - dangerous (e.g. PPDR systems, emergency networks).

- Quick *alternatives* evaluation:
  - star/mesh topology;
  - TCP or UDP for an App;
  - WiMAX or LTE connection;
  - ...

- Evaluate *complex* analytical models (optimal formula unavailable):
  - different QoS solutions;
  - optimizations routing problem for IoT;
  - access techniques for challenging environments;
  - ...

Simulation: Pros and Cons

- **Pros**
  - cheaper: quite always;
  - find bugs in advance;
  - generality: over numerical techniques, over topology, ...;
  - detail: tuning the granularity of system details.

- **Cons**
  - accuracy: does the system reflects reality?;
  - large scale system: lot of resources to simulation;
  - may be slow: (computationally expensive, 1 min real time could be hours of simulated time).
Inside Simulation

What’s in a simulation program?

- **simulated time**: internal variable that keeps track of simulated time (could be faster or slower than real time);

- **system “state”**: variables maintained by simulation program; it defines the system “stat” (e.g. track number of packets in queues, current value of TX timer, . . .);

- **events**: points in the time when system changes state:
  - each event has associate *event time* (e.g. enqueue/dequeue event, state changes, . . .);
  - model for time between events (probabilistic) caused by external environment.
Simulation structure:

- simulation program maintains and updates list of future events: the event list;

- well defined set of events;

- for each event there is a simulated system action, update of the event list.
Inside Simulation: A formal view

\[ simulation : (\mathcal{S}, \mathcal{E}^n) \xrightarrow{f} (\mathcal{S}, \mathcal{E}^m) \]

where:
\( \mathcal{S} \) is the state space;
\( \mathcal{E} \) is the event space;
\( \mathcal{E}^n = \{ (e_1, e_2, \ldots, e_n) \mid e_i \in \mathcal{E}, \forall i \in [1, n] \} \).

\[ simulation_{\text{step}} : (s, (e_1, \ldots, e_n)) \mapsto (s', (e_2, \ldots, e_n) \cup (e'_1, \ldots, e'_m)) \]

where:
\((e_1, \ldots, e_n) \in \mathcal{E}^n\) in the current list of event of the system;
\(s \in \mathcal{S}\) is the current system state;
\((e_2, \ldots, e_n) \cup (e'_1, \ldots, e'_m) \in \mathcal{E}^{n+m-1}\) is the new event list of the system;
\(s' \in \mathcal{S}\) is the new system state.
Emulator is an hw/sw that duplicates the functions of one computer system, so that the emulated behaviour closely resembles the behaviour of the real system.

- Common in gaming (Nintendo game over PC . . .);
- A simulation 2.0;
- Real packets over simulated network;
- Simulated packets over real network.
Models of a Simulator

A list of common models “modelled” by a network simulator …

… what is a network!? 
A list of common models “modelled” by a network simulator ... 

... what is a network!?  

- Nodes

- Links
A list of common models “modelled” by a network simulator ...

... what is a network!?

- **Nodes**
  - End-system (host)
  - Router
  - Hub ...

- **Links**
  - Ethernet
  - Point-to-Point
  - Wireless ...
Models of a Simulator

Applications

- Bulk TCP transfer (very common)
- TCP/UDP “on-off” application
- Web Browsing
- P2P file transfer
- Video streaming
- VoIP
- Chat . . .

Protocols

- TCP vs UDP
- IPv4 vs IPv6
- Routing Protocol (BGP, OSPF, . . .)
Models of a Simulator

- **Network Interfaces**
  - Wired/Wireless
  - Layer 2 protocol (802.x family)

- **Packets**
  - Real data vs “Dummy”

- **Routers and Queueing**
  - I/O buffers
  - Route lookup delays
  - Routing table representation
  - Queueing techniques
Output of a Simulator

How to analyse the simulation results?

- **Trace file**
  - Log packet receipt/transmit
  - Log queue size, drop...

- **Built-in statistics gathering**
  - Link utilization
  - Queue occupancy
  - Throughput
  - Loss rate

- **Custom Tracing**
  - User specifies which packets/links/nodes to trace
Simulation Tools

Who is the best?

