

---

# ns-3 development overview

# ns-3 tutorial agenda

---

- 3:00-4:30: ns-3 current capabilities
  - Project overview
  - Walkthrough of basic simulation scenario
  - Parallel simulations and visualization
  - Emulation
- 4:30-4:40: 10-minute break
- 4:40-5:45: Work in progress
  - ns-3 development process
  - Automation
  - Direct code execution
  - Virtual machine and testbed integration
- 5:45-6:00: Q & A

# ns-3 development process

---

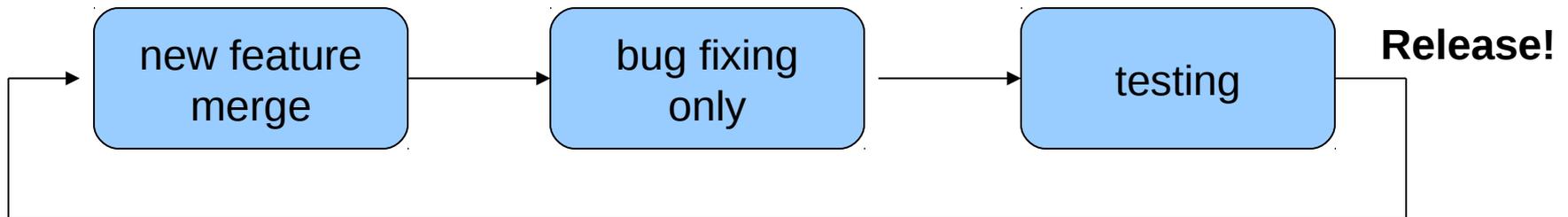
ns-3 is run as an open source project backed by research funding

- GPLv2 licensing stance
- open mailing lists
- use standard tools (Mercurial, Bugzilla, Mediawiki, GNU/Linux development)
- 13 maintainers worldwide

# ns-3 development process

---

- date-driven quarterly releases



- All code for merge to ns-3 is openly reviewed by maintainers
  - \_Syntactic (style) reviews
  - \_Design reviews
  - \_Documentation and tests

# current merge queue

---

- ns-3.10 release (January 2010)
  - new TCP model with modular congestion control
  - Virtual Access Point (VAP) for WiFi
  - BulkSend application
  - Pyviz visualizer
  - Energy model for WiