Automating ns-3 Experimentation in Multi-Host Scenarios

Alina Quereilhac
Damien Saucez
Thierry Turletti
Walid Dabbous
ns-3 features for advanced simulations

- ns-3 is a modular discrete-event network simulator that provides
  - application and protocol emulation with DCE,
  - special devices (e.g., FD and Tap NetDevice),
  - real-time scheduler,
  - interactive mode.
ns-3 features for advanced simulations

- ns-3 is a modular discrete-event network simulator that provides

Why is that interesting?

- real-time scheduler,
- interactive mode.
ns-3 is modular

Host

ns-3 instance

Node 1

Applications	 Simu

Protocols	 Simu

Devices	 Simu

Node 2

Applications	 Simu

Protocols	 Simu

Devices	 Simu

Link	 Simu
ns-3 is more than a simulator

- Parallelisation
  - run independent simulations.

- Hybrid emulation
  - connect simulations with a real system.

- Distributed simulations
  - span the simulation over multiple hosts.
ns-3 is more than a simulator

- Parallelisation
- run independent simulations.

Hybrid emulation

This is fastidious
(e.g., configuration, synchronisation)

- Distributed simulations
- span the simulation over multiple hosts.
NEPI to make it easy

- NEPI, Network Experiment Programming Interface, is a framework to manage network experiments that abstracts components behind a common interface: the resource.

- Runs locally, no need to modify the experiment facility e.g., ns-3, PlanetLab.
Everything is a resource

- The user interacts with the Experiment Controller (EC), which controls the resources.

- Every resource implements the same interface

  - e.g., deploy, start, stop.
An experiment is a graph of interconnected resources.

Each resource has 3 set of properties:

- attributes (e.g., configuration),
- traces (e.g., stderr, stdout),
- states (i.e., STARTED, STOPPED, FAILED).
Ping example

```python
from nepi.execution.ec import ExperimentController
```
Ping example

```
from nepi.execution.ec import ExperimentController

ec = ExperimentController(exp_id="ping")
```
from nepi.execution.ec import ExperimentController

ec = ExperimentController(exp_id="ping")

node = ec.register_resource("linux::Node")

ec.set(node, "hostname", "my-hostname")

ec.set(node, "username", "my-user")

ec.set(node, "identity", "ssh-key-file")
from nepi.execution.ec import ExperimentController

e = ExperimentController(exp_id="ping")

node = e.register_resource("linux::Node")

e.set(node, "hostname", "my-hostname")

e.set(node, "username", "my-user")

e.set(node, "identity", "ssh-key-file")

app = e.register_resource("linux::Application")

e.set(app, "command", "ping -c3 192.168.0.1")
Ping example

```python
from nepi.execution.ec import ExperimentController

ec = ExperimentController(exp_id="ping")
	node = ec.register_resource("linux::Node")
	ec.set(node, "hostname", "my-hostname")
	ec.set(node, "username", "my-user")
	ec.set(node, "identity", "ssh-key-file")

app = ec.register_resource("linux::Application")
	ec.set(app, "command", "ping -c3 192.168.0.1")
	ec.register_connection(node, app)
```
from nepi.execution.ec import ExperimentController

ec = ExperimentController(exp_id="ping")

node = ec.register_resource("linux::Node")

ec.set(node, "hostname", "my-hostname")

ec.set(node, "username", "my-user")

ec.set(node, "identity", "ssh-key-file")

app = ec.register_resource("linux::Application")

ec.set(app, "command", "ping -c3 192.168.0.1")

ec.register_connection(node, app)

ec.deploy()

ec.wait_finished(app)
Ping example

```python
from nepi.execution.ec import ExperimentController

ec = ExperimentController(exp_id="ping")

node = ec.register_resource("linux::Node")

ec.set(node, "hostname", "my-hostname")

ec.set(node, "username", "my-user")

ec.set(node, "identity", "ssh-key-file")

app = ec.register_resource("linux::Application")

ec.set(app, "command", "ping -c3 192.168.0.1")

ec.register_connection(node, app)

ec.deploy()

ec.wait_finished(app)

print ec.trace(app, "stdout")
```

from nepi.execution.ec import ExperimentController

c = ExperimentController(exp_id="ping")
	node = c.register_resource("linux::Node")

c.set(node, "hostname", "my-hostname")

c.set(node, "username", "my-user")

c.set(node, "identity", "ssh-key-file")

app = c.register_resource("linux::Application")

c.set(app, "command", "ping -c3 192.168.0.1")

c.register_connection(node, app)

c.deploy()

c.wait_finished(app)

print c.trace(app, "stdout")

c.shutdown()
NEPI for ns-3

- NEPI controls (remote) ns-3 simulations
  - via ns-3 Python bindings
  - and a message passing protocol.
Hands-on*

* with simplified code, see the paper for the exact code
A mobility use case with simulated and real nodes
To do list