- **ns2**
  - Original “design” by Steve McCanne
  - OTcl/C++ hybrid
  - open source
  - De-facto standard in academic research (last decade)

- **Georgia Tech Network Simulator (GTNetS)**
  - Completely C++
  - Designed for distributed simulation (scalable)
  - BGP model
Simulation Tools

- **OPNET**
  - Commercial, closed source tool
  - De-facto standard in Military (cash!)
  - Full-Featured, nice GUI
  - Fine-grained data analysis feature

- **QualNet**
  - Commercial, closed source tool
  - Competes primarily with OPNET
  - Strong in Wireless models
Simulation Tools

- **SSFNet**
  - Both Java and C++ versions
  - Designed for “parallel” simulations (multiCore, not distributed)

- **OMNet++**
  - C++ engine
  - Common in European Community

- **MiniNet**
  - Python models
  - Used by the SDN community (OpenFlow paradigm)
  - Full Emulator
Simulation Tools: NS3

Network Simulator 3

discrete-event network simulator for Internet systems

- Partially founded by US NSF grant
- Large Community (Investigators, Programmer, Staff, Volunteers)
- Modular and Scalable software
- Abstraction and Realism (Accurate!)
- Integration, between emulation
- Lot of Modules (WiFi, cellular, ...)  
- Education (examples, tutorials, projects, courses)
- Maintenance (validations, documentation, distribution)
Simulation Tools: NS3 key Features

- Flexible event scheduler
- Output traces in ascii or Pcap (readable with WireShark)
- Emulation mode
  - Integration with real networks or real packets
  - Real-Time Scheduler
- Doxygen documentation
- Mercurial code repo
Simulation Tools: NS3 key Decisions

- Use of “smart pointers” to ease memory management
- Use of “Object Aggregation” to allow easy object extension functionality
- Simulation event scheduling on arbitrary functions with arbitrary argument lists
- Packet objects manage sequential array (easy add/remove headers or data)
ns3 basic model
ns3 protocol stack

Protocol stack encapsulates:
- TCP sockets
- transport protocol
- network protocol
- routing
  ...
<table>
<thead>
<tr>
<th>ns3 current modules</th>
</tr>
</thead>
<tbody>
<tr>
<td>aodv</td>
</tr>
<tr>
<td>click</td>
</tr>
<tr>
<td>csma</td>
</tr>
<tr>
<td>emu</td>
</tr>
<tr>
<td>internet</td>
</tr>
<tr>
<td>mobility</td>
</tr>
<tr>
<td>network</td>
</tr>
<tr>
<td>openflow</td>
</tr>
<tr>
<td>propagation</td>
</tr>
<tr>
<td>tap-bridge</td>
</tr>
<tr>
<td>topology-read</td>
</tr>
<tr>
<td>visualizer</td>
</tr>
<tr>
<td>applications</td>
</tr>
<tr>
<td>config-store</td>
</tr>
<tr>
<td>dama</td>
</tr>
<tr>
<td>energy</td>
</tr>
<tr>
<td>lte</td>
</tr>
<tr>
<td>mpi</td>
</tr>
<tr>
<td>nix-vector-routing</td>
</tr>
<tr>
<td>point-to-point</td>
</tr>
<tr>
<td>point-to-point-layout</td>
</tr>
<tr>
<td>spectrum</td>
</tr>
<tr>
<td>test</td>
</tr>
<tr>
<td>uan</td>
</tr>
<tr>
<td>wifi</td>
</tr>
<tr>
<td>bridge</td>
</tr>
<tr>
<td>core</td>
</tr>
<tr>
<td>dsdv</td>
</tr>
<tr>
<td>flow-monitor</td>
</tr>
<tr>
<td>mesh</td>
</tr>
<tr>
<td>netanim</td>
</tr>
<tr>
<td>olsr</td>
</tr>
<tr>
<td>point-to-point-layout</td>
</tr>
<tr>
<td>stats</td>
</tr>
<tr>
<td>tools</td>
</tr>
<tr>
<td>virtual-net-device</td>
</tr>
<tr>
<td>wimax</td>
</tr>
</tbody>
</table>
Stack: real
Network: simulated
- **Stack**: simulated
- **Network**: real

