

Experimentation Tools for Networking Research

Mathieu Lacage

INRIA, Planète

Nov 15th 2010

Outline

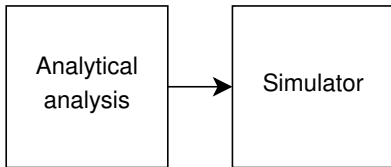
Protocol evaluation

Protocol evaluation

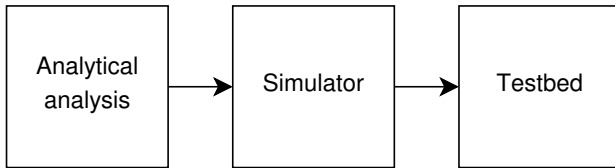


Analytical
analysis

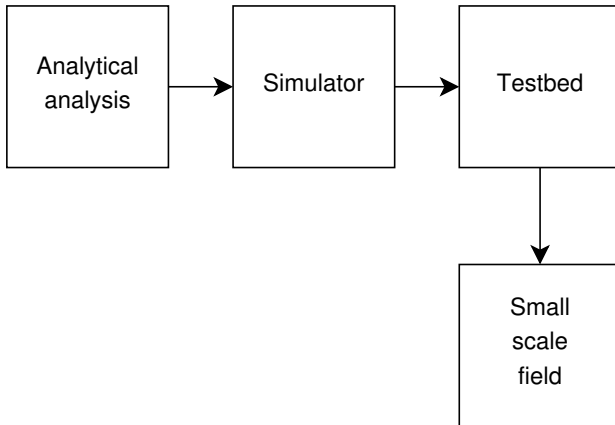
Protocol evaluation



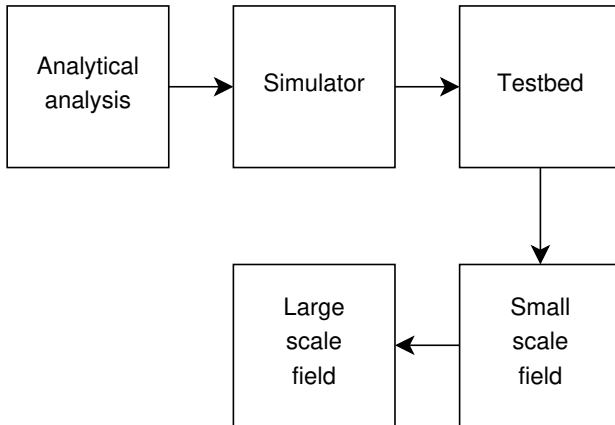
Protocol evaluation



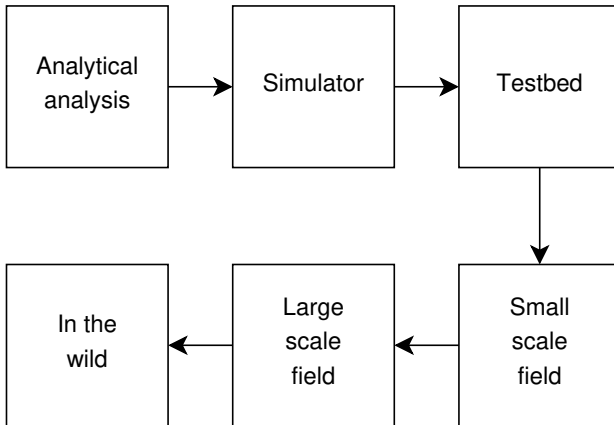
Protocol evaluation



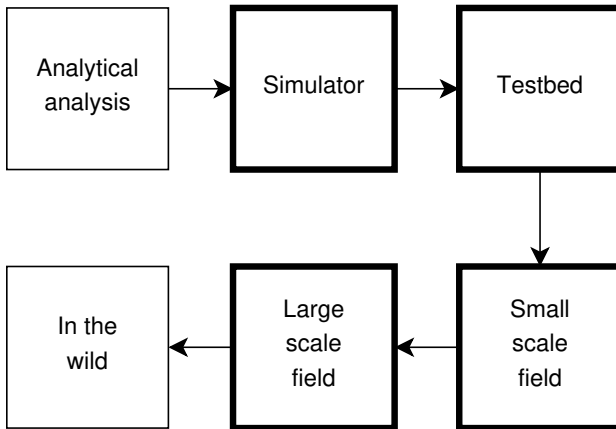
Protocol evaluation



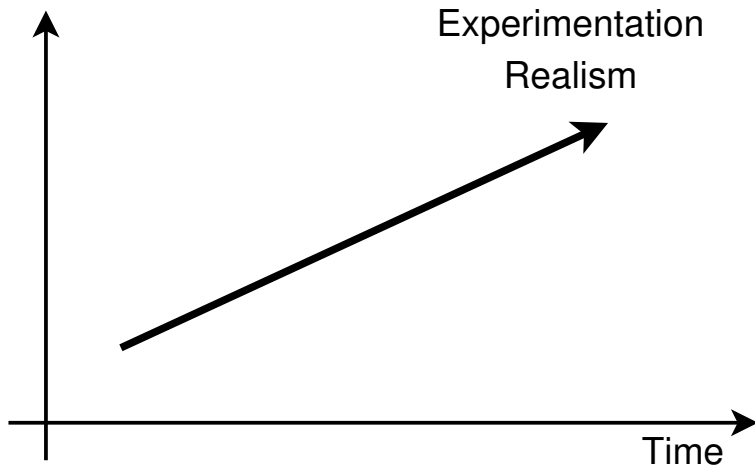
Protocol evaluation



Protocol evaluation



Experimentation context



Why ?

