NEPI: Network Experiment Programming Interface

Alina Quereilhac
{alina.quereilhac@inria.fr}
Team DIANA
INRIA Sophia Antipolis, France

Network experiment resources

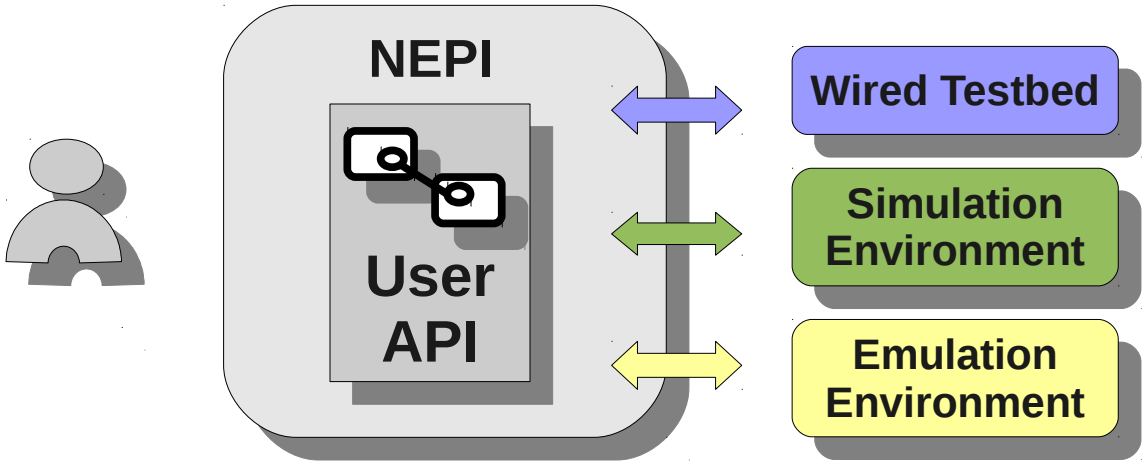
- To conduct network experiments we need resources
  - A resource can be a node in your lab or in a public testbed, a virtual machine, a ns-3 simulation, or even an application …
- There is a large offer of resources for network experimentation provided by different platforms
- But different platforms are accessed and used in different ways, making it necessary to master different tools and technologies

Network experiment resources

How to make it easier to take advantage of the wide offer of network experimentation resources?

NEPI - One tool for many platforms

- NEPI is a tool that provides a uniform API to run experiments on many platforms
  - Allows to manage resources on different platforms using a same tool
  - Allows to mix simulated, emulated and live resources on a same experiment



NEPI - Network experiment management

- NEPI is a framework to manage network experiments
  - Supports different stages of experiment life-cycle

Design

(Offline)

Execution

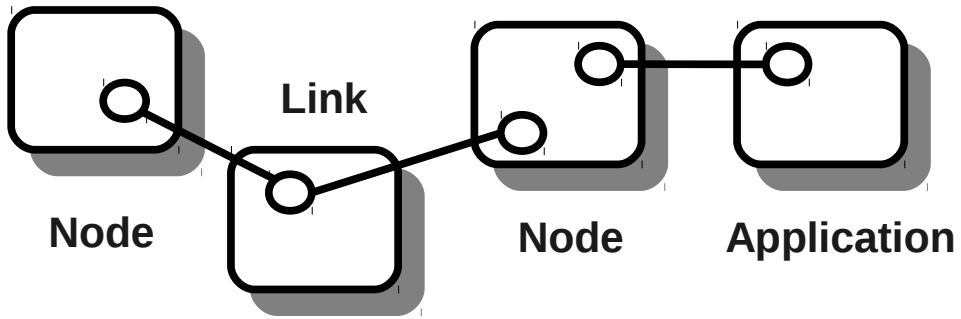
Experiment design

Experiment design

- Generate XML describing an experiment
  - Describe resources to be used (**e.g. nodes, channels,etc**)
  - Describe resource relationships(**e.g. app1 runs on node1**)
  - Describe resource configuration
  - Describe results to be collected
- Provide enough detail to enable reproduction
- This XML will be used as input for execution

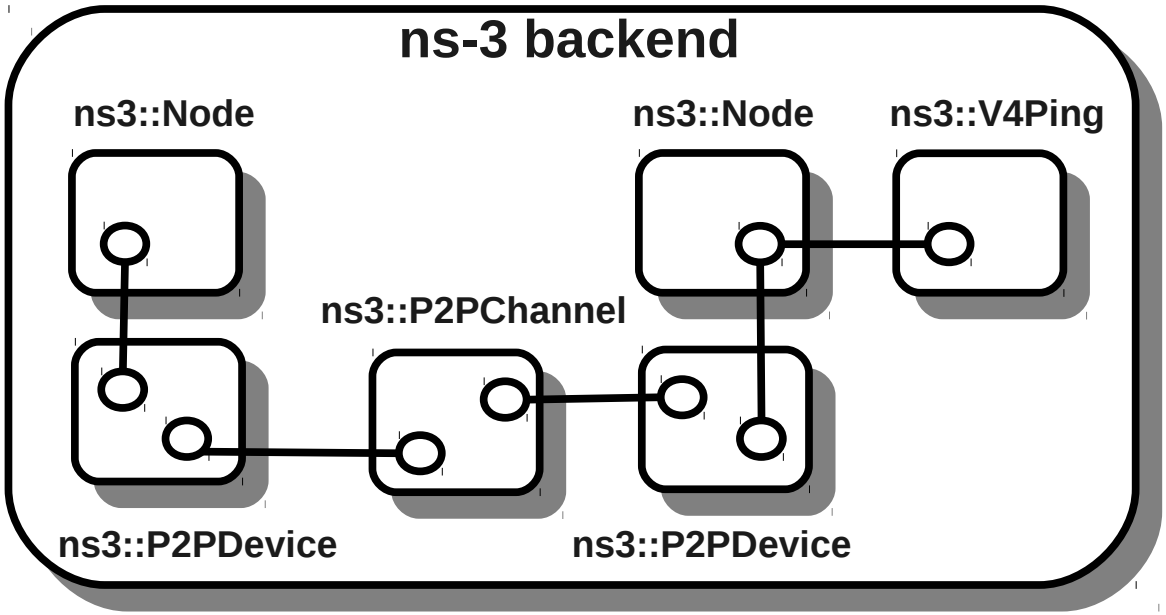
Experiment representation

- An experiment is described as a graph of 'Boxes and Connectors'
  - Boxes represent resources
  - Connectors define constraints between resources
  - Boxes have attributes
  - Boxes are associated to traces (results)



