Virtual Machines and ns-3

Tom Henderson and Craig Dowell
University of Washington
Jeff Ahrenholz, Tom Goff, and Gary Pei
The Boeing Company

Supporting organization: **Brian Adamson**
Naval Research Laboratory

Workshop on ns-3
March 2010

Outline

- Goals and related work
- Common Open Research Emulator (CORE)
- Issues with CORE and ns-3
- Pure Linux containers
- Python-based “netns3”
- Next steps

Test and Evaluation Options

Test and evaluation options

Pure simulation	Simulation cradles	Virtual/Physical testbeds	Field experiments	Live networks

Increasing realism

Increasing complexity

Can we develop tools to span this space?

Goals

- Lightweight virtualization of kernel and application processes, interconnected by simulated networks
- Benefits:
  - Implementation realism in controlled topologies or wireless environments
  - Model availability
- Limitations:
  - Not as scalable as pure simulation
  - Runs in real-time
  - Integration of the two environments

ns-3 related work

Test and evaluation options

Pure simulation | **Simulation cradles** | **Virtual/Physical testbeds** | **Field experiments** | **Live networks**

NSC (Jansen)
Protolib (NRL)
ns-3-simu (Lacage)

CORE
NEPI
Testbeds:
- ORBIT
- CMU wireless emulator

Other recent related work

CORE is the Common Open Research Emulator that controls lightweight virtual machines and a network emulation subsystem (more on this later)

NEPI/NEF: Using Independent Simulators, Emulators, and Testbeds for Easy Experimentation

- Lacage, Ferrari, Hansen, Turletti (Roads 2009 workshop)

EMANE is an Extendable Mobile Ad-hoc Network Emulator that allows heterogeneous network emulation using a pluggable MAC and PHY layer architecture.

- http://labs.cengen.com/emane
- being integrated with CORE

Related work (cont.)

Synchronized Network Emulation: Matching prototypes with complex simulations

- Weingartner, Schmidt, Heer, and Wehrle (Hotmetrics 2008)
- Hendrik von Lehn, “A WiFi Emulation Framework for ns-3” (this afternoon)

Protocol platform abstraction libraries

- VIPE (Virtual Platform for Network Experimentation)
  - Landsiedel, Kunz, Gotz, Wehrle (VISA 2009 workshop)
- Protolib prototyping toolkit (from NRL)

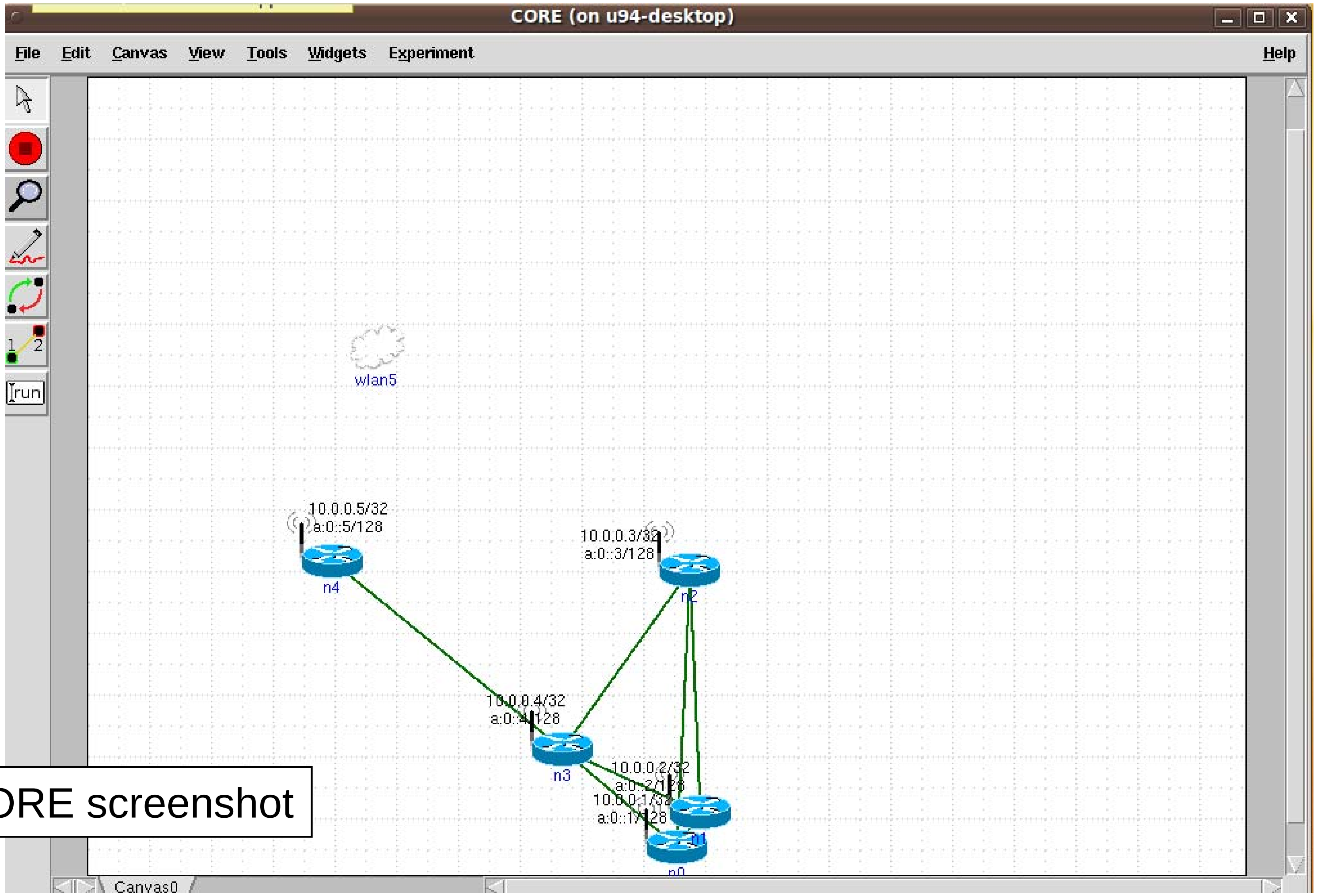
ns-2 emulation (Mahrenholz/Ivanov)

Trellis, a Platform for for Building Fast, Flexible Virtual Networks (ROADS 2008)

RapidMesh (Drexel) has run ns-3 on the Emulab testbed

Alvarez et al, “Limitations of network emulation with single machine and distributed ns-3” (SIMUTools 2010)

Basic CORE demonstration



CORE screenshot

What is CORE (cont.)?