One protocol isolated

One protocol isolated -> Multiple protocol interaction

Why ?

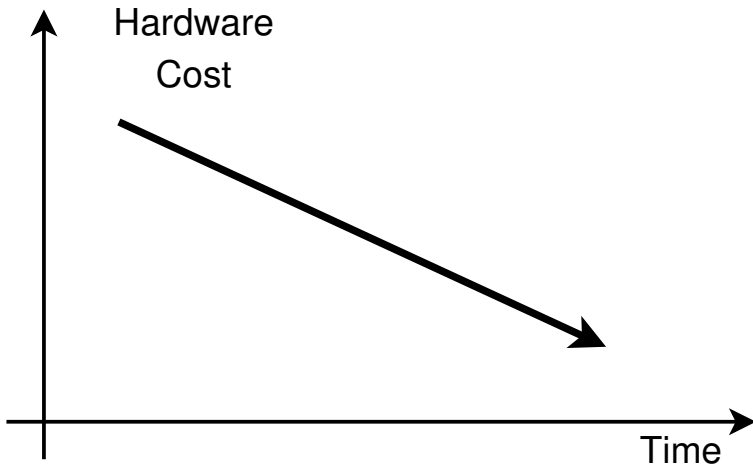
One protocol isolated -> Multiple protocol
interaction

Steady state

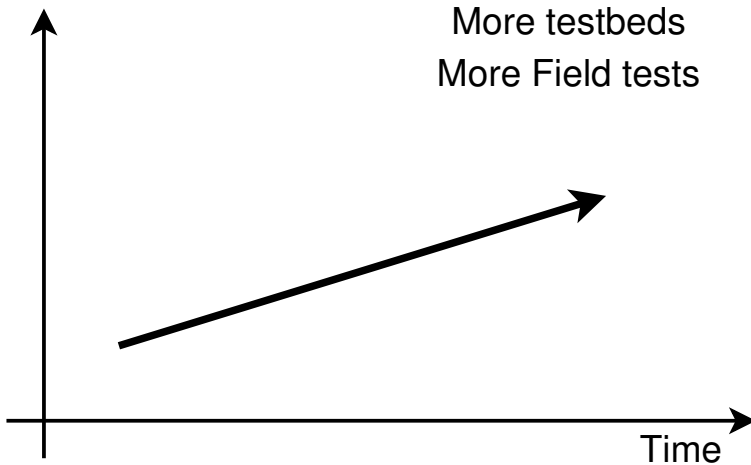
One protocol isolated -> Multiple protocol interaction