Boxes & Backends

- Boxes have types (e.g. ns3::Node, Planetlab::Node)
- Boxes belong to only 1 backend (platform)
- Backend instances are represented as squares
- Boxes are assigned a Global Unique Identifier (**guid**)

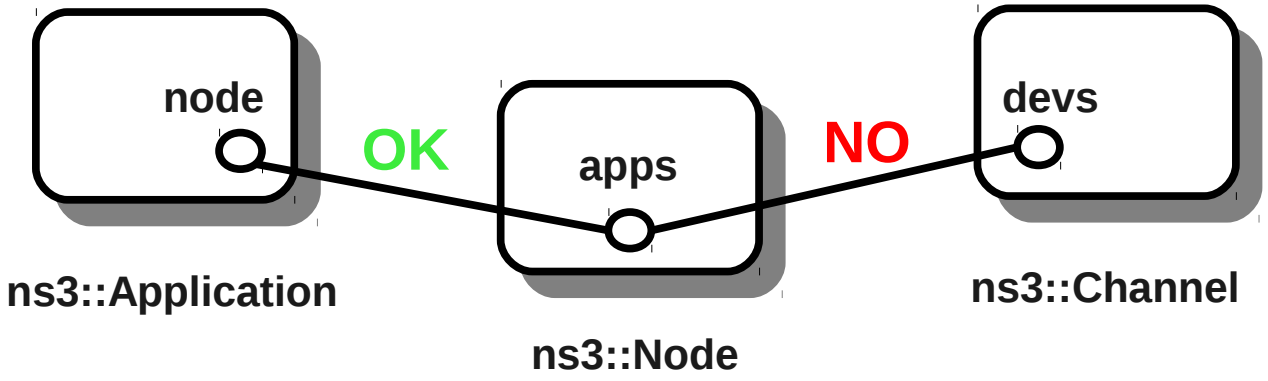


Connectors

- Connectors are identified by names
- Boxes can have many connectors
- Not all boxes can be connected to all connectors
- There are rules for allowed connections defined by:

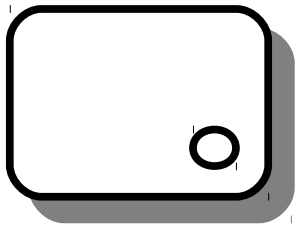
  (BoxType1, ConnectorType1, BoxType2, ConnectorType2)

- Connection rules are mapped to deployment behavior during experiment deployment



Attributes

- Boxes hold a list of attributes
- Attributes expose the resource configuration
- Attributes are define by {name, value, type}
- The attribute type allows to validate the value



PlanetLab::Node

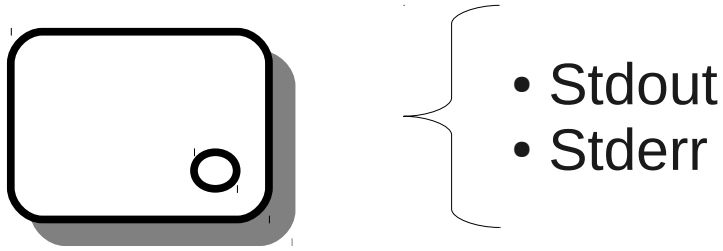
Hostname: nepi1.pl.sophia.inria.fr – String

Min. CPU: 30 – Integer

Architecture: x86_64 - String

Traces

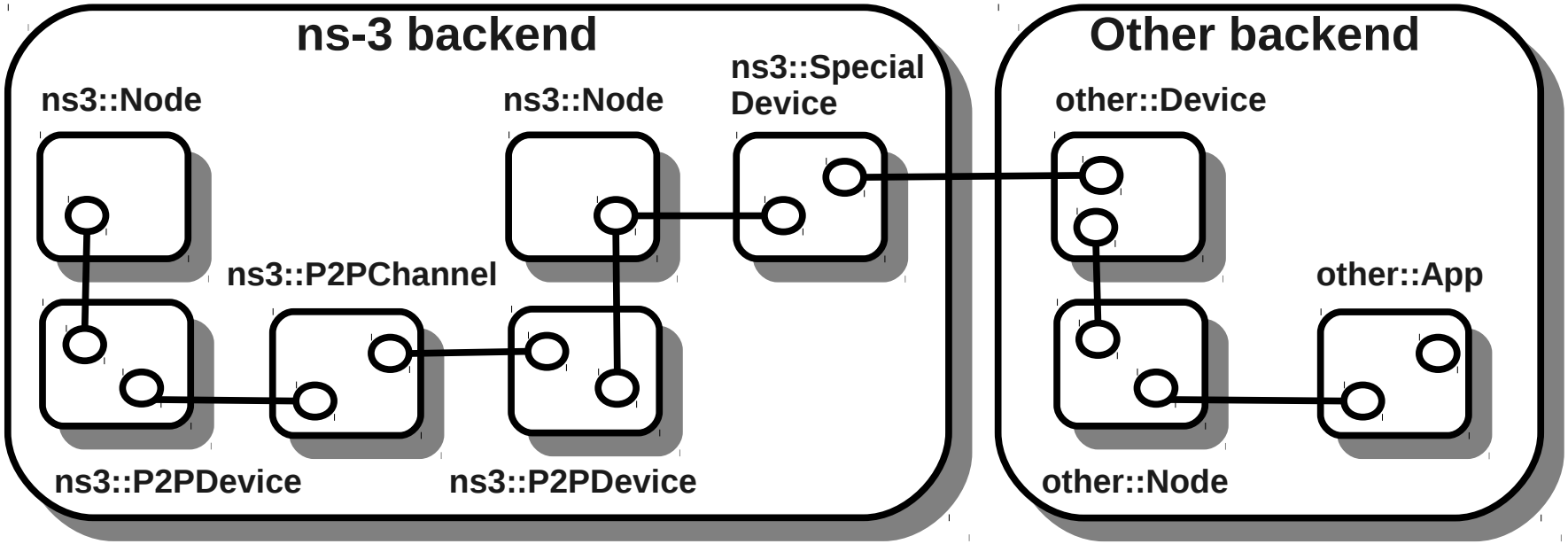
- Boxes hold a list of traces which can be activated
- A trace defines data to be collected into a file during experiment execution
- This data can be obtained from measurements or application output (e.g. stderr, tcpdump)
- Different boxes expose different traces