- CORE consists of a (1) GUI, (2) services layer, and an (3) API tying components together.

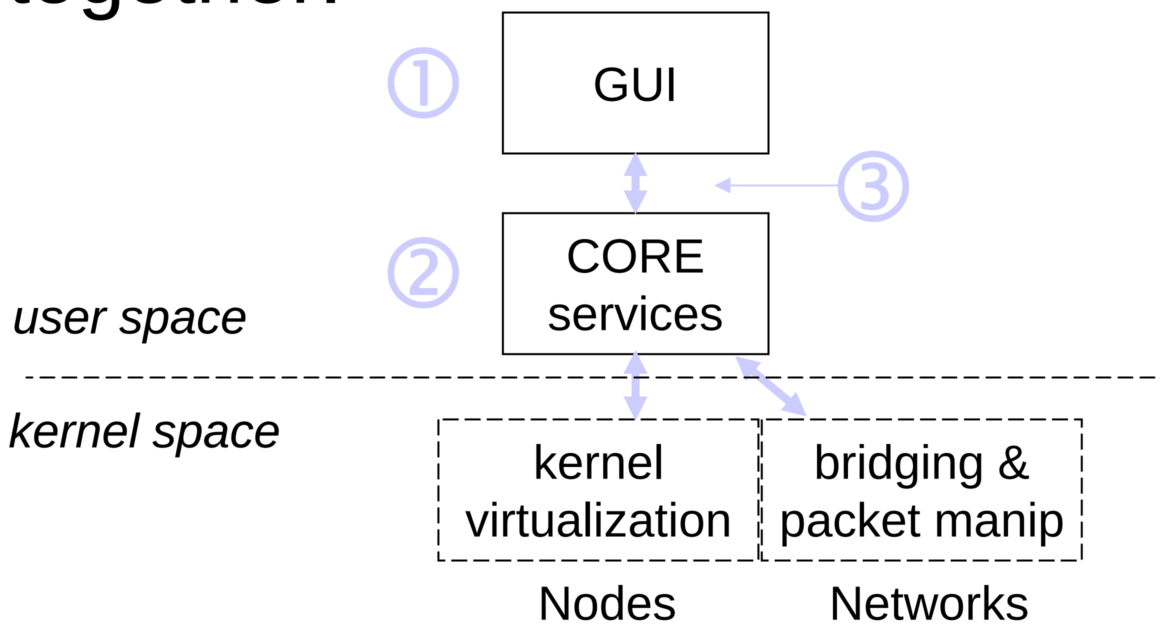


Figure courtesy of Jeff Ahrenholz

Platforms

- Modular architecture allows for interchangeable parts

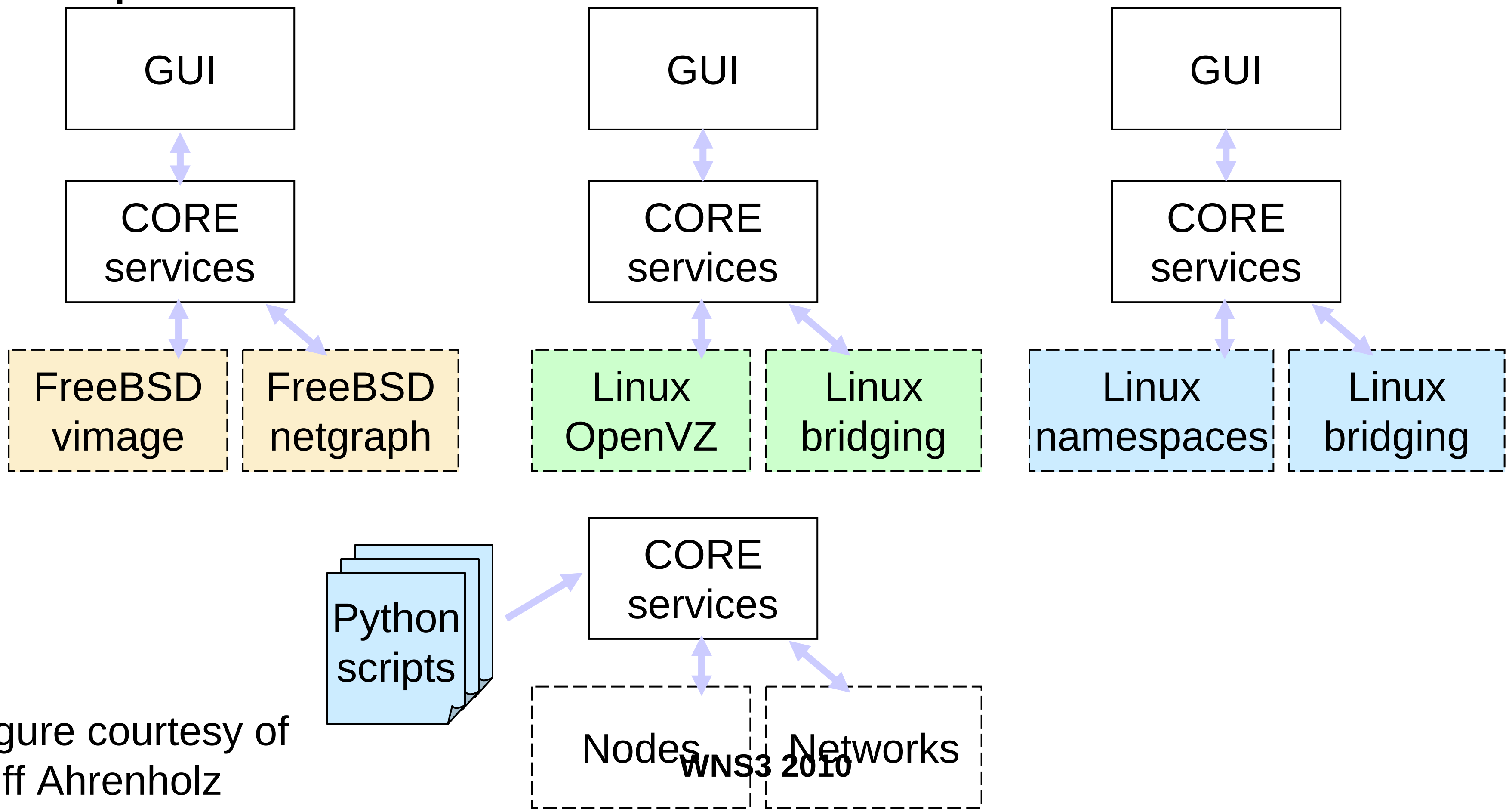


Figure courtesy of Jeff Ahrenholz

What's new?

- Linux Network Namespaces Architecture

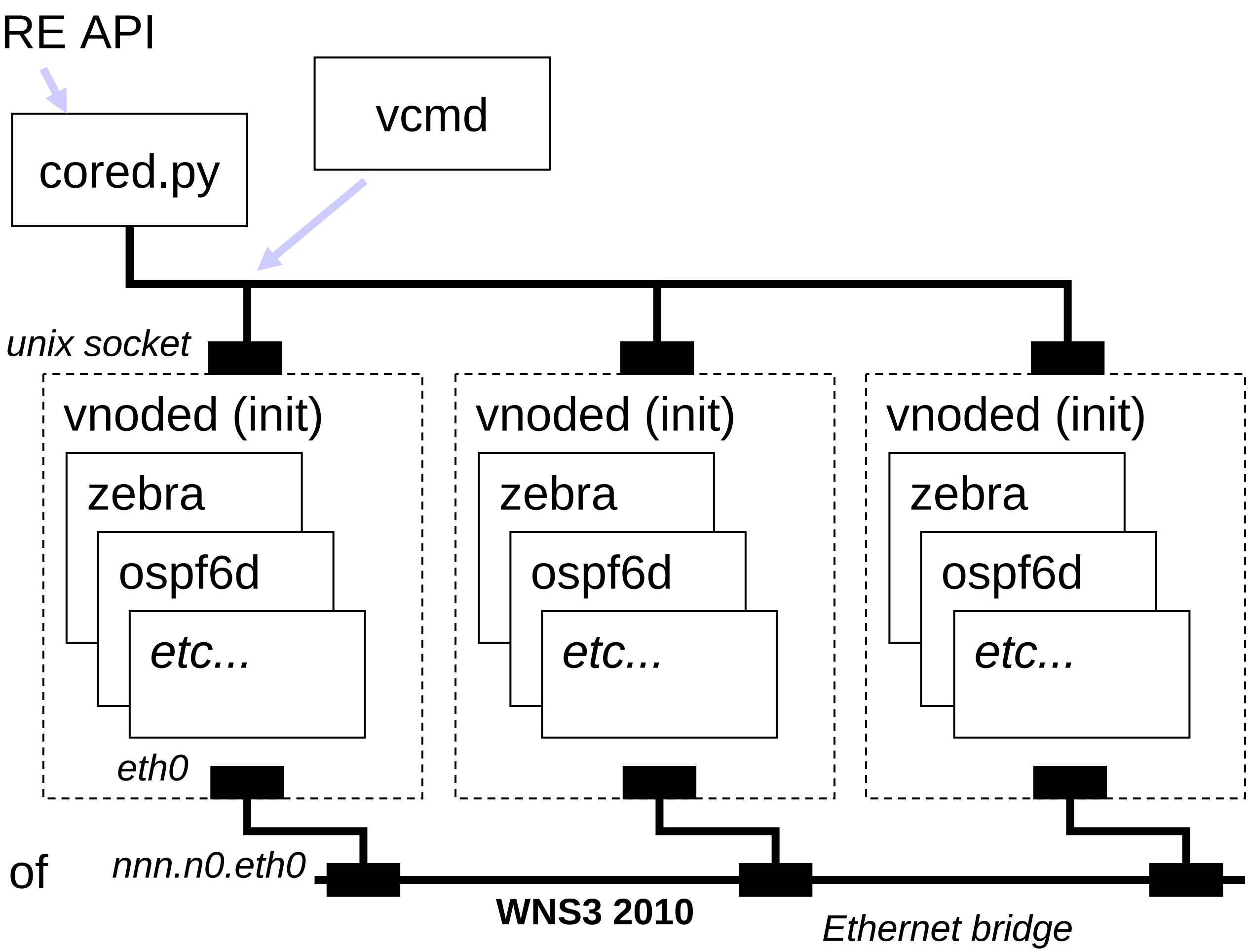


Figure courtesy of
Jeff Ahrenholz

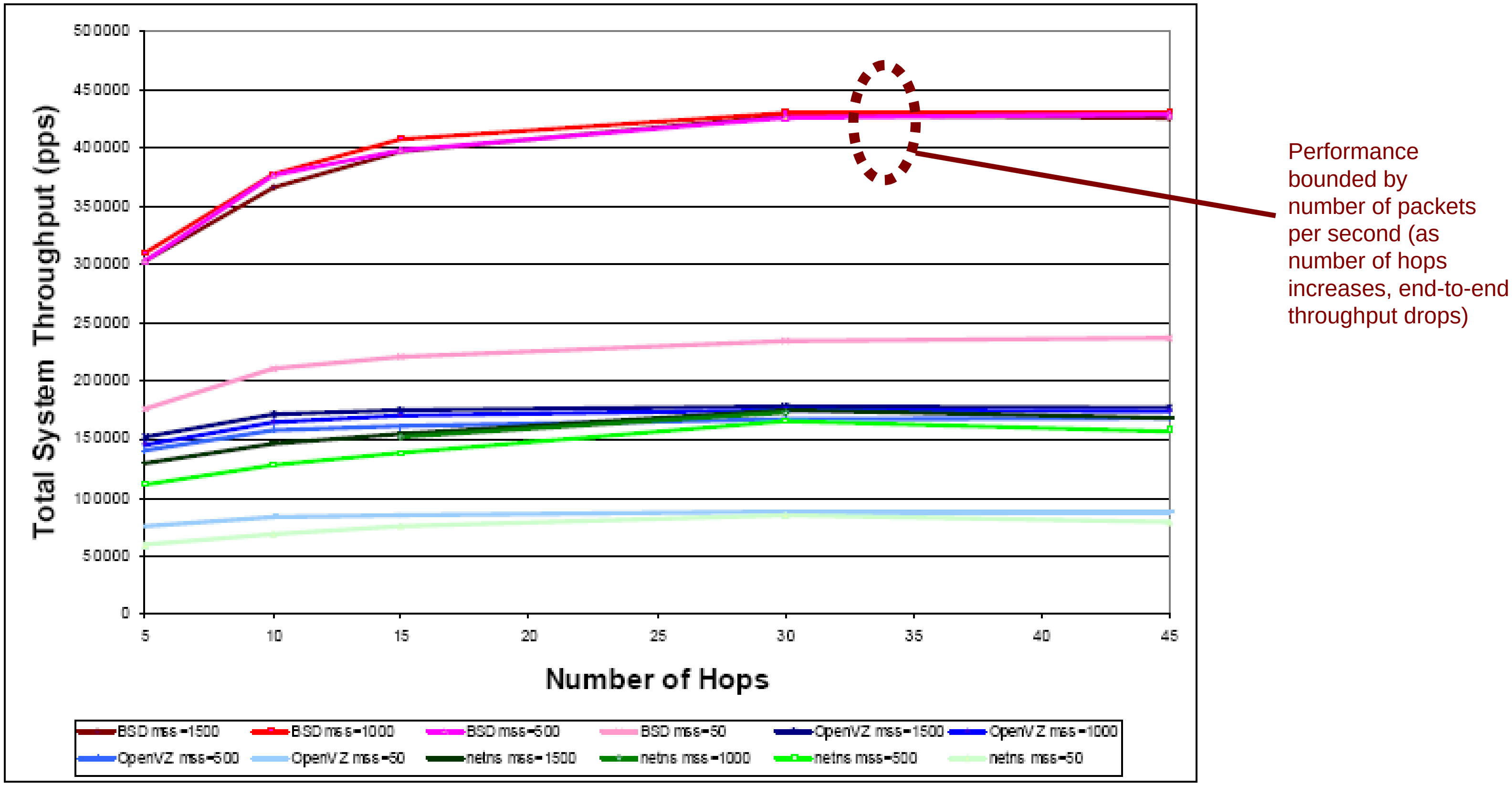
Why integrate netns extensions?

- Performance
  - more lightweight than OpenVZ containers
- Shared filesystem
  - like FreeBSD, vs. core-root with OpenVZ
- Python
  - virtualized elements easily customized with inheritance
  - allows complex scripting of CORE scenarios
  - ns-3 Python bindings for more natural CORE + ns-3
- Mainline kernel vs OpenVZ patch
  - Fedora 12/Ubuntu 9.10 stock kernel support – just install CORE RPM and run
  - stable OpenVZ 2.6.18 kernel is getting old

How Many Nodes?

- It depends.
- Single server
  - typical 3.0 GHz quad Xeon 2.0GB RAM
  - Can instantiate up to 3,600 nodes (with no interfaces)
  - Can run ~100 Quagga routers running OSPFv3-MANET
  - Your mileage may vary: if pushing around lots of data or consuming many cycles, maybe 2-3 nodes?
- Multiple servers
  - Farm of emulation servers using RJ45 node, Span tunnels, etc
  - Consider networking between servers (latency, bandwidth)

How Many Packets?



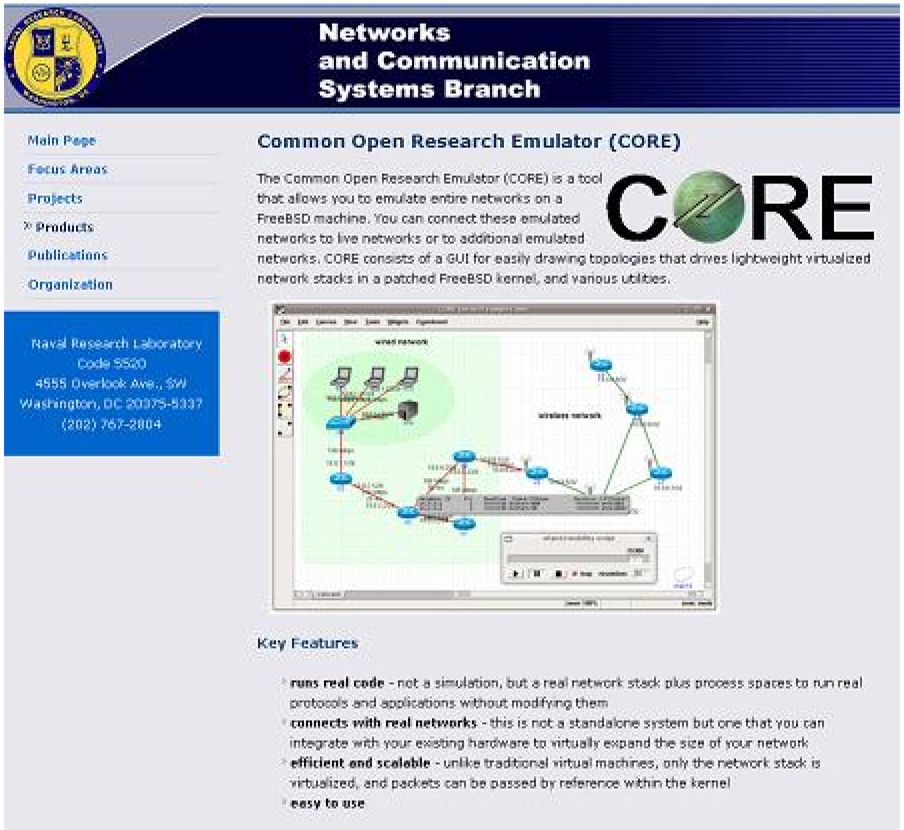
- 430kpps (BSD Netgraph) vs. 170kpps (Linux Netns / OpenVZ bridging)
- Performance relatively insensitive to packet size