Steady state -> Transient behavior

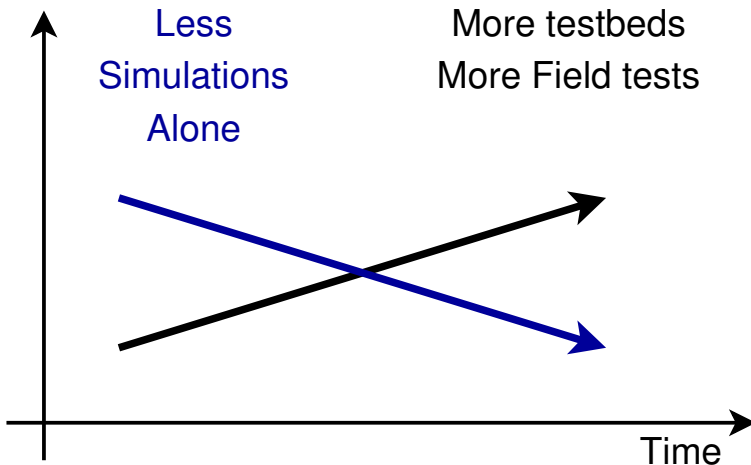
Cheaper hardware



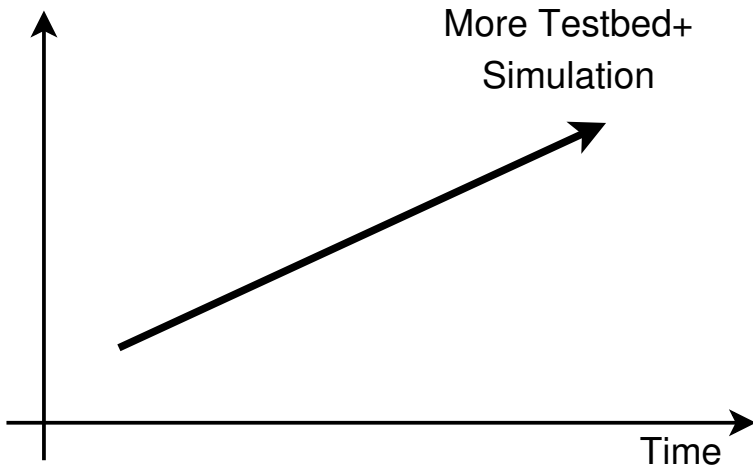
Consequences



Consequences



Consequences



Downsides

Downsides

A lot more work

Downsides

A lot more work

Must master many experimentation
environments

Downsides

A lot more work

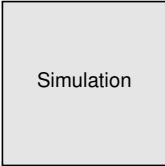
- Must master many experimentation environments

- Must implement protocols twice

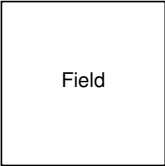
Objectives

Objectives

Smooth transition simulations/testbed/field tests



Simulation

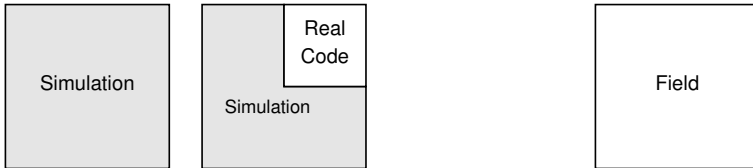


Field

Objectives

Smooth transition simulations/testbed/field tests

Run real protocol implementation in simulation

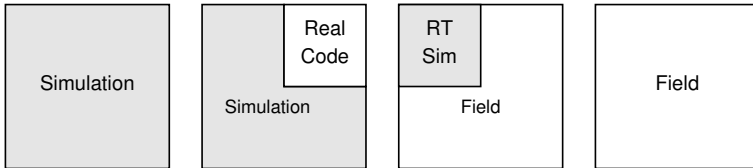


Objectives

Smooth transition simulations/testbed/field tests

Run real protocol implementation in simulation

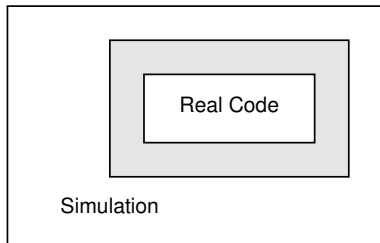
Use simulation as realtime emulator



Run real protocol implementation in simulation

Problems

Run real protocol implementation in simulation
Virtualization of execution environment



Problems

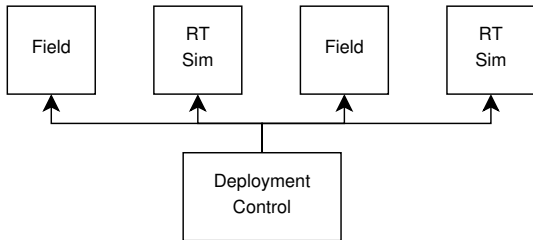
Run real protocol implementation in simulation
Virtualization of execution environment

Use simulation as realtime emulator

Problems

Run real protocol implementation in simulation
Virtualization of execution environment

Use simulation as realtime emulator
Setup, deployment, of emulation platforms



Problems

Run real protocol implementation in simulation
Virtualization of execution environment

Use simulation as realtime emulator
Setup, deployment, of emulation platforms

In both cases

Problems

- Run real protocol implementation in simulation
 - Virtualization of execution environment

- Use simulation as realtime emulator
 - Setup, deployment, of emulation platforms

- In both cases
 - Transparency with other simulation models

Problems

- Run real protocol implementation in simulation
 - Virtualization of execution environment

- Direct Code Execution**

- Use simulation as realtime emulator
 - Setup, deployment, of emulation platforms

- In both cases
 - Transparency with other simulation models

Problems

- Run real protocol implementation in simulation
 - Virtualization of execution environment

- Direct Code Execution**

- Use simulation as realtime emulator
 - Setup, deployment, of emulation platforms

- NEPI**

- In both cases
 - Transparency with other simulation models

Problems

- Run real protocol implementation in simulation
 - Virtualization of execution environment

- Direct Code Execution**

- Use simulation as realtime emulator
 - Setup, deployment, of emulation platforms

- NEPI**

- In both cases
 - Transparency with other simulation models
 - Simulated Packets**

ns-3 Contributions

ns-3 Contributions

ns-3: 300 KLOC

AODV	OLSR		
UDP/IPv4/ARP		TCP	UDP/Ipv6
Csma	PointToPoint	Uan	Bridge
Spectrum	Wifi	Mesh	Wimax
Core	Mobility	MPI	

ns-3 Contributions

ns-3: 300 KLOC

Wrote about 80 KLOC

AODV	OLSR		
UDP/IPv4/ARP		TCP	UDP/Ipv6
Csma	PointToPoint	Uan	Bridge
Spectrum	Wifi	Mesh	Wimax
Core	Mobility	MPI	

Scientific Contributions

Scientific Contributions

Simulated packets

Scientific Contributions

Simulated packets

More CPU efficient than other simulators

Scientific Contributions

Simulated packets

More CPU efficient than other simulators

Automatic conversion simulation/network
format

Scientific Contributions

Simulated packets

More CPU efficient than other simulators

Automatic conversion simulation/network
format

Direct Code Execution

Scientific Contributions

Simulated packets

- More CPU efficient than other simulators
- Automatic conversion simulation/network format

Direct Code Execution

- 10x more CPU efficient than other DCE frameworks

Scientific Contributions

Simulated packets

- More CPU efficient than other simulators
- Automatic conversion simulation/network format

Direct Code Execution

- 10x more CPU efficient than other DCE frameworks
- Large applicability scope

Scientific Contributions

Simulated packets

- More CPU efficient than other simulators
- Automatic conversion simulation/network format

Direct Code Execution

- 10x more CPU efficient than other DCE frameworks
- Large applicability scope

NEPI

Scientific Contributions

Simulated packets

- More CPU efficient than other simulators
- Automatic conversion simulation/network format

Direct Code Execution

- 10x more CPU efficient than other DCE frameworks
- Large applicability scope

NEPI

- Unified experiment description

Scientific Contributions

Simulated packets

- More CPU efficient than other simulators
- Automatic conversion simulation/network format

Direct Code Execution

- 10x more CPU efficient than other DCE frameworks
- Large applicability scope

NEPI

- Unified experiment description
- Automated deployment

Outline

Requirements

Transparent conversion to/from real bytes

CPU and memory efficiency

Fragmentation/Reassembly

Simulation-only data

Pretty printing

Extensibility

Robust Application Programming Interface
(API)

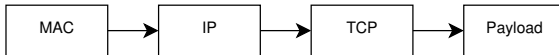
Related Work

Two approaches

Related Work

Two approaches

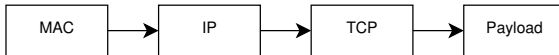
Packet is list of headers: GTNetS, OMNeT++,
SSFNet



Related Work

Two approaches

Packet is list of headers: GTNetS, OMNeT++,
SSFNet



Packet is buffer of bytes: Yans, GloMoSim



Pros and Cons

	List	Buffer
Fragmentation, Reassembly		✓
Conversion real bytes		✓
Simulation-only data	✓	
Pretty printing	✓	
CPU, memory efficiency	✓	

Our solution

Packet as buffer of bytes

Fragmentation, Reassembly

Automatic conversion to/from real bytes

Our solution

Packet as buffer of bytes

- Fragmentation, Reassembly

- Automatic conversion to/from real bytes

Tags

- Simulation-only data

Our solution

Packet as buffer of bytes

- Fragmentation, Reassembly

- Automatic conversion to/from real bytes

Tags

- Simulation-only data

Metadata

- Pretty printing

Our solution

Packet as buffer of bytes

- Fragmentation, Reassembly

- Automatic conversion to/from real bytes

Tags

- Simulation-only data

Metadata

- Pretty printing

Copy On Write (COW)

- CPU, memory efficiency

Performance evaluation

Micro-benchmark scenarios

- Reception

- Forwarding

- Transmission

- Re-transmission

Relative speedup

	ns-3	OMNet++	GTNetS	Yans
Reception	1.00	1.98	2.25	3.07
Transmission	1.00	2.13	1.78	2.16
Forwarding	1.00	2.83	1.92	2.44
Retransmission	1.00	1.92	1.70	4.76

Contributions

Contributions

More CPU efficient than other simulators

Contributions

More CPU efficient than other simulators

Transparent support for real network bytes

Outline

The manual approach

Global variables

```
static int g_var;  
g_var ++;
```

```
static int [100]g_var_array;  
g_var_array[current_id ()] ++;
```

The manual approach

Global variables

```
static int g_var;  
g_var ++;
```

```
static int [100]g_var_array;  
g_var_array[current_id ()] ++;
```

Redirect system calls

```
clock ();
```

```
dce_clock ();
```

The manual approach

Global variables

```
static int g_var;  
g_var ++;
```

```
static int [100]g_var_array;  
g_var_array[current_id ()] ++;
```

Redirect system calls

```
clock ();
```

```
dce_clock ();
```

Re-implement all system calls

```
clock_t dce_clock (void) {  
    return Simulator::Now ().GetMicroSeconds ();  
}
```

The problem

Manual modifications: does not scale

Painful to do once

Impossible to do for software updates

Related work

Related work

Network Simulation Cradle

- Automated source modifications for C code

- Hard to extend to C++

Related work

Network Simulation Cradle

- Automated source modifications for C code

- Hard to extend to C++

Weaves

- Automated textual assembly modifications

- Invalid assumptions about
compiler-generated
code

Related work

Network Simulation Cradle

- Automated source modifications for C code

- Hard to extend to C++

Weaves

- Automated textual assembly modifications

- Invalid assumptions about
compiler-generated
code

COOJA

- Automated memory virtualization

- Slow

Executable and Linkable Format (ELF) loader

Fast

Automated memory virtualization

Automated system call redirection

Executable and Linkable Format (ELF) loader

Fast

Automated memory virtualization

Automated system call redirection

Userspace system calls

Executable and Linkable Format (ELF) loader

Fast

Automated memory virtualization

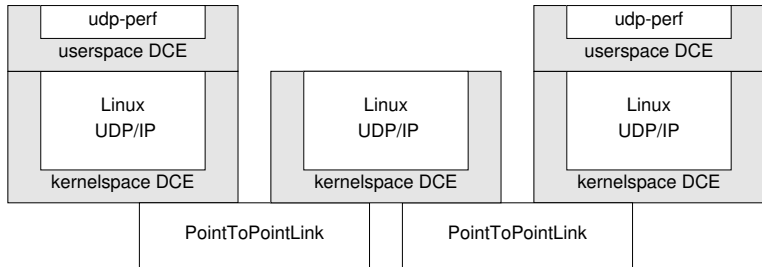
Automated system call redirection

Userspace system calls

Kernelspace system calls

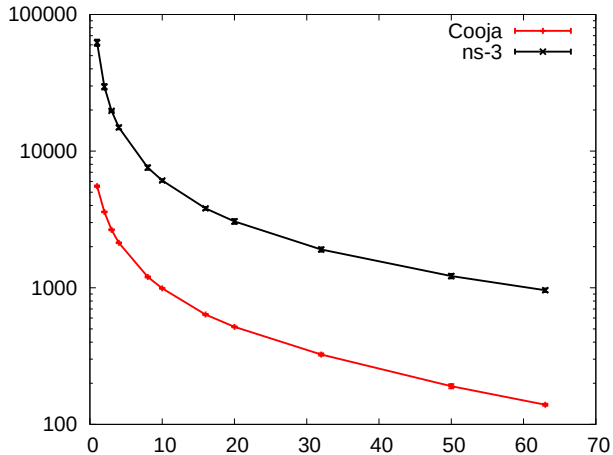
Loader Performance

Scenario:



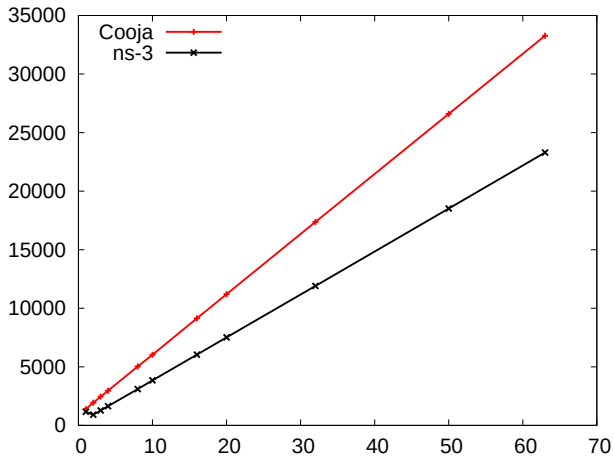
Loader Performance

Packets per
wall clock
second
vs
Number of
nodes



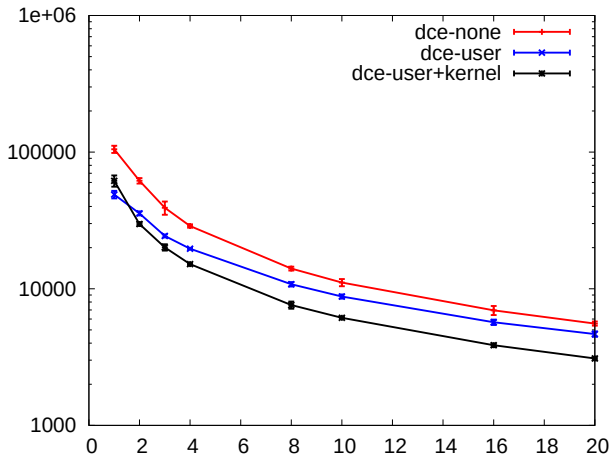
Loader Performance

Memory(bytes)
vs
Number of
nodes



System Performance

Packets per
wall clock
second
vs
Number of
nodes



Tested programs

Binary	Missing functions
/sbin/ip	24
/bin/ping	2
/bin/traceroute	5
/usr/sbin/zebra	42
/usr/sbin/ospfd	43
/usr/sbin/mip6d	38

Untested programs

Binary	Missing functions
ccnd	11
/usr/sbin/httpd	18

Contributions

10x more efficient than existing alternatives

Contributions

10x more efficient than existing alternatives

Larger scope than existing alternatives

Userspace: ping, traceroute, quagga, etc.