![Diagram showing the structure of ns3 emulation]
Numbers about ns3

- Line of code: \( \sim 300k \)
- Downloads: \( > 50k \);
- Subscribed users: \( > 3.5k \);
- Developers: \( > 1k \);
- Citations: \( > 100k \)
ns2/ns3 became the main choice for research usage. Source: ACM Digital Library:

<table>
<thead>
<tr>
<th></th>
<th>ns2</th>
<th>OPNET</th>
<th>QualNet</th>
</tr>
</thead>
<tbody>
<tr>
<td>≥ layer 4</td>
<td>123 (75%)</td>
<td>30 (18%)</td>
<td>11 (7%)</td>
</tr>
<tr>
<td>= layer 3</td>
<td>186 (70%)</td>
<td>48 (18%)</td>
<td>31 (12%)</td>
</tr>
<tr>
<td>≤ layer 2</td>
<td>114 (43%)</td>
<td>96 (36%)</td>
<td>55 (21%)</td>
</tr>
</tbody>
</table>

nowadays ns3 moves also conferences, workshops, tutorials and GSoC; ns3 is currently the standard *de-facto* for research purposes.
Tutorial on NON-Simulation

How to perform a **real** TCP connection on a Linux system

Only facing the non-flexibility of a real system we’ll understand the usefulness of simulators
Real TCP Connection on a Real System

We want this:

![Diagram](Diagram.png)

How to run it? We need:

- Two nodes
  - a TCP sender
  - a TCP receiver
- a real (configurable) link/connection between the nodes
- something to generate (the desired) TCP traffic
- a smart way to monitor what’s happening on our system
Real TCP Connection on a Real System

We want this:

![Diagram showing a connection between two nodes n0 and n1 with bandwidth 10 Mbps and delay 20 ms]

Dummy solution:

- Buy two laptop
- Buy a 10mbit/s Ethernet long enough to have 20ms of propagation delay (unreal)
- Configure the two laptops in order to communicate through the cable
- Start a specific TCP connection, generate some traffic and analyse it
First *better* solution:

LXC is a very flexible and easy-to-configure virtual machine well integrated with the main Linux system.
First *better* solution:

We can easily create nodes and let them communicate
First *better* solution:

Install LXC and execute the ex. test: **Step-by-step guide**
Install LXC and create two containers: on Ubuntu

Legend: \textbf{ms} = main system, \textbf{ca} = container a and \textbf{cb} = container b.

Open a Terminal:

\begin{verbatim}
ms$ sudo apt-get install lxc
ms$ sudo lxc-create -t ubuntu -n my-container-a
ms$ sudo lxc-create -t ubuntu -n my-container-b
\end{verbatim}

both \textit{my-container-a} and \textit{my-container-b} have the default config with user/pass equal to ubuntu/ubuntu.

Look at the containers status with:

\begin{verbatim}
ms$ sudo lxc-ls
\end{verbatim}
Move into a container (ca: container a)

On the main system:

```bash
ms$ sudo lxc-start -n my-container-a
ms$ sudo lxc-attach -n my-container-a
```

this command switch environment and we move into a ca shell

Look at the connections of ca with:

```bash
ca$ ifconfig
```

we focus on eth0, we refer to the ip address of ca as eth0_a
Move into a container \((cb : \text{container b})\)

On the main system:

\[
\begin{align*}
\text{ms}\$ & \text{ sudo lxc-start -n my-container-b} \\
\text{ms}\$ & \text{ sudo lxc-attach -n my-container-b}
\end{align*}
\]

this command switch environment and we move into a \(cb\) shell

Look at the connections of \(cb\) with:

\[
\begin{align*}
\text{cb}\$ & \text{ ifconfig}
\end{align*}
\]

we focus on \(eth0\), we refer to the ip address of \(cb\) as \(eth0_b\)
Create the TCP connection \textit{ca -> cb}