More details on experiments

- CORE versions
  - FreeBSD 8.0, Linux OpenVZ 2.6.18, Netns 2.6.31
- Hardware
  - IBM x3550 type 7978, quad-core Xeon E5335 2.0GHz, 2.0GB RAM
- iperf used for end-to-end TCP throughput
  - iperf client run on a separate machine, hooked to CORE via the Gigabit Ethernet on the host
- multi-hop experiments use a daisy chain of routers
  - Total system throughput is defined as the observed iperf throughput * number of hops in the topology (packets per sec)
  - Gives a rough estimate of the number of packets being handled by CORE, per second

CORE is an open source project

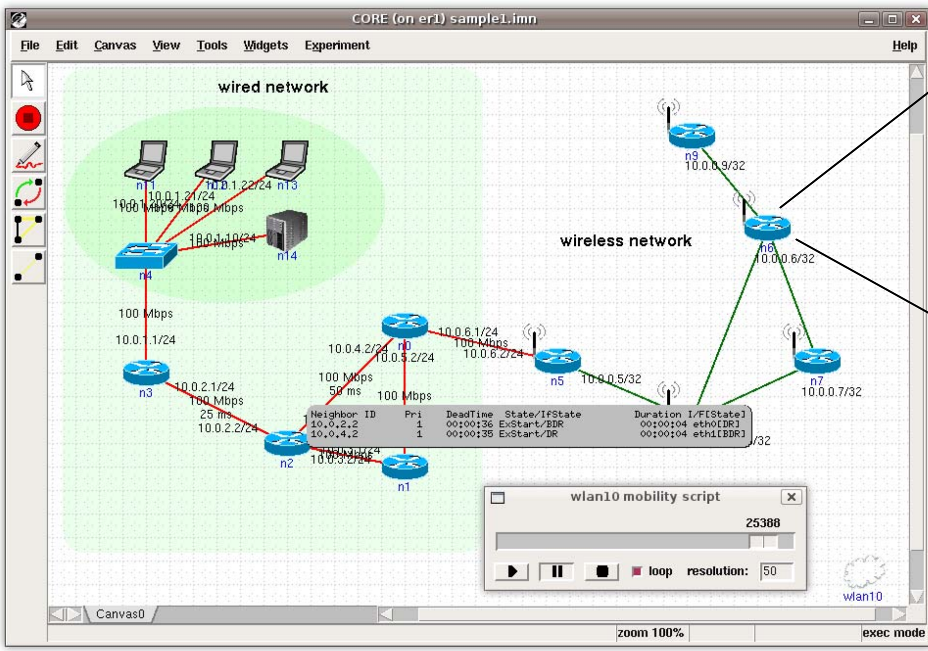
- Web site and code repository hosted by NRL ITD

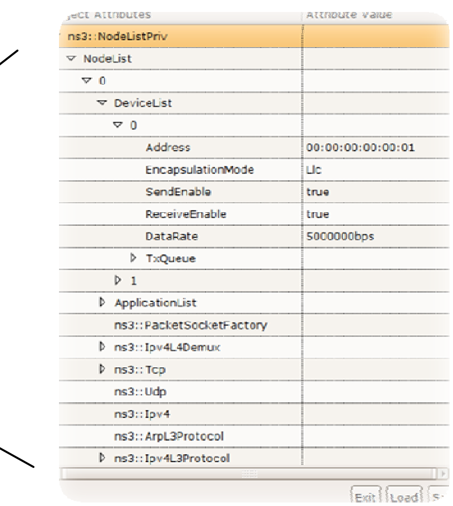


- Open source licensed
  - –modified BSD license
- Source code at NRL SVN
  - –https://pf.itd.nrl.navy.mil/sf/sfmain/do/
- Wiki/Bug tracker:
  - –http://code.google.com/p/coreemu/
- Mailing lists at NRL:
  - –core-users
  - –core-dev