Kernelspace: Linux IP, TCP, etc.

Contributions

10x more efficient than existing alternatives

Larger scope than existing alternatives

Userspace: ping, traceroute, quagga, etc.

Kernelspace: Linux IP, TCP, etc.

Potential usecases:

Debugging platform: single debugger controls
all protocol instances

Contributions

10x more efficient than existing alternatives

Larger scope than existing alternatives

Userspace: ping, traceroute, quagga, etc.

Kernelspace: Linux IP, TCP, etc.

Potential usecases:

Debugging platform: single debugger controls
all protocol instances

Development platform

Contributions

10x more efficient than existing alternatives

Larger scope than existing alternatives

Userspace: ping, traceroute, quagga, etc.

Kernelspace: Linux IP, TCP, etc.

Potential usecases:

Debugging platform: single debugger controls
all protocol instances

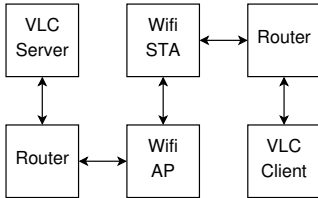
Development platform

Test platform

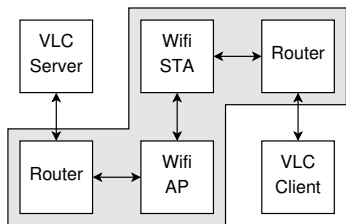
Outline

Objective Scenario

Objective Scenario



Objective Scenario



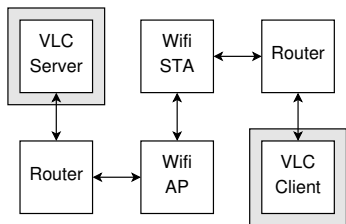
Simulated links/networks

ns-3 simulation models ✓

Realtime scheduler ✓

Tap device ✓

Objective Scenario



Simulated links/networks

ns-3 simulation models ✓

Realtime scheduler ✓

Tap device ✓

Light weight Virtual Machines

Linux Network Namespaces ✓

Easy Deployment

Easy Deployment

Problem

Tap/VM creation and setup

Easy Deployment

Problem

- Tap/VM creation and setup

- Coherent IP address assignment across simulation and VMs

Easy Deployment

Problem

- Tap/VM creation and setup

- Coherent IP address assignment across simulation and VMs

- Coherent IP forwarding tables across simulation and VMs

Easy Deployment

Problem

- Tap/VM creation and setup

- Coherent IP address assignment across simulation and VMs

- Coherent IP forwarding tables across simulation and VMs

Solution

- Automate everything

Easy Deployment

Problem

- Tap/VM creation and setup

- Coherent IP address assignment across simulation and VMs

- Coherent IP forwarding tables across simulation and VMs

Solution

- Automate everything

BUT

- Need global view of experiment topology

Related Work

Related Work

Emulab Tcl: ad hoc, hard to generalize

Related Work

Emulab Tcl: ad hoc, hard to generalize

OMF Ruby: unclear how to extend it to model complex topologies

Related Work

Emulab Tcl: ad hoc, hard to generalize

OMF Ruby: unclear how to extend it to model complex topologies

OMNeT++ NED: hard to ensure correctness

Related Work

Emulab Tcl: ad hoc, hard to generalize

OMF Ruby: unclear how to extend it to model complex topologies

OMNeT++ NED: hard to ensure correctness

SSF DML: hard to ensure correctness

Related Work

Emulab Tcl: ad hoc, hard to generalize

OMF Ruby: unclear how to extend it to model complex topologies

OMNeT++ NED: hard to ensure correctness

SSF DML: hard to ensure correctness

Geni RSPEC: hard to ensure correctness

Related Work

Emulab Tcl: ad hoc, hard to generalize

OMF Ruby: unclear how to extend it to model complex topologies

OMNeT++ NED: hard to ensure correctness

SSF DML: hard to ensure correctness

Geni RSPEC: hard to ensure correctness

Geni Omnispec: hard to ensure correctness

NEPI Object Model

NEPI Object Model

Functional unit / Box

Example:

IP stack

TCP stack

Ethernet card

NEPI Object Model

Functional unit / Box
Attributes

Example:
IP checksum
IP forwarding

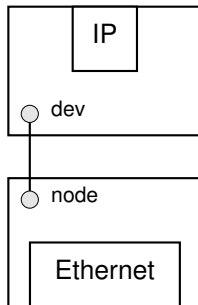
NEPI Object Model

Functional unit / Box
Attributes
Trace sources

Example:
Out packets
In packets

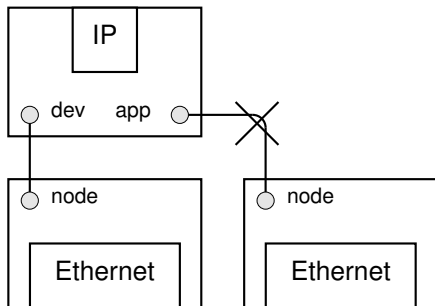
NEPI Object Model

Functional unit / Box
Attributes
Trace sources
Connectors



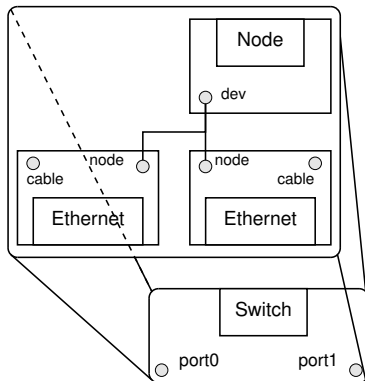
NEPI Object Model

Functional unit / Box
Attributes
Trace sources
Connectors
Connection checking

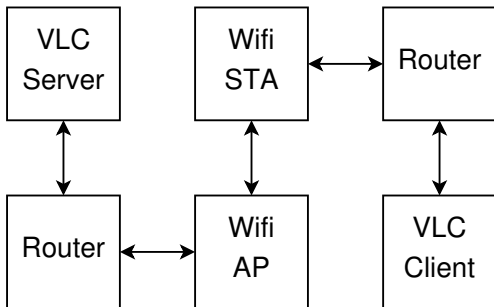


NEPI Object Model

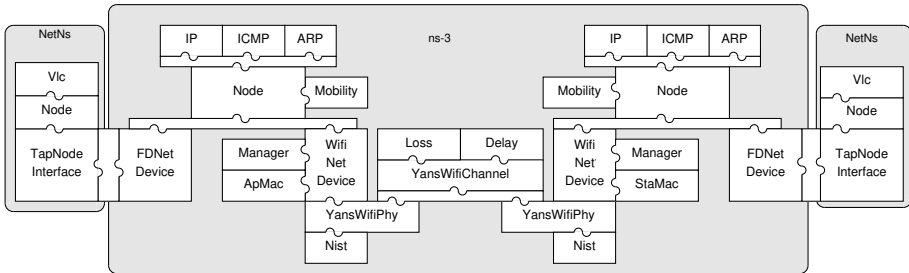
Functional unit / Box
Attributes
Trace sources
Connectors
Connection checking
Hierarchical



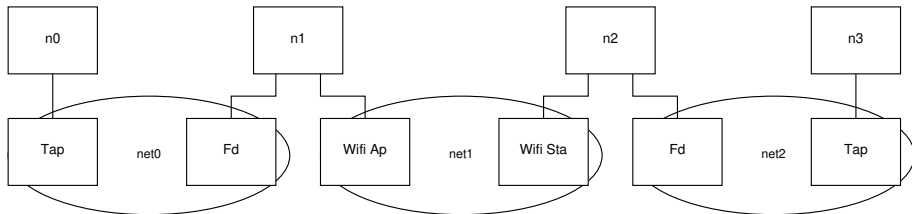
Objective Scenario



NEPI Representation



Global IP topology



Demo screenshot

The screenshot displays a network demonstration environment with several windows:

- client**: A terminal window showing network configuration and statistics for the client machine. It includes details for the eth0 interface (IP: 10.0.0.10, MAC: 3e:dd:08:9e:b5:7d) and the loopback interface (IP: 127.0.0.1, MAC: 00:00:00:00:00:00). It also shows ping statistics for 10.0.0.6.
- server**: A terminal window showing network configuration and statistics for the server machine. It includes details for the eth0 interface (IP: 10.0.0.7, MAC: 3e:dd:08:9e:b5:7d) and the loopback interface (IP: 127.0.0.1, MAC: 00:00:00:00:00:00). It also shows ping statistics for 10.0.0.10.
- NEPI DEMO (as superuser)**: A window displaying a network diagram with two nodes, node0 and node1, connected by a dashed line. The diagram shows the physical layout of the network.
- rtsp://10.0.0.10:5004/test.ts - VLC media player**: A window showing a video stream being played back. The video shows a group of people in a room.
- test.ts - VLC media player**: A window showing a video stream being played back. The video shows a group of people in a room.

The bottom of the screen shows a taskbar with several open applications: Applications, Places, System, client, NEPI DEMO (as super...), xterm, and rtsp://10.0.0.10:5004/t...