PlanetLab::Application

Hybrid experiment design

- Boxes from different testbeds can be interconnected as well
- Connections are not arbitrary (e.g. can't connect a ns3::V4Ping to a PlanetLab::Node)

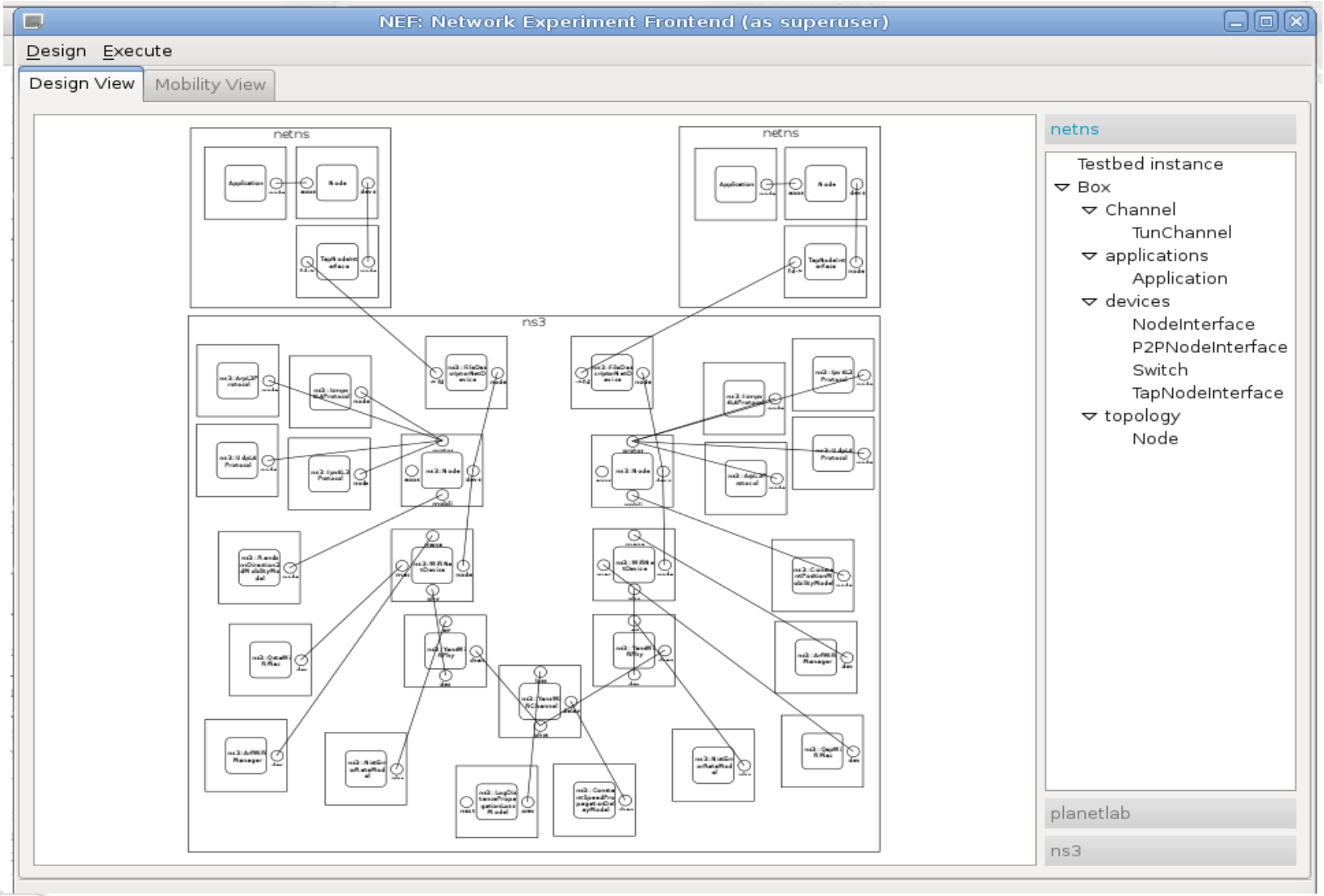


So, how do we actually design an experiment using NEPI?

Using NEPI

- There are 2 ways of using NEPI
  - NEF (graphical user interface)
    - Is a PyQt desktop application
    - Allows to Drag&Drop Boxes & interconnect them
  - Python script
    - NEPI is implemented in Python
    - The nepi.design module provides design support

NEF, Network Experiment Frontend



Python script – design I

- Import NEPI design module

```
from nepi.core.design import ExperimentDescription, FactoriesProvider
```

- Instantiate ExperimentDescription

```
exp_desc = ExperimentDescription ()
```

- Create a backend instance (testbed description)

```
testbed_id = "ns3"
provider = FactoriesProvider(testbed_id)
tbd_desc = exp_desc.add_testbed_description(provider)
```

Python script – design II

- Create and configure boxes

```
chan = tbd_desc.create("ns3::PointToPointChannel")
chan.set_attribute_value("Delay", "0ns")
```

- Interconnect boxes using connectors

```
iface = ns3_desc.create("ns3::PointToPointNetDevice")
iface.connector("chan").connect(chan.connector("dev2"))
```

- Add IP addresses

```
ip = iface.add_address()
ip.set_attribute_value("Address", "10.0.0.2")
```