Integrating ns-3 and GUIs

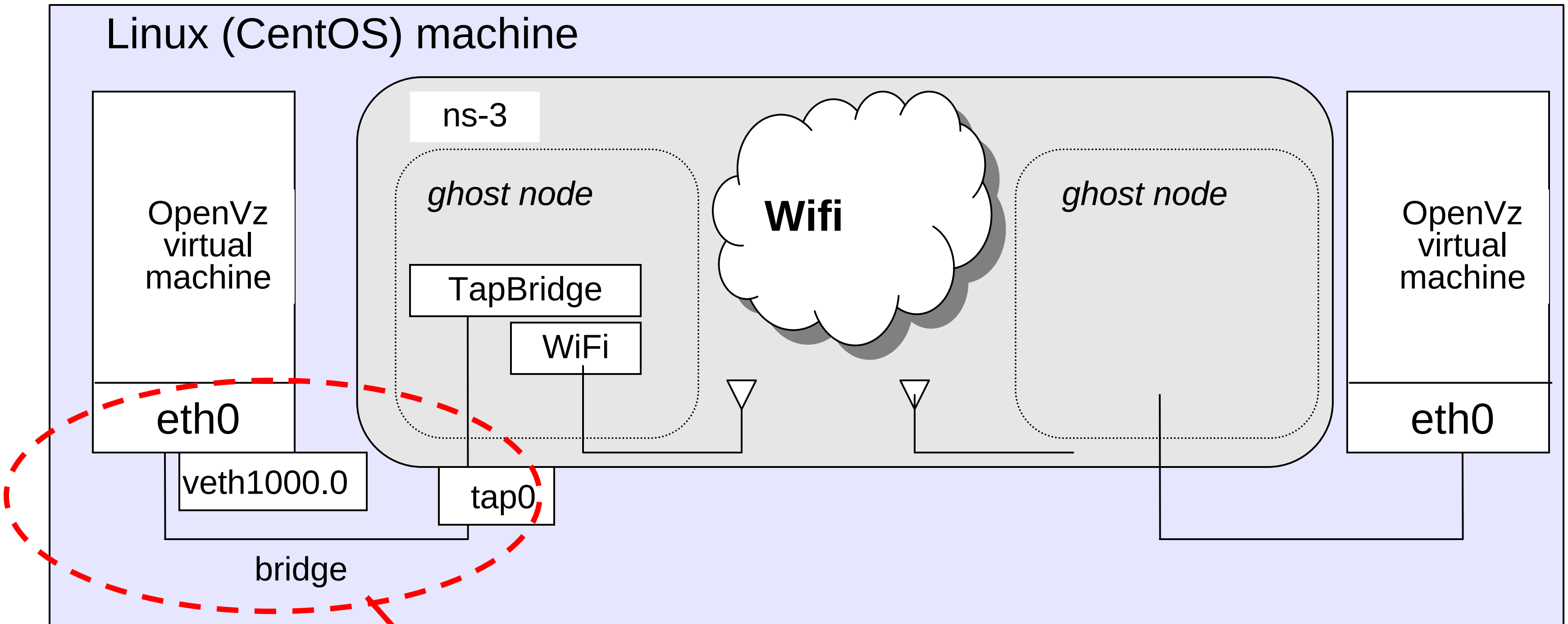
- Example CORE and ns-3
  - CORE could glue virtual machines to ns-3 networks





Not so fast...

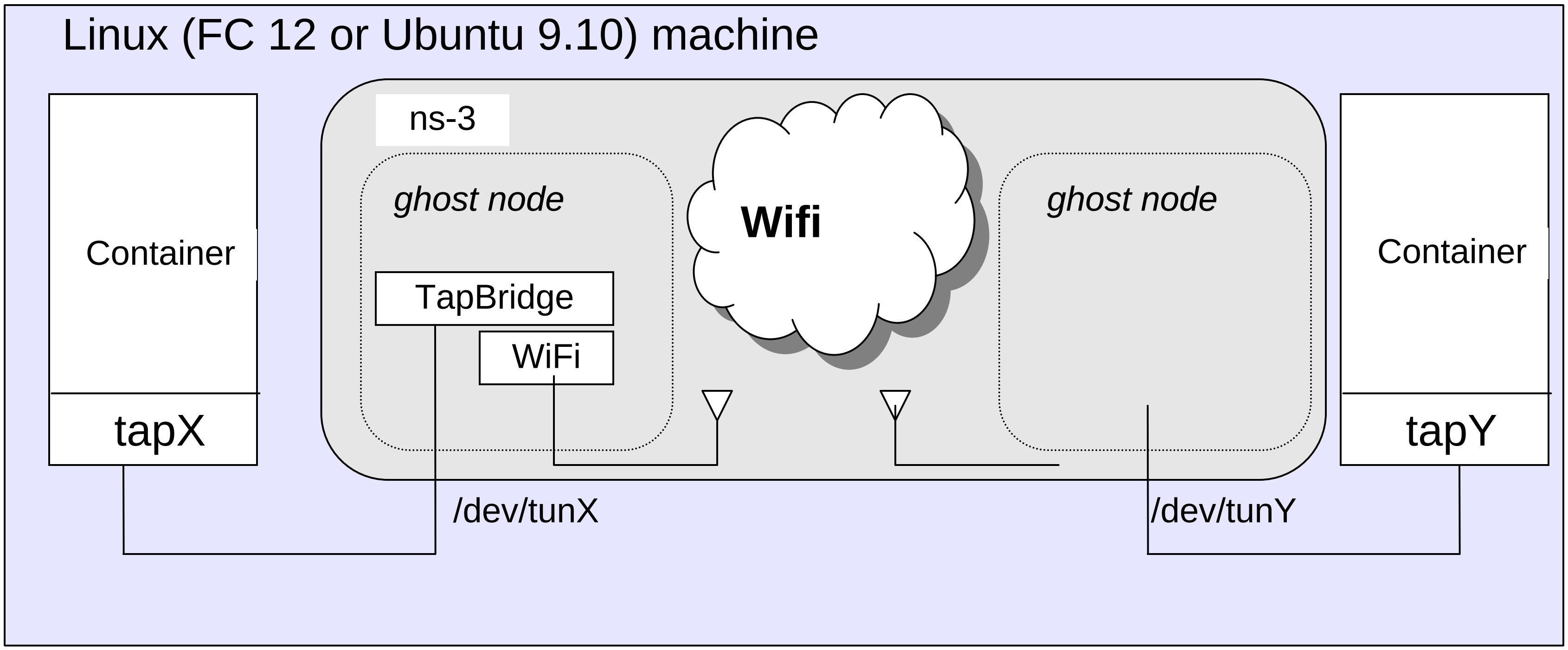
Initial steps: OpenVz and ns-3 integration