Contributions

An integrated experimentation environment

- Unified experiment description

- Entire workflow support

- Automated deployment

- Strong coherency checking

Outline

Papers Published

Papers Published

“Yet another network simulator”, Proceedings
of
the 2006 workshop on ns-2

Papers Published

“Yet another network simulator”, Proceedings of

the 2006 workshop on ns-2

“NEPI: using independent simulators, emulators, and testbeds for easy experimentation”, SIGOPS Operating Systems Review

Papers in progress

“The ns-3 network simulator: experience learned”, Software Practice & Experience

Papers in progress

“The ns-3 network simulator: experience learned”, Software Practice & Experience
“Direct Code Execution”, Networked Systems Design and Implementation

Chair

NSTools 2007

TPC

WNS3 2009, 2010, 2011

Simutools 2008,2009

Simulated Packets

More CPU efficient than other simulators

Simulated Packets

More CPU efficient than other simulators

Transparent support for real network bytes

Direct Code Execution

10x more CPU efficient than other DCE frameworks

Direct Code Execution

10x more CPU efficient than other DCE frameworks

A robust implementation

- Userspace and kernelspace protocols

- C,C++ protocols

- ABI compatibility for userspace protocols

An integrated experimentation environment

- Unified experiment description

- Entire workflow support

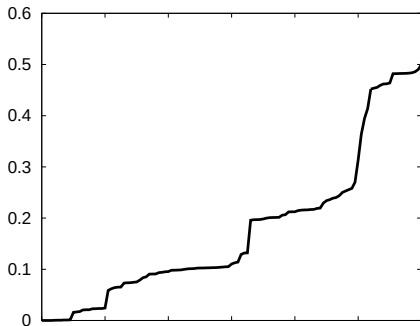
- Automated deployment

- Strong coherency checking

Impact

Instrumental in creating an
Active
Growing
Open source
Simulation community

Instrumental in creating an
Active
Growing
Open source
Simulation community
Contributors > 80



Instrumental in creating an

Active

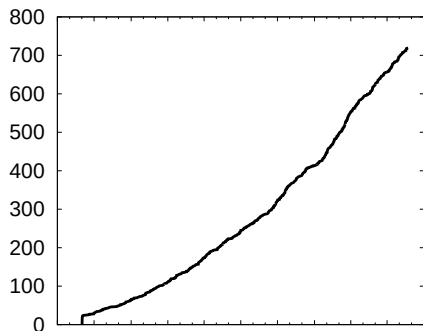
Growing

Open source

Simulation community

Contributors > 80

Hundreds of users



Instrumental in creating an

Active

Growing

Open source

Simulation community

Contributors > 80

Hundreds of users

> 30 papers by users

2 sigcomm'09

1 sigcomm'10

Outline

Tracing with DCE

Tracing with DCE

Need to recompile for tracing changes

Tracing with DCE

Need to recompile for tracing changes
Ideal workflow:

Tracing with DCE

Need to recompile for tracing changes
Ideal workflow:

Trace function *foo* in *bar.cc*

Tracing with DCE

Need to recompile for tracing changes

Ideal workflow:

- Trace function *foo* in *bar.cc*

- Trace line 122 in *bar.cc*

Tracing with DCE

Need to recompile for tracing changes

Ideal workflow:

- Trace function *foo* in *bar.cc*

- Trace line 122 in *bar.cc*

- Trace variable *foo* at line 144 in *bar.cc*

Tracing with DCE

Need to recompile for tracing changes

Ideal workflow:

- Trace function *foo* in *bar.cc*

- Trace line 122 in *bar.cc*

- Trace variable *foo* at line 144 in *bar.cc*

BUT, not Java: no introspection

Dynamic instrumentation

Dynamic instrumentation

Locate code in memory

Dynamic instrumentation

Locate code in memory

Parse dynamic loader data structures✓

Dynamic instrumentation

Locate code in memory

Parse dynamic loader data structures✓

Parse debugging information

Dynamic instrumentation

Locate code in memory

- Parse dynamic loader data structures✓

- Parse debugging information

Insert assembly hooks

Dynamic instrumentation

Locate code in memory

- Parse dynamic loader data structures ✓

- Parse debugging information

Insert assembly hooks

- Simple case (not thread-safe) ✓

Dynamic instrumentation

Locate code in memory

- Parse dynamic loader data structures ✓

- Parse debugging information

Insert assembly hooks

- Simple case (not thread-safe) ✓

- General case harder

Dynamic instrumentation

Locate code in memory

- Parse dynamic loader data structures ✓

- Parse debugging information

Insert assembly hooks

- Simple case (not thread-safe) ✓

- General case harder

Multi-threaded Scheduler

Multi-threaded Scheduler

Conservative algorithm implemented:

Multi-threaded Scheduler

Conservative algorithm implemented:
Fully transparent for users

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

- Thread-safety is easy

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

- Thread-safety is easy

- Efficient thread-safety is *hard*

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

- Thread-safety is easy

- Efficient thread-safety is *hard*

- Can we build more efficient thread-safety ?

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

- Thread-safety is easy

- Efficient thread-safety is *hard*

- Can we build more efficient thread-safety ?

- Can we use optimistic algorithm transparently ?

Multi-threaded Scheduler

Conservative algorithm implemented:

- Fully transparent for users

- Measurable speedup

- Thread-safety is easy

- Efficient thread-safety is *hard*

- Can we build more efficient thread-safety ?

- Can we use optimistic algorithm transparently ?

Transparency is key !

Thank you !

Questions ?