On \textit{cb}, our server:

\begin{verbatim}
cb$ sudo apt-get install iperf
cb$ sudo iperf -s
\end{verbatim}

On \textit{ca}, our client:

\begin{verbatim}
ca$ sudo apt-get install iperf
ca$ sudo iperf -c eth0 -t 5
\end{verbatim}

Good news: \textit{ca -> cb} works!
Monitor the connection $ca \rightarrow cb$

On the main system:

```bash
ms$ wireshark
```

or simply open the program "graphically" by clicking on it

Start to monitor the $lxcbr0$ interface, created by the main system immediately after the creation of the first lxc. Start again the client $ca$, what you will see is this:

(continue)
Monitor the connection $ca \rightarrow cb$

On WireShark: Statistics $\rightarrow$ IO Graph $\rightarrow$ done!
But remember...

We want this:

![Diagram showing network nodes n0 and n1 with bandwidth 10 Mbps and delay 20 ms]

But we have this:

![Wireshark graph showing network traffic]
Set BandWidth and Delay on both \textit{ca} and \textit{cb}

On the container \textit{ca}:

\texttt{ca$ sudo tc qdisc add dev eth0 handle 1: root htb default 11}
\texttt{ca$ sudo tc class add dev eth0 parent 1: classid 1:1 htb rate 1.25mbps}
\texttt{ca$ sudo tc class add dev eth0 parent 1:1 classid 1:11 htb rate 1.25mbps}
\texttt{ca$ sudo tc qdisc add dev eth0 parent 1:11 handle 10:1 netem delay 20ms}

do the same on \textit{cb}

\textbf{NOTE}

mbps for \textit{tc} tool is MBps actually

painful? not as a \textbf{real} system...but if we want more...this is not enough
Monitor the connection $ca \rightarrow cb$

Start again WireShark monitoring $lxcbr0$, start the client $ca$ and:
Tutorial on ns3
Install and Execute
Install ns3: Using git as source code manager

On Ubuntu:

$ git clone https://github.com/nsnam/ns-3-dev-git.git
$ cd ns-3-dev-git
$ ./waf configure --enable-examples
$ ./waf --run first

On Mac OSx

$ git clone https://github.com/nsnam/ns-3-dev-git.git
$ cd ns-3-dev-git
$ ./waf configure --enable-examples
$ ./waf --run first
Run first example: TCP Bulk

Draft of the example:

```
n0 n1
BW 10 Mbps
Delay 20 ms
```

How to run it?

```
$ ./waf -run "tcp-bulk-send -tracing"
$ cat tcp-bulk-send.tr
$ tcpdump -tt -r tcp-bulk-send-0-0.pcap
$ wireshark tcp-bulk-send-0-0.pcap
```
Learn from first example: TCP Bulk

The goal is:

![Network Diagram]

- Node n0
- BW 10 Mbps
- Delay 20 ms
- Node n1
Learn from first example: TCP Bulk

Step 0 header and main:

```cpp
#include <string>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/internet-module.h"
#include "ns3/applications-module.h"
#include "ns3/network-module.h"
#include "ns3/packet-sink.h"

using namespace ns3;

NS_LOG_COMPONENT_DEFINE ("TcpBulkSendExample");

int main (int argc, char *argv[]) {
  bool tracing = false;
  uint32_t maxBytes = 0;
```
Step 1 create the nodes:

```c
// Explicitly create the nodes required by the topology (shown above).
//
NS_LOG_INFO("Create nodes.");
NodeContainer nodes;
nodes.Create(2);
```
Learn from first example: TCP Bulk

Step 2 create the link:

```
NS_LOG_INFO("Create channels.");
//
// Explicitly create the point-to-point link required by the
topology (shown above).
//
PointToPointHelper pointToPoint;
pointToPoint.SetDeviceAttribute("DataRate", StringValue("10Mbps"));
pointToPoint.SetChannelAttribute("Delay", StringValue("20ms"));
```
Learn from first example: TCP Bulk

Step 3 connect nodes and link:

```
NetDeviceContainer devices;
devices = pointToPoint.Install (nodes);
//
// Install the internet stack on the nodes
//
InternetStackHelper internet;
internet.Install (nodes);
```
Learn from first example: TCP Bulk