CORE virtual device bridged to Linux tap device, hooked to ns-3

Issues:
1) Linux bridging performance
2) MAC address coordination

“Tap” mode: netns and ns-3 integration



Tap device pushed into namespaces; no bridging needed

Issues

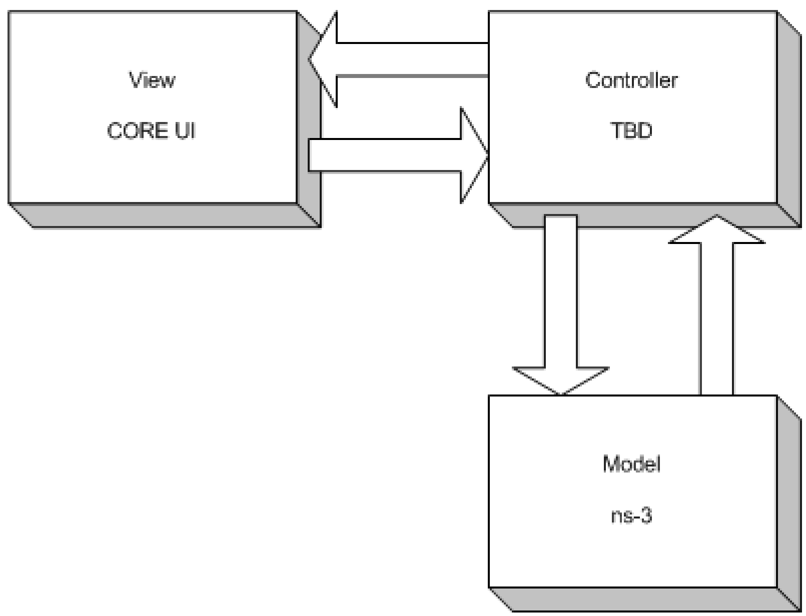
- Ease of use
  - Configuration management and coherence
  - Information coordination (two sets of state)
    - e.g. IP/MAC address coordination
  - Output data exists in two domains
  - Debugging
- Error-free operation (avoidance of misuse)
  - Synchronization, information sharing, exception handling
    - Checkpoints for execution bring-up
    - Inoperative commands within an execution domain
    - Deal with run-time errors
  - Soft performance degradation (CPU) and time discontinuities

Integrating ns-3 and GUIs

- What Happens Under the Sheets
  - Network is modeled (simulated);
  - View of network model and state is displayed;
  - UI actions are translated into model commands;
  - Model actions are translated into UI changes
- Variation on an old theme: Model-View-Controller (MVC)
  - Model: The internal state of the application;
  - View: The presentation to the user;
  - Controller: Maps user requests into actions and model state changes into view changes.

Integrating ns-3 and GUIs

- ns-3 is the model, CORE UI is the view
- What is the controller?



Integrating ns-3 and GUIs

- What is ultimately needed is a Controller Framework.
  - Controller is typically closely bound to the View;
  - Ideally, need a relatively abstract framework that provides a basis for building simulation GUIs as easily as possible;
  - Framework allows for controller implementations based on different models (ns-3, CORE native, etc.), and different view implementations (CORE UI, PyViz?, NetAnim?);
  - Inherit from generic framework and create a specific controller for your GUI that knows how to talk to ns-3.
- Sounds good, but what is it really?

Integrating ns-3 and GUIs

- Controller framework (to first approximation)
  - Provides proxies for first class objects in model;
  - Provides mechanism for backing objects with real system objects (bridges, taps, virtual machines);
  - Connects objects (nodes, devices and channels) into networks and adds objects to nodes (applications, mobility models, etc.);
  - Discovers attributes for first-class objects in model and provides configuration mechanisms to GUI;
  - Provides state / mode control (run, stop) and reporting (simulation running, stopped, VM ready, bridge up);
  - Mediates event reporting (allows tracing)

NEPI/NEF: Using Independent Simulators, Emulators, and Testbeds for Easy Experimentation

–Lacage, Ferrari, Hansen, Turletti (Roads 2009 workshop)

Controllers under development

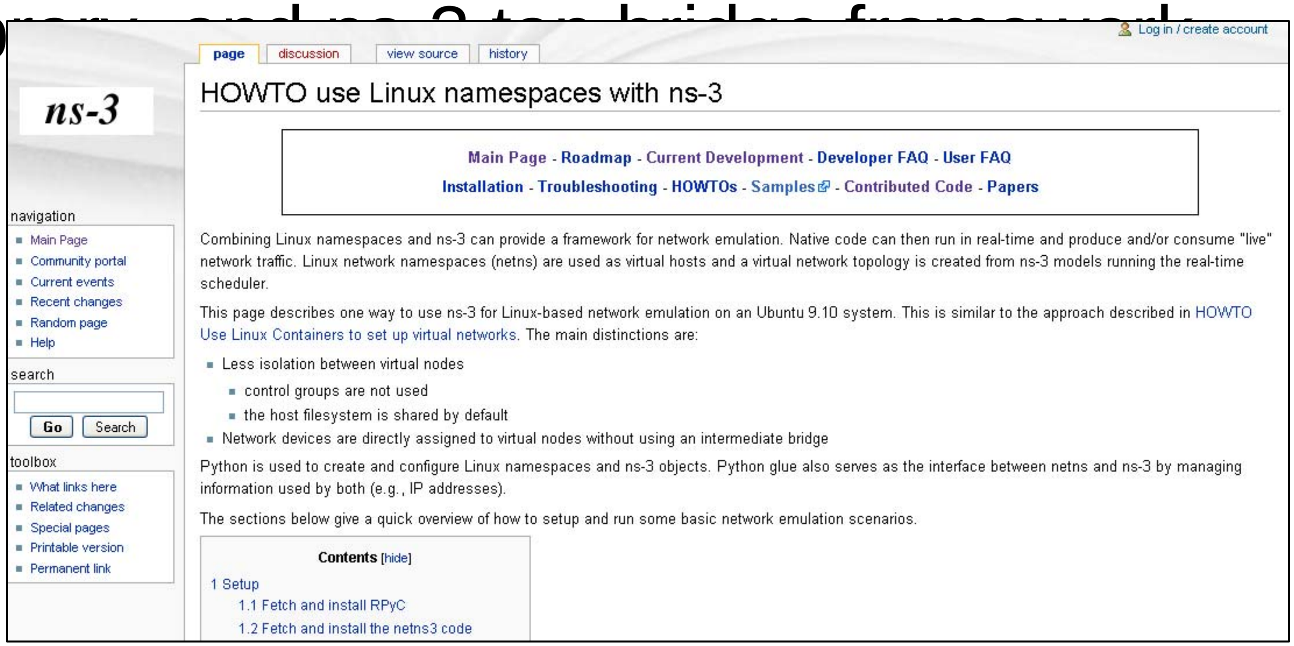
- NEPI:
  - **View:** Network Experimentation Frontend (NEF) GUI
  - **Backends:** ns-3, (virtual) machines, hybrids
- CORE
  - **View:** CORE GUI (or Python script)
  - **Backends:** netns, ns-3, EMANE
- netns3
  - **View:** Python program
  - **Backend:** netns + ns-3 (no abstraction)