- Enable traces

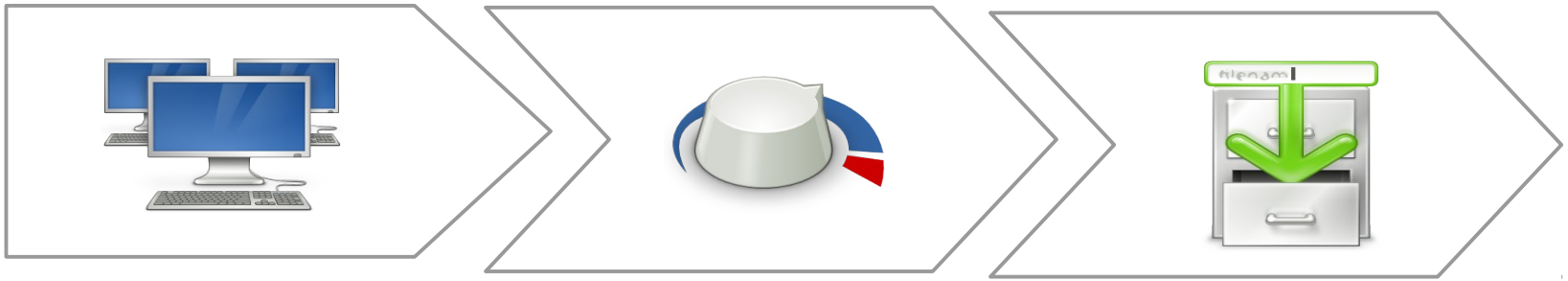
```
iface.enable_trace(“P2PPcapTrace”)
```

Experiment execution

Experiment execution

- Different stages of execution

Deployment | **Control** | **Result collection**



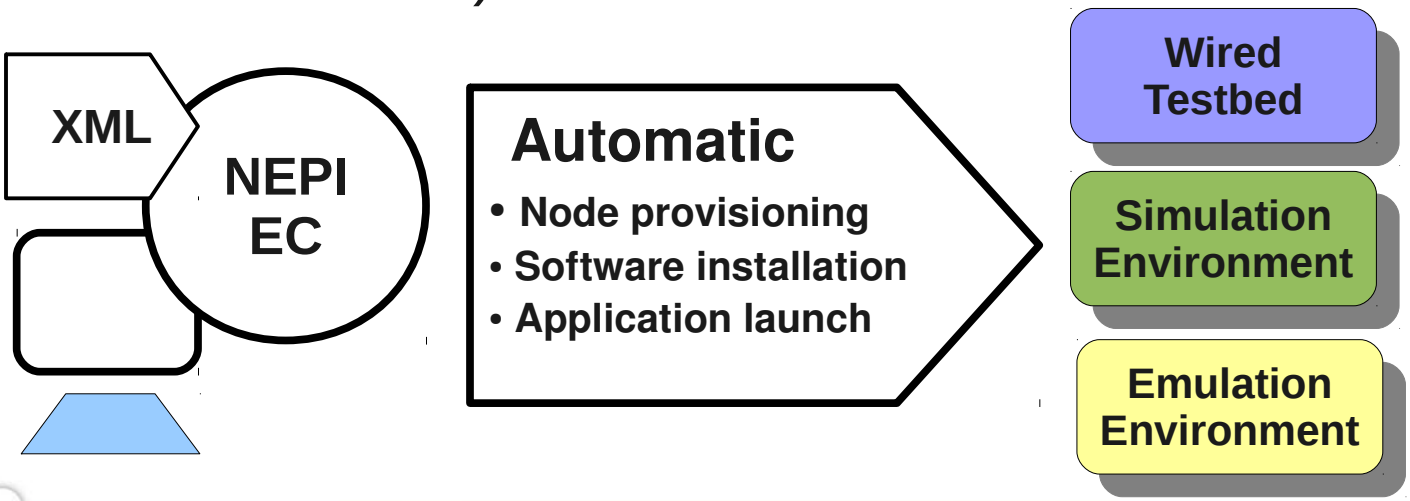
- **Resource discovery & provision**
- **Resource configuration**
- **Software installation**
- **Application launch**

- **Modify configuration**
- **Monitor running status**

- **Download result files**

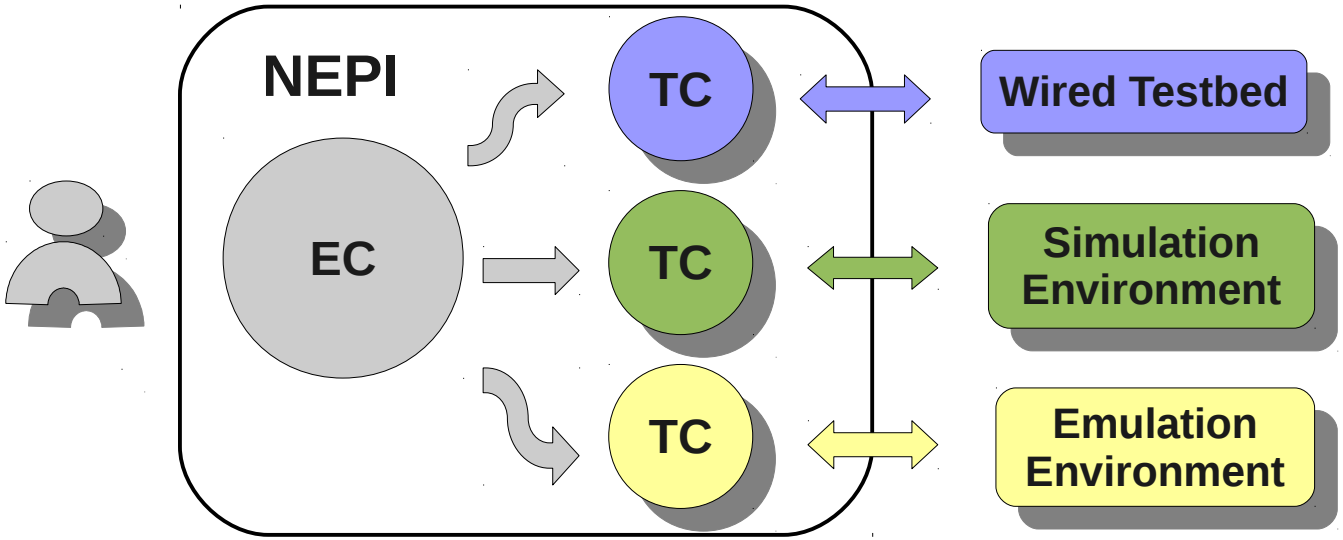
Experiment Controller

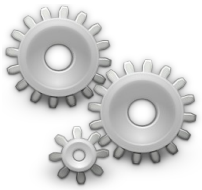
- The Experiment Controller (EC) is the entity responsible to orchestrate execution
- The EC receives as input the XML experiment description generated during design
- The EC can be launched from a user machine and automates experiment deployment (without user intervention)