Step 4 configure the network:

```
//
// We’ve got the "hardware" in place. Now we need to add IP addresses.
//
NS_LOG_INFO("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i = ipv4.Assign (devices);
```
Learn from first example: TCP Bulk

Step 5 create the application:

```cpp
NS_LOG_INFO ("Create Applications.");

// Create a BulkSendApplication and install it on node 0

uint16_t port = 9; // well-known echo port number
BulkSendHelper source ("ns3::TcpSocketFactory",
    InetSocketAddress (i.GetAddress (1), port));

// Set the amount of data to send in bytes. Zero is unlimited.
source.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
ApplicationContainer sourceApps = source.Install (nodes.Get (0));
sourceApps.Start (Seconds (0.0));
sourceApps.Stop (Seconds (5.0));
```
Learn from first example: TCP Bulk

Step 6 create the receiver socket:

```cpp
// Create a PacketSinkApplication and install it on node 1
PacketSinkHelper sink ("ns3::TcpSocketFactory",
    InetSocketAddress (Ipv4Address::GetAny (),
        port));
ApplicationContainer sinkApps = sink.Install (nodes.Get (1));
sinkApps.Start (Seconds (0.0));
sinkApps.Stop (Seconds (5.0));
```
Learn from first example: TCP Bulk

Step 7 set up tracing:

```cpp
// Set up tracing if enabled
if (tracing) {
    AsciiTraceHelper ascii;
    pointToPoint.EnableAsciiAll (ascii.CreateFileStream ("tcp-bulk-send.tr");
    pointToPoint.EnablePcapAll ("tcp-bulk-send", false);
}
```
Learn from first example: TCP Bulk

Step 8 actual simulation:

```c++
// Now, do the actual simulation.
NS_LOG_INFO("Run Simulation.");
Simulator::Stop (Seconds (10.0));
Simulator::Run ()
Simulator::Destroy ()
NS_LOG_INFO("Done.");
```
Demo on TCP Bulk

Pcap analysis of TCP Bulk example with WireShark:
Run second example: Global Routing

Draft of the example:

```
+-----+ 1 Mbps  2 ms  +-----+
| n0  |             | n2  |
+-----+  2 ms  5 Mbps +-----+
| n1  |             |     |
+-----+ 15 Mbps  10 ms +-----+
|     |             | n3  |
```

How to run it?

```
$ ./waf --run simple-global-routing
$

$ cat simple-global-routing.tr

$ tcpdump -tt -r simple-global-routing-2-3.pcap

$ wireshark simple-global-routing-2-3.pcap
```
Learn from second example: Global Routing

The goal is:

```
 n0
  1 Mbps  2 ms
  n2
  15 Mbps 10 ms
 n2
  5 Mbps  2 ms
 n1
  2 ms
 n3
```
Learn from second example: Global Routing

Step 1 create the nodes:

n0
n1
n2
n3

// Here, we will explicitly create four nodes. In more sophisticated topologies, we could configure a node factory.
NS_LOG_INFO ("Create nodes.");
NodeContainer c;
c.Create (4);
NodeContainer n0n2 = NodeContainer (c.Get (0), c.Get (2));
NodeContainer n1n2 = NodeContainer (c.Get (1), c.Get (2));
NodeContainer n3n2 = NodeContainer (c.Get (3), c.Get (2));

InternetStackHelper internet;
internet.Install (c);
Learn from second example: Global Routing

Step 2 create the link:

```
// We create the channels first without any IP addressing information
NS_LOG_INFO ("Create channels.");
PointToPointHelper p2p;
p2p.SetDeviceAttribute ("DataRate", StringValue ("1Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("2ms"));
NetDeviceContainer d0d2 = p2p.Install (n0n2);

p2p.SetDeviceAttribute ("DataRate", StringValue ("5Mbps"));
NetDeviceContainer d1d2 = p2p.Install (n1n2);

p2p.SetDeviceAttribute ("DataRate", StringValue ("15Mbps"));
p2p.SetChannelAttribute ("Delay", StringValue ("10ms"));
NetDeviceContainer d3d2 = p2p.Install (n3n2);
```
Learn from second example: Global Routing

