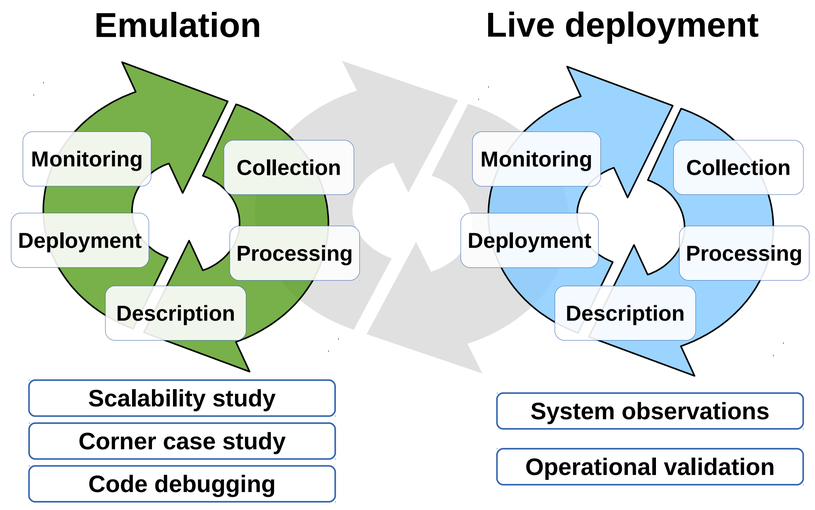
Demonstrating a Unified ICN Development and Evaluation Framework

Alina Quereilhac*, Damien Saucez*, Priya Mahadevan†, Thierry Turletti*, Walid Dabbous*
*INRIA Sophia Antipolis, France - †PARC, Palo Alto, USA

An Iterative Methodology for ICN Software Development

- ICN technologies are meant to be **deployed in live production environments**
- Building **production quality software** requires the use of adequate development and evaluation environments, supporting both controlled and realistic evaluation scenarios
- **Emulation and live deployment** can be combined in a same iterative development and evaluation process, to develop production quality ICN software

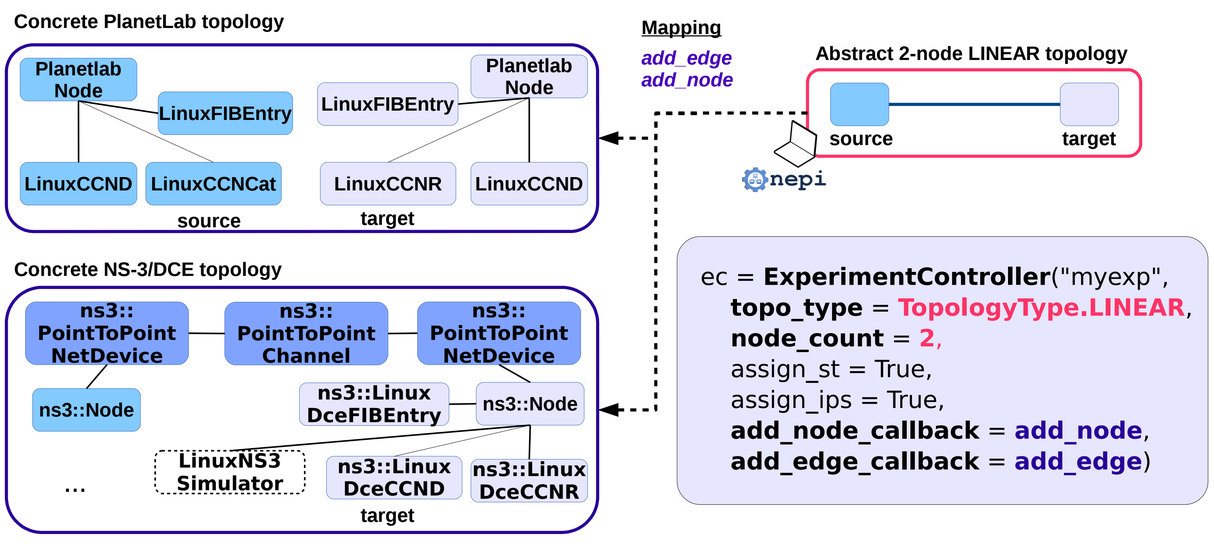
- **Emulation**
  - Controlled evaluation scenarios
  - Emulate larger scale
  - Emulate corner cases (e.g., attacks)
  - Reproduce error conditions
- **Live Deployment**
  - Realistic evaluation scenarios
  - Capture realistic traffic characteristics
  - Validate production software



ICN Development and Evaluation Framework

a) Experiment Description

- An **abstract topology** is a high level representation of the "shape" of a network experiment
- A **concrete topology** (or executable experiment description) includes all necessary information to instantiate the experiment (e.g., what concrete **resources** will be used, how to configure them, etc)
- The **Experiment Controller** entity represents an executable experiment and can **map** an abstract topology to a **concrete Emulated or Live topology**




```
ec = ExperimentController("myexp",
    topo_type = TopologyType.LINEAR,
    node_count = 2,
    assign_st = True,
    assign_ips = True,
    add_node_callback = add_node,
    add_edge_callback = add_edge)
```