Tesbed controllers

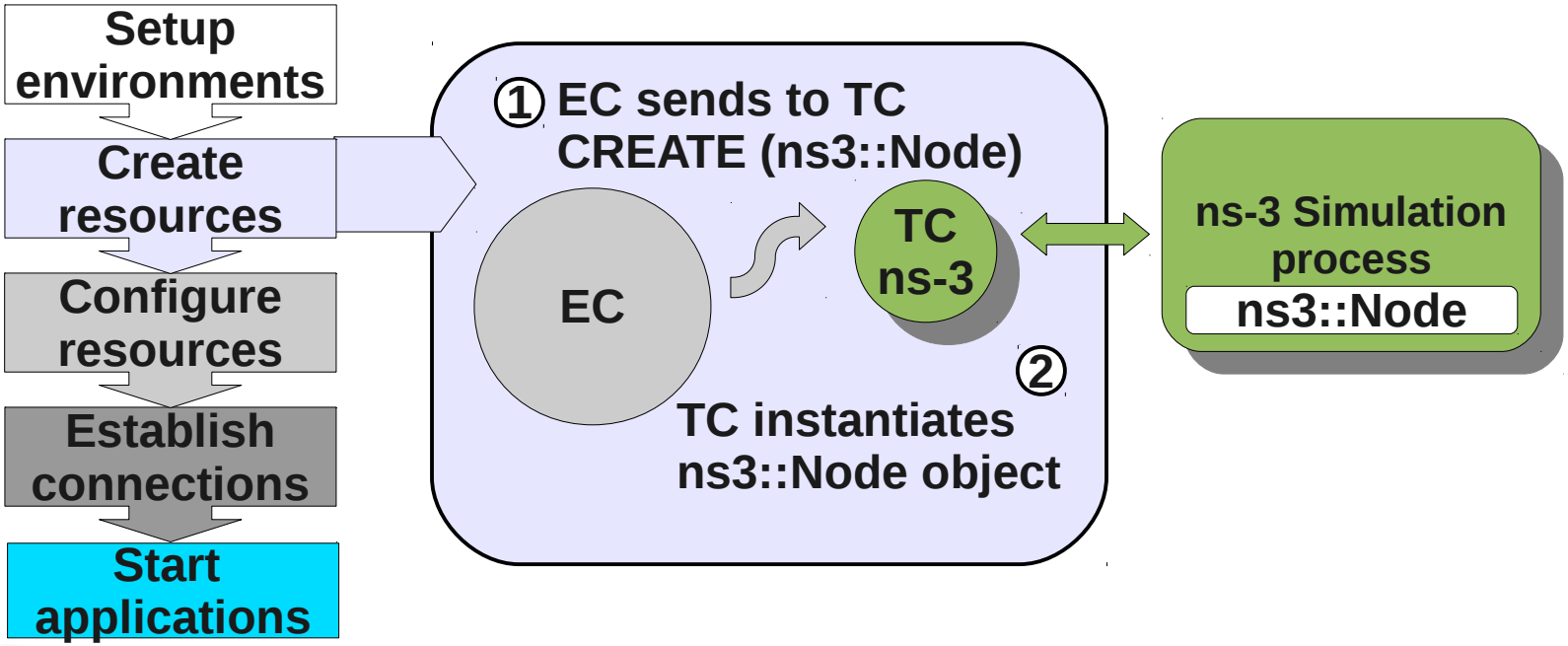
- NEPI uses two levels of controllers
  - One global generic experiment controller (EC)
  - Many testbed controllers (TC)
- TCs “know” about environment specific behavior
- New environments can be supported by implementing new testbed controllers

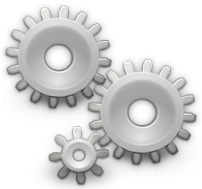




Experiment deployment

- Deployment consists of a sequence of predefined steps
- The EC sends messages to instruct TCs to perform required actions on each step



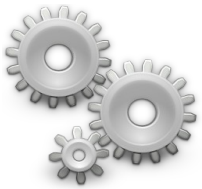


Configure traces

- During resource configuration a TC will invoke the trace function for all enabled traces
- Trace functions are defined by the developer of a NEPI backend (not by the user)

```
def p2ppcap_trace(testbed_instance, guid, trace_id):
    node_guid = _get_node_guid(testbed_instance, guid)
    element = testbed_instance._elements[guid]
    filename = "trace-p2p-node-%d-dev-%d.pcap" % (node_guid, guid)
    filepath = _follow_trace(testbed_instance, guid, trace_id, filename)
    helper = testbed_instance.ns3.PointToPointHelper()
    helper.EnablePcap(filepath, element, explicitFilename = True)
```

- Traces generate result files that are stored locally where they were generated and can be downloaded by the user at any moment