Integrating ns-3 and GUIs

- Clearly a large effort. Can it be factored into something more manageable for the short term?
  - Start small and only build what you need today;
  - Start with a dedicated controller that glues CORE GUI to ns-3;
  - "CORE services" layer from before
  - Ignore the temptation to build a generic framework for now, just get a prototype done that we can evaluate.
  - Then pick the next UI and extend framework; repeat.

netns3

- Written by Tom Goff (Boeing)
  - Documentation and prototype posted on wiki
- Basic Python-based framework using ns-3 Python bindings, RPyC distributed computing library, and ns-3 tap bridge framework


Log in / create account

page | discussion | view source | history

ns-3

navigation

- Main Page
- Community portal
- Current events
- Recent changes
- Random page
- Help

search

Go | Search

toolbox

- What links here
- Related changes
- Special pages
- Printable version
- Permanent link

HOWTO use Linux namespaces with ns-3

Main Page - Roadmap - Current Development - Developer FAQ - User FAQ
Installation - Troubleshooting - HOWTOs - Samples - Contributed Code - Papers

Combining Linux namespaces and ns-3 can provide a framework for network emulation. Native code can then run in real-time and produce and/or consume "live" network traffic. Linux network namespaces (netns) are used as virtual hosts and a virtual network topology is created from ns-3 models running the real-time scheduler.

This page describes one way to use ns-3 for Linux-based network emulation on an Ubuntu 9.10 system. This is similar to the approach described in HOWTO Use Linux Containers to set up virtual networks. The main distinctions are:

- Less isolation between virtual nodes
  - control groups are not used
  - the host filesystem is shared by default
- Network devices are directly assigned to virtual nodes without using an intermediate bridge

Python is used to create and configure Linux namespaces and ns-3 objects. Python glue also serves as the interface between netns and ns-3 by managing information used by both (e.g., IP addresses).

The sections below give a quick overview of how to setup and run some basic network emulation scenarios.

Contents [hide]

1 Setup
1.1 Fetch and install RPyC
1.2 Fetch and install the netns3 code

netns3 demo

File Edit View Terminal Help

```python
# Miscellaneous helpers

import ns3
import netns
import subprocess
import optparse
from threading import Thread

ns3.GlobalValue.Bind("SimulatorImplementationType",
                     ns3.StringValue("ns3::RealtimeSimulatorImpl"))

ns3.GlobalValue.Bind("ChecksumEnabled", ns3.BooleanValue("true"))

class Ns3Netns(netns.Netns):

    def addnetif(self, ifname, ipaddrs = [],
                 rename = "eth0", up = True, now = False):
        """\
        Add a network interface to the netns and do basic
        configuration.  By default, the given network interface is
        renamed eth0, given IP address(es), and brought up at when the
        simulation starts.  Waiting until the simulation runs ensures
        tap devices created by ns-3 exist.
        """
        def doaddnetif():
            self.acquire_netif(ifname, rename)
            if rename:
                name = rename
            else:
                name = ifname
            if up:
                self.ifup(name)
            for ipaddr in ipaddrs:
                self.add_ipaddr(name, ipaddr)
        if now:
            doaddnetif()
        else:
            # schedule when simulation starts
            ns3.Simulator.Schedule(ns3.Time("0"), doaddnetif)

class NetnsNode(ns3.Node, Ns3Netns):
                                                1,1           Top
```

tomh@u94-desktop: ~/wns3/ns3netns/examples

File Edit View Terminal Help

```python
    def createnet(self, devhelper, nodecontainer,
                  addrhelper, mask, ifnames = []):
        tapbridge = ns3.TapBridgeHelper()
        devices = devhelper.Install(nodecontainer)
        for i in xrange(nodecontainer.GetN()):
            n = nodecontainer.Get(i)
            dev = devices.Get(i)
            tap = tapbridge.Install(n, dev)
            tap.SetMode(ns3.TapBridge.CONFIGURE_LOCAL)
            tapname = "ns3tap%d" % i
            tap.SetAttribute("DeviceName", ns3.StringValue(tapname))
            addr = addrhelper.NewAddress()
            tap.SetAttribute("IpAddress", ns3.Ipv4AddressValue(addr))
            tap.SetAttribute("Netmask", ns3.Ipv4MaskValue(mask))
            if ifnames:
                ifname = ifnames[i]
            else:
                ifname = "eth0"
            n.addnetif(tapname,
                       ipaddrs = ["%s/%s" % (addr, mask.GetPrefixLength())],
                       rename = ifname)
            n.ipaddr = str(addr)

    def createnodes(self, numnodes, devhelper, prefix = "10.0.0.0/8",
                    nodenum = 0):
        addrhelper, mask = parseprefix(prefix)
        nc = ns3.NodeContainer()
        for i in xrange(numnodes):
            name = "n%s" % nodenum
            n = NetnsNode(name, logfile = "/tmp/%s.log" % name)
            self.nodes.append(n)
            nc.Add(n)
            nodenum += 1
        self.createnet(devhelper, nc, addrhelper, mask)

    def run(self):
        self.setup()
        print "running simulator for %s sec" % self.options.simtime
        t = self.simthread(self.options.simtime)
        t.join()
                                                   81,0-1        43%
```

Next steps

- NRL is funding UW to prototype CORE + ns-3
  - Initial prototype for a concrete (adhoc wifi) scenario
  - Identify integration issues, deal with some of them
  - Learn about scaling limits and pitfalls to avoid
  - Coordinate with Jeff Ahrenholz, Tom Goff (Boeing) and INRIA (others?)
- Possible areas of future work
  - Time warping (e.g. Xen-based prototype)
  - Distributed across physical hosts
  - Kernel packet treatments or hardware (high speed forwarding)