b) Experiment Deployment

- The **Experiment Runner** automates replication (re-execution) of a same experiment until a **metric** defined by the user converges according to a **law**
- The **compute_metric_callback** argument is a user defined function that is invoked after each experiment replication to compute the experiment metric
- The **evaluate_convergence_callback** argument is a user defined function that, based on the experiment metrics, decides whether the experimentation has been repeated enough times to be statistically representative

```
runner = ExperimentRunner()

runs = runner.run(ec,
       min_runs = 10,
       max_runs = 100,
       compute_metric_callback = avg_interest_rtt,
       evaluate_convergence_callback = normal_law,
       wait_guids = wait_guids(ec))
```

```
def normal_law(ec, run, sample):
    x = numpy.array(sample)
    n = len(sample)
    std = x.std()
    se = std / math.sqrt(n)
    m = x.mean()
    se95 = se * 2

    return m * 0.05 >= se95
```

```
def avg_interest_rtt(ec, run):

    # Parse downloaded CCND logs
    content_names = ccn_parser.process_content_history_logs(
        ec.run_dir, ec.netgraph.topology)

    # statistics on RTT
    rtts = [content_names[content_name]["rtt"] \
           for content_name in content_names.keys()]

    # sample mean and standard deviation
    sample = numpy.array(rtts)
    n, min_max, mean, var, skew, kurt = stats.describe(sample)
    std = math.sqrt(var)
    ci = stats.t.interval(0.95, n-1, loc = mean,
           scale = std/math.sqrt(n))

    # store values in global list
    global metrics
    metrics.append((mean, ci[0], ci[1]))

    return mean
```

The Framework

NEPI

NEPI, is a framework to automate execution of network experiments on various platforms

nepi

NS-3/DCE

DCE is an application layer emulator for the NS-3 simulator. It supports execution of unmodified application binaries in simulated networks

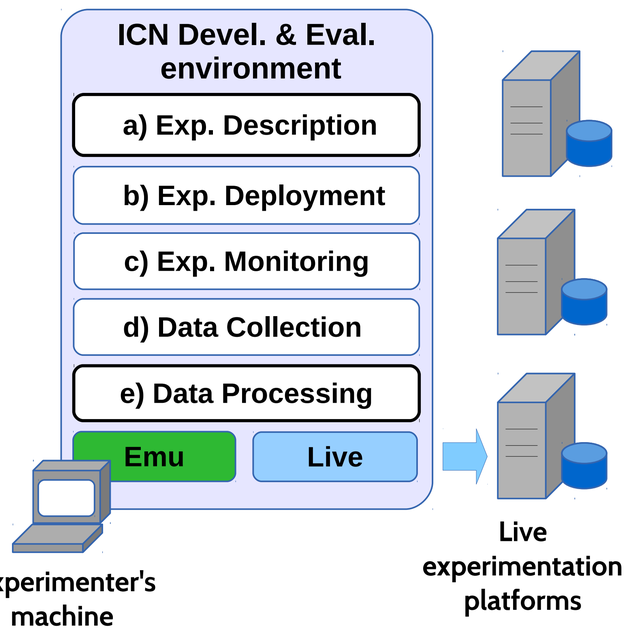
ns-3

PlanetLab

PlanetLab is a distributed testbed with thousands of hosts interconnected through the Internet

PLANETLAB

- Extensions to NEPI
  - We added support for DCE emulation
  - We integrated CCNx applications
- Experiment life cycle supported in 5 steps
- Steps b), c) and d) are fully automated
- Same ICN binaries are used for emulation and live deployment



c) Experiment Monitoring

- Errors on critical resources during deployment will cause controlled interruptions of the experiment
- Errors are notified on standard output
- The state of a resource can be queried at any time during experiment execution

d) Data Collection

- Results can be scheduled for automatic collection at the end of the experiment using a **Collector resource** (see log excerpt below)
- Results can be downloaded at any time using the *trace* primitive

```
[...]
2014-09-12 12:45:25,532 Collector INFO  Collector guid 24 - Collecting 'stderr' traces to local directory /tmp/201409121245032..
2014-09-12 12:45:25,556 LinuxNS3Simulation INFO  guid 2 - host localhost - Retrieving 'files-6/var/log/14012/stderr' trace all
2014-09-12 12:45:25,532 Collector INFO  Collector guid 31 - Collecting 'stderr' traces to local directory /tmp/014091212450324..
[...]
```

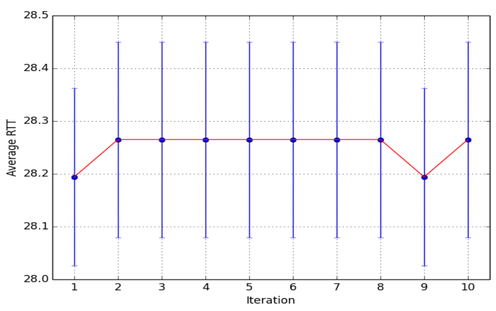
e) Data Processing

```
def post_process(ec, runs):
    global metrics # metrics collected by avg_interest_rtt

    # plot convergence graph
    y = numpy.array([float(m[0]) for m in metrics])
    low = numpy.array([float(m[1]) for m in metrics])
    high = numpy.array([float(m[2]) for m in metrics])
    error = [y - low, high - y]
    x = range(1,runs + 1)

    # plot average RTT and confidence interval for each iteration
    pyplot.errorbar(x, y, yerr = error, fmt='o')
    pyplot.plot(x, y, 'r-')
    pyplot.xlim([0.5, runs + 0.5])
    pyplot.xticks(numpy.arange(1, len(y)+1, 1))
    pyplot.xlabel('Iteration')
    pyplot.ylabel('Average RTT')
    pyplot.grid()
    pyplot.show()
```

- Data processing can be done with standard Python libraries, after the Experiment Runner run() method terminates, using the data collected during the execution of the experiments.



You can learn more about the framework at http://nepi.inria.fr

All sources are GPLv3

informatics mathematics

parc® Palo Alto Research Center

http://nepi.inria.fr