Establish connections

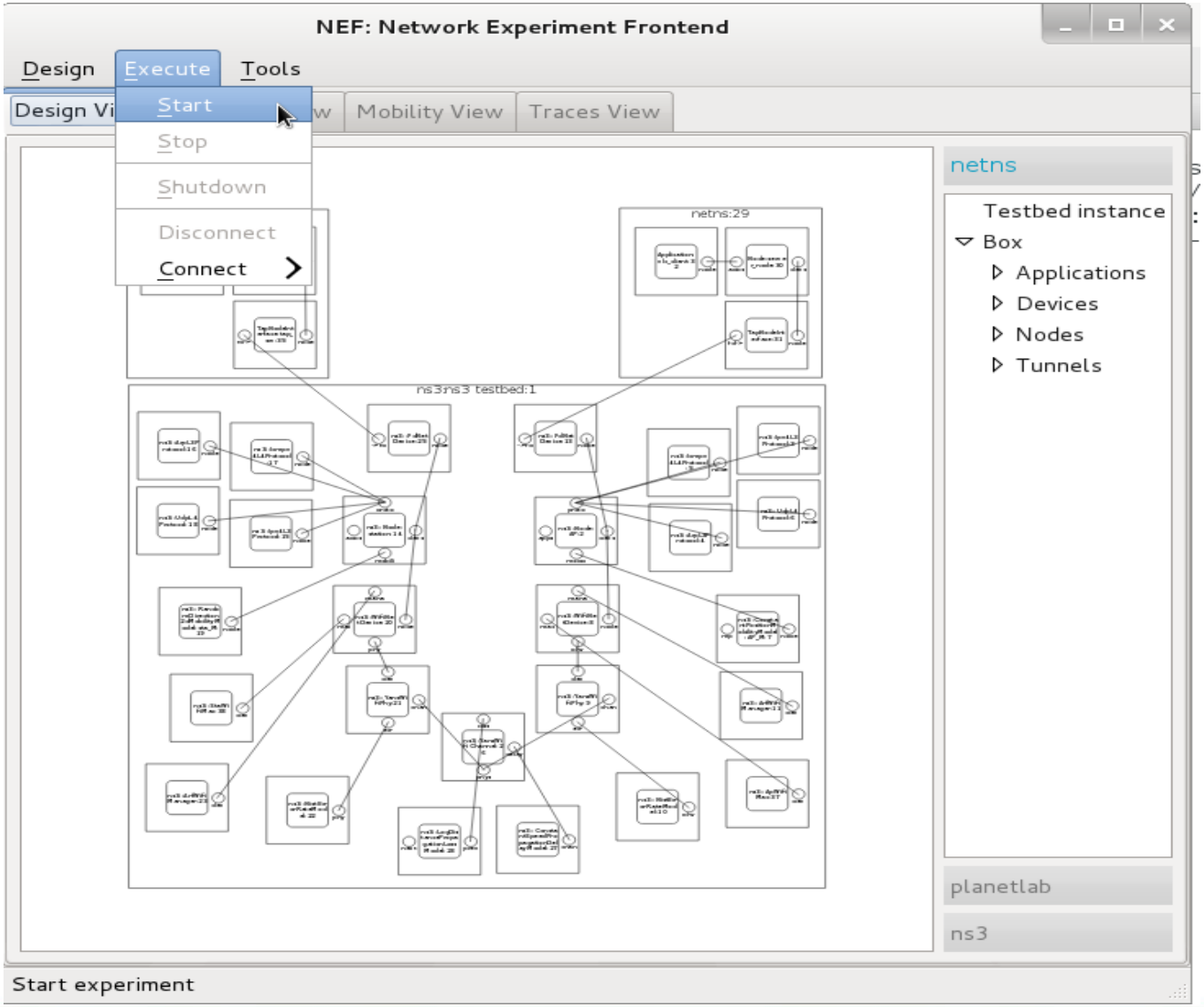
- Connection rules are mapped to connection functions
  (BoxType1, ConnectorType1, BoxType2, ConnectorType2)
- A connection function receives the guids of the boxes to be connected

```
def connect_node_device(testbed_instance, node_guid, device_guid):
    node = testbed_instance._elements[node_guid]
    device = testbed_instance._elements[device_guid]
    node.AddDevice(device)
```

- The EC will automatically invoke the connection functions for all connections during deployment
- Connection functions are defined by the developer of a NEPI backend (not by the user)

How do we run an experiment with NEPI?

Run experiment using NEF



Python script – experiment start

- Import NEPI execution module

```
from nepi.core.execute import ExperimentController
```

- General XML experiment description

```
xml = exp_desc.to_xml()
```

- Instantiate the experiment controller (EC)

```
controller = ExperimentController(xml)
```

- Start the experiment

```
controller.start()
```

Python script – experiment control

- Modify configuration during run-time

```
time.sleep(5)
controller.set(chan.guid, "Delay", "10s")
time.sleep(5)
controller.set(chan.guid, "Delay", "0s")
```

- Wait until some application has finished

```
while not controller.is_finished(app.guid):
    time.sleep(0.5)
```

Python script – experiment control

- Modify configuration during run-time

```
time.sleep(5)
controller.set(chan.guid, "Delay", "10s")
time.sleep(5)
controller.set(chan.guid, "Delay", "0s")
```

- Wait until some application has finished

```
while not controller.is_finished(app.guid):
    time.sleep(0.5)
```

- For the moment control capabilities are limited
- We are working to improve control API
  - Start application X after application Y started
  - Start application X at time T

Python script – result collect

- A result can be retrieved from any remote location invoking the “trace” method

```
result = controller.trace(iface.guid, "P2PPcapTrace")
```

- Then it can be stored in a local file

```
f = open(“result.pcap”, “w”)
f.write(result)
f.close()
```

- Results can be retrieved while the experiment is running

Python script – experiment stop

- Stopping the controller stops running applications and flushes result files

```
controller.stop()
```