Step 3 configure the network:

```
// Later, we add IP addresses.
NS_LOG_INFO("Assign IP Addresses.");
Ipv4AddressHelper ipv4;
ipv4.SetBase("10.1.1.0", "255.255.255.0");
Ipv4InterfaceContainer i0i2 = ipv4.Assign((d0d2));

ipv4.SetBase("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer i1i2 = ipv4.Assign((d1d2));

ipv4.SetBase("10.1.3.0", "255.255.255.0");
Ipv4InterfaceContainer i3i2 = ipv4.Assign((d3d2));

// Create router nodes, initialize routing database and set up
// the routing
// tables in the nodes.
Ipv4GlobalRoutingHelper::PopulateRoutingTables();
```
Learn from second example: Global Routing

Step 4 create first app sender:

```c
// Create the OnOff application to send UDP datagrams of size
// 210 bytes at a rate of 1 Mb/s
NS_LOG_INFO("Create Applications.");
uint16_t port = 9;     // Discard port (RFC 863)
OnOffHelper onoff("ns3::UdpSocketFactory",
                   Address(InetSocketAddress(i3i2.GetAddress(0), port)));
onoff.SetConstantRate(DataRate("1Mb/s"));
ApplicationContainer apps = onoff.Install(c.Get(0));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));
```
Learn from second example: Global Routing

Step 5 create first app receiver:

```
// Create a packet sink to receive these packets
PacketSinkHelper sink("ns3::UdpSocketFactory",
    Address(InetSocketAddress(IPv4Address::GetAny(), port)));

apps = sink.Install(c.Get(3));
apps.Start(Seconds(1.0));
apps.Stop(Seconds(10.0));
```
Learn from second example: Global Routing

Step 6 create second app sender:

// Create a similar flow from n3 to n1, starting at time 1.1 seconds
onoff.SetAttribute ("Remote",
    AddressValue (InetSocketAddress (i1i2.
        GetAddress (0), port)));

onoff.SetConstantRate (DataRate ("5Mb/s"));
apps = onoff.Install (c.Get (3));
apps.Start (Seconds (1.1));
apps.Stop (Seconds (10.0));
Learn from second example: Global Routing

Step 7 create second app receiver:

```java
// Create a packet sink to receive these packets
apps = sink.Install (c.Get (1));
apps.Start (Seconds (1.1));
apps.Stop (Seconds (10.0));
```
Learn from second example: Global Routing

Step 8 set up tracing:

AsciiTraceHelper ascii;
p2p.EnableAsciiAll (ascii.CreateFileStream ("simple-global-routing.tr"));
p2p.EnablePcapAll ("simple-global-routing");
Learn from second example: Global Routing

Step 9 actual simulation:

```cpp
NS_LOG_INFO("Run Simulation.");
Simulator::Stop (Seconds (12));
Simulator::Run ();
NS_LOG_INFO("Done.");
Simulator::Destroy ();
return 0;
```
Demo on Global Routing

Pcap analysis of Global Routing example with WireShark:

![Graph showing packet capture analysis with Wireshark]
Next steps with ns3

- Download it (see slide 46 of this presentation for details)
- Try it (look at the examples of this presentation)
- Play it (tune your own code)
- Do not forget to use WireShark

Next practical lesson: end of the course

- Bring your own laptop (with ns-3 on)
  - you can’t? form a group
    - you can’t? follow at least. Don’t worry
- Follow the lesson actively
- Try to solve some exercises (together)
Exam Proposals about ns3

- MultiPath-TCP
- TCP variants (like Cubic, default linux TCP)
- Performance measurements
- Narrow time measurement
- Cross-layer message passing
- User mobility study
- AQM algorithms (queueing discipline)
Resources

- ns3 website: http://www.nsnam.org
- Developer mailing list: http://mailman.isi.edu/mailman/listinfo/ns-developers
- User mailing list: http://groups.google.com/group/ns-3-users
- Tutorial: http://www.nsnam.org/docs/tutorial/tutorial.html
- Code server: http://code.nsnam.org
Contacts

carloaugusto.grazia@unimore.it