- Deploy ns-3 on a PlanetLab host
  - Model the simulated network in ns-3
  - Run a real transmitter application
  - Interconnect the ns-3 instance with the real transmitter application
  - Run the experiment
Deploy ns-3 on PlanetLab
Deploy ns-3 on PlanetLab

ec = ExperimentController(exp_id="hybrid")
Deploy ns-3 on PlanetLab

```python
ec = ExperimentController(exp_id="hybrid")

host = ec.register_resource("planetlab::Node")

ec.set(host, "hostname", hostname)

ec.set(host, "username", username)

ec.set(host, "identity", ssh_key)
```
Deploy ns-3 on PlanetLab

```python
ec = ExperimentController(exp_id="hybrid")

host = ec.register_resource("planetlab::Node")

ec.set(host, "hostname", hostname)

ec.set(host, "username", username)

ec.set(host, "identity", ssh_key)

simu = ec.register_resource("linux::ns3::Simulation")
```
Deploy ns-3 on PlanetLab

```python
ec = ExperimentController(exp_id="hybrid")

host = ec.register_resource("planetlab::Node")

ec.set(host, "hostname", hostname)

ec.set(host, "username", username)

ec.set(host, "identity", ssh_key)

simu = ec.register_resource("linux::ns3::Simulation")

ec.register_connection(simu, host)
```
Model the simulated network in ns-3 - the topology
Model the simulated network in ns-3 - the topology

```python
channel = ec.register_resource("ns3::YansWifiChannel")
```
Model the simulated network in ns-3 - the topology

```
channel = ec.register_resource("ns3::YansWifiChannel")

ap = add_ns3_node(ec, simu, agent_ip, prefixlen,
                   channel, ap_mode=True)
```
Model the simulated network in ns-3 - the topology

```python
channel = ec.register_resource("ns3::YansWifiChannel")

ap = add_ns3_node(ec, simu, agent_ip, prefixlen,
                  channel, ap_mode=True)

agent = add_dce_agent(ec, ap)
```
Model the simulated network in ns-3 - the topology

```python
channel = ec.register_resource("ns3::YansWifiChannel")

ap = add_ns3_node(ec, simu, agent_ip, prefixlen,
                   channel, ap_mode=True)

agent = add_dce_agent(ec, ap)

for ip in ips:
    sensor = add_ns3_node(ec, simu, ip, prefixlen,
                          channel, ap_mode=False)

transmitter = add_dce_transmitter(ec, sensor, agent_ip)

add_ns3_route(ec, sensor, network="0.0.0.0/0", nexthop=agent_ip)
```
Model the simulated network in ns-3 - the nodes

```python
def add_ns3_node(ec, simu, ip, prefixlen, channel, ap_mode=False):
```
Model the simulated network in ns-3 - the nodes

def add_ns3_node(ec, simu, ip, prefixlen, channel, ap_mode=False):

    ns3_node = ec.register_resource("ns3::Node")

    ec.set(ns3_node, "enableStack", True)
Model the simulated network in ns-3 - the nodes

```python
def add_ns3_node(ec, simu, ip, prefixlen, channel, ap_mode=False):
    ns3_node = ec.register_resource("ns3::Node")
    ec.set(ns3_node, "enableStack", True)
    ec.register_connection(ns3_node, simu)
```

---

15
Model the simulated network in ns-3 - the nodes

```python
def add_ns3_node(ec, simu, ip, prefixlen, channel, ap_mode=False):
    ns3_node = ec.register_resource("ns3::Node")
    ec.set(ns3_node, "enableStack", True)
    ec.register_connection(ns3_node, simu)
    dev, phy = add_ns3_wifi_device(ec, ns3_node, ip, prefixlen, ap_mode)
    ec.register_connection(channel, phy)
```
def add_ns3_node(ec, simu, ip, prefixlen, channel, ap_mode=False):

    ns3_node = ec.register_resource("ns3::Node")

    ec.set(ns3_node, "enableStack", True)

    ec.register_connection(ns3_node, simu)

    dev, phy = add_ns3_wifi_device(ec, ns3_node, ip, prefixlen, ap_mode)

    ec.register_connection(channel, phy)

    if not ap_mode:
        add_ns3_random_mobility(ec, ns3_node)

    return ns3_node
Model the simulated network in ns-3 - the applications

```python
def add_dce_transmitter(ec, ns3_node, target):
```
Model the simulated network in ns-3 - the applications

```python
def add_dce_transmitter(ec, ns3_node, target):
    transmitter = ec.register_resource("linux::ns3::dce::Application")
```
def add_dce_transmitter(ec, ns3_node, target):

    transmitter = ec.register_resource("linux::ns3::dce::Application")

    ec.set(transmitter, "sources", "code/transmitter.c")

    ec.set(transmitter, "build", "gcc -fPIC -pie 
            -rdynamic ${SRC}/transmitter.c -o ${BIN_DCE}/transmitter")

    ec.set(transmitter, "binary", "transmitter")

    ec.set(transmitter, "arguments", target)
Model the simulated network in ns-3 - the applications

```python
def add_dce_transmitter(ec, ns3_node, target):

    transmitter = ec.register_resource("linux::ns3::dce::Application")

    ec.set(transmitter, "sources", "code/transmitter.c")

    ec.set(transmitter, "build", "gcc -fPIC -pie
    -rdynamic ${SRC}/transmitter.c -o ${BIN_DCE}/transmitter")

    ec.set(transmitter, "binary", "transmitter")

    ec.set(transmitter, "arguments", target)

    ec.register_connection(transmitter, ns3_node)

    return transmitter
```
Interconnect the ns-3 instance with the real transmitter application

- Attach a **File Descriptor NetDevice** to the ns-3 node constituting the access point (ns3::FdNetDevice).

- Create a **TAP device** on the PlanetLab host (planetlab::Tap).

- Connect the “real” TAP to the File Descriptor NetDevice (planetlab::ns3::TunTapFdLink).

- **Add routes**
  - to the simulated network via the access point (planetlab::Vroute),
  - to the real network via the TAP (ns3::Route).
Run the experiment

- ec.deploy()
Conclusion

- ns-3 provides all the building blocks to perform
  - distributed simulations
  - hybrid experiments
- but is fastidious to use as-is.
- NEPI hides the complexity of hybridation and distribution to automate ns-3 experiments.
Automating ns-3 Experimentation in Multi-Host Scenarios

Alina Quereilhac
Damien Saucez
Thierry Turletti
Walid Dabbous