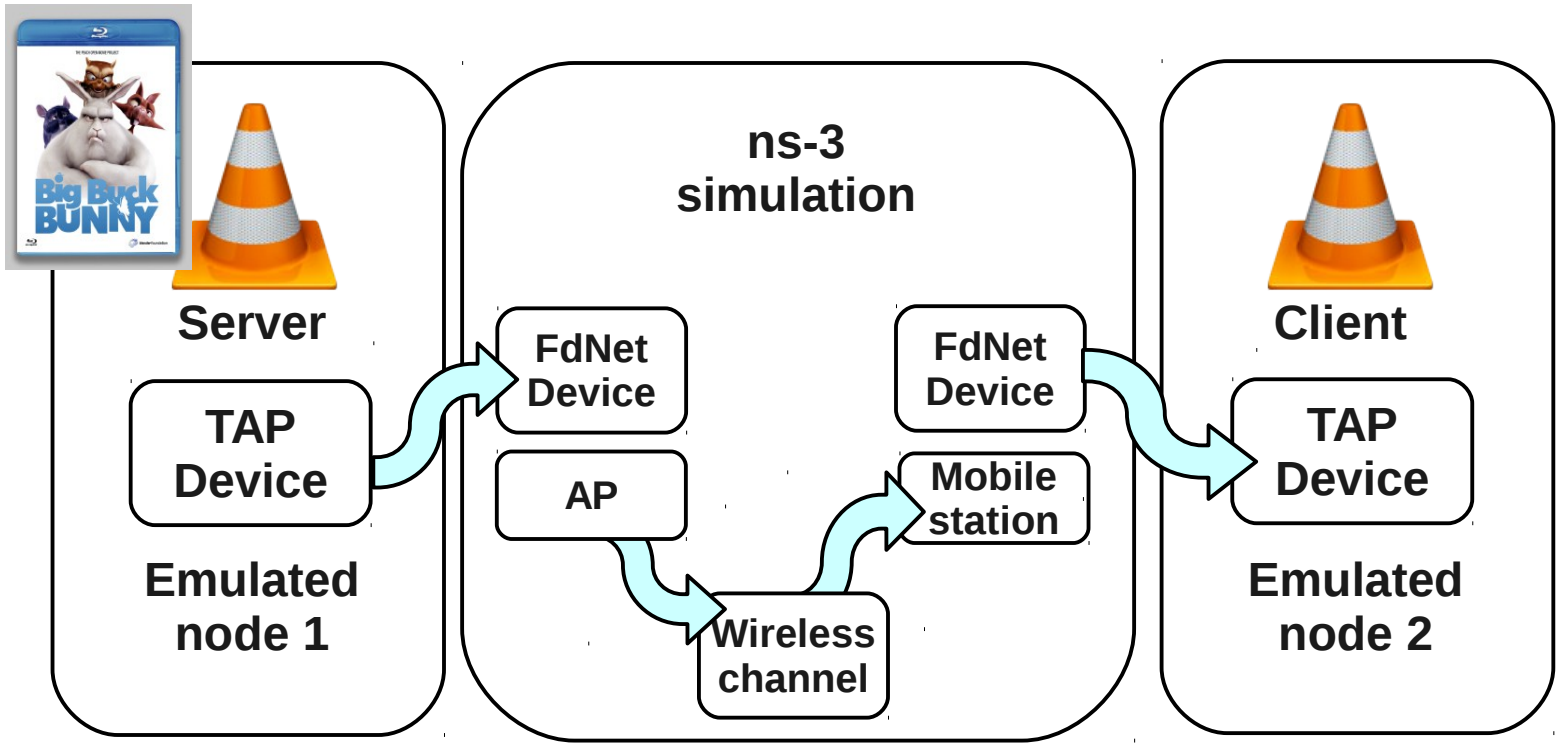
- Controller shutdown releases resources. After shutdown results are no longer available

```
controller.shutdown()
```

Hybrid experiment example

Hybrid experiment

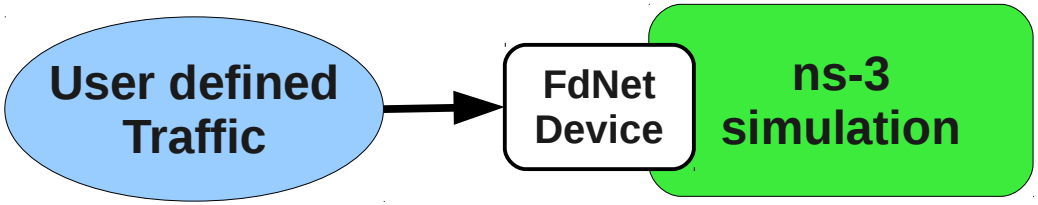
- Can we use ns-3 to evaluate video traffic on mobile Wireless environments without implementing a video traffic model?



Hybrid experiment demo with NEF

ns3 ::FdNetDevice

- FdNetDevice is a ns-3 device which can read and write traffic using a file descriptor provided by the user
  - The file descriptor can be associated to a TAP device, to a raw socket, to a user space process generating and consuming traffic, etc
  - The user can have full freedom to define how external traffic is generated and ns-3 traffic is consumed
  - Can be used independently from NEPI
- **For more info** → http://nepi.inria.fr/wiki/FdNetDevice
- Reviews are wanted for the fd-net-device module !!



Python script – hybrid experiment I

- Using FdNetDevice example

```
vim nepi/examples/fd_cross_testbed_experiment.py
```

- Create ns-3 backend

```
ns3_provider = FactoriesProvider("ns3")
ns3_desc = exp_desc.add_testbed_description(ns3_provider)
ns3_desc.set_attribute_value("SimulatorImplementationType",
"ns3::RealtimeSimulatorImpl")
ns3_desc.set_attribute_value("ChecksumEnabled", True)
```

- Create netns backend

```
netns_provider = FactoriesProvider("netns")
netns_desc = exp_desc.add_testbed_description(netns_provide
```

Python script – hybrid experiment II

- Create a ns-3 Node box and its protocol stack

```
node = ns3_desc.create("ns3::Node")
ipv4 = ns3_desc.create("ns3::Ipv4L3Protocol")
arp  = ns3_desc.create("ns3::ArpL3Protocol")
icmp = ns3_desc.create("ns3::Icmpv4L4Protocol")
udp = ns3_desc.create("ns3::UdpL4Protocol")
node.connector("protos").connect(ipv4.connector("node"))
node.connector("protos").connect(arp.connector("node"))
node.connector("protos").connect(icmp.connector("node"))
node.connector("protos").connect(udp.connector("node"))
```

- Create a FdNetDevice box

```
fddev = ns3_desc.create("ns3::FdNetDevice")
node.connector("devs").connect(fddev.connector("node"))
ip = fddev.add_address()
ip.set_attribute_value("Address", "10.0.1.1")
```

Python script – hybrid experiment III

- Create a netns Node box

```
netns_node = netns_desc.create("Node")
```

- Create a TAP interface box

```
tap = netns_desc.create("TapNodeInterface")
tap.set_attribute_value("up", True)
netns_node.connector("devs").connect(tap.connector("node"))
ip = tap.add_address()
ip.set_attribute_value("Address", "10.0.1.2")
```

- Connect the ns-3 FdNetDevice with the netns TAP

```
fddev.connector("->fd").connect(tap.connector("fd->"))
```

Supported backends

Backends

- Currently supports 4 backends

NETNS

PlanetLab

- Worldwide distributed network, composed of thousands of nodes interconnected through the Internet
  - Nodes are shared by multiple experiments
  - PlanetLab Central: http://www.planet-lab.org
  - PlanetLab Europe: http://www.planet-lab.eu

Netns

- Light-weight virtual machine to emulate Ethernet networks
  - Run arbitrary applications inside the virtual machines
  - Uses the Linux host real network stack
  - Uses LXC Linux Containers technology (netns)
  - Uses link emulation based on packet scheduling (netem)
  - More info: http://nepi.inria.fr/wiki/netns

OMF

- A Control and Management Framework for Networking Testbed
  - Originally designed for Wireless testbeds
  - Many Wireless deployments open to researchers (NICTA, NITOS, w-Ilab.t,..)
  - More info: http://mytestbed.net/projects/omf
  - Support in NEPI is an ongoing effort

Related work

Related work I

- **SAFE** - Simulation Automation Framework for Experiments
  - http://redmine.eg.bucknell.edu/perrone/projects/framework
  - Manages multiple independent replications of ns-3
  - NEPI is not a ns-3 specific controller
- **CORE** – Common Open Research Emulator
  - http://cs.itd.nrl.navy.mil/work/core/
  - Mixes container based emulation with ns-3 models
  - NEPI aims at mixing any type of resources
- **EMULAB**
  - http://www.emulab.net/
  - Supports emulation and live experimentation on Emulab facility
  - Uses NS format to describe network topologies (same experiment can be simulated with one description)
  - NEPI aims to be independent from a particular facility

Related work II

- **OMF** - cOntrol and Management Framework
  - http://mytestbed.net/projects/omf
  - Controls resources running OMF management software
  - NEPI aims at managing resources without having to modify them
- **TEAGLE**
  - http://trac.panlab.net/trac/wiki
  - Controls Panlab federated resources through the Panlab Teagle portal
  - NEPI aims at being extensible by any user to support arbitrary resources (there is no central coordination or administration instance)

Related work III

- **ProtoGENI**
  - http://www.protogeni.net/
  - Supports resource provisioning through SFA but does not support resource control
  - NEPI aims at supporting both provisioning and control
- **PLUSH & NEBULA**
  - http://plush.cs.williams.edu/nebula/
  - Supports exp life-cycle control for PlanetLab resources
  - NEPI aims to be independent from a particular facility

Related work IV

- NEPI attempts to be a general solution to provide life-cycle control support for non specific platform resources
- Other similar tools are different in that they:
  - Target specific facility resources (e.g. Emulab, SAFE, Plush)
  - Require modifying resources by pre-running specific code (e.g. OMF RC)
  - Resolve only one part of experiment life-cycle (e.g. ProtoGeni)

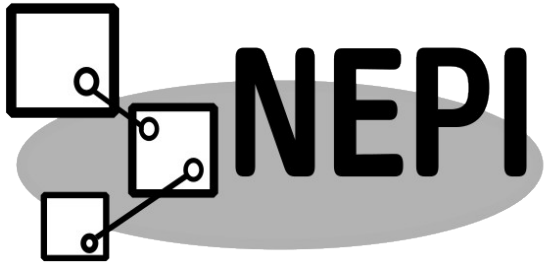
Future steps

Future steps

- New improved version of NEPI
  - Replace TestbedControllers by ResourceControllers
  - Support description of resource run time behavior (e.g. start app1 after app2)
  - Support “high-level” experiment description
  - Support running “a same” experiment on different platforms
- Implement new testbed federation architecture (Openlab and Fed4FIRE initiatives)
  - SFA (provisioning) + FRCP (control)
  - Testbeds implementing SFA + FRCP will be supported “out of the box” by NEPI
  - Support simulation as a resource through FRCP using ns-3 simulator

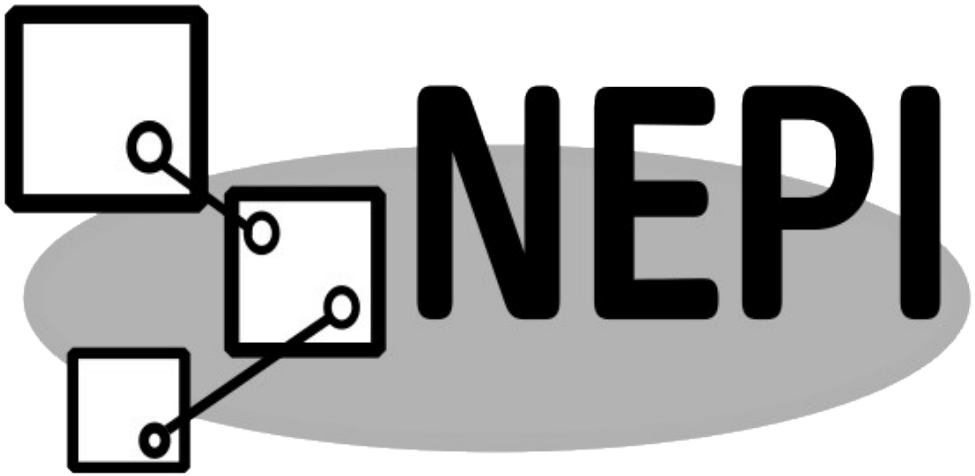
People

- Lucia Guevgeozian
- Julien Tribino
- Claudio Freire
- Martin Ferrari
- Mathieu Lacage
- Thierry Turletti
- Walid Dabbous



More info

- Visit NEPI wiki page for more information and examples **http://nepi.inria.fr**
- Tutorials and source code available!



Thank you

http://nepi.inria.fr
alina.quereilhac@inria.fr

Questions?

http://nepi.inria.fr
alina.quereilhac@inria.fr

Extending NEPI

Adding a new backend

- NEPI was designed to be extended for arbitrary environments
- Steps to create a new backend
  1. Add a new directory under src/nepi/testbeds/ (e.g. src/nepi/testbeds/omf)
  2. Add a metadata.py file and define all the boxes, connector and attributes for the boxes
  3. Implement the functions to be invoked on each type of box upon creation, connection, start, stop
  4. Add a execute.py file and extend the TestbedController class, adding environment specific behavior

Adding new ns-3 models

- Build the Python bindings for the new model
- Add an import to the new module to

  src/nepi/testbeds/ns3/ns3_bindings_import.py
- Add metadata for the new model
  1. Add new attributes to
     src/nepi/testbeds/ns3/attributes_metadata.py
  2. Add new connectors to
     src/nepi/testbeds/ns3/connectors_metadata.py
  3. Add new traces to
     src/nepi/testbeds/ns3/traces_metadata.py
  4. Add new box types to
     src/nepi/testbeds/ns3/factories_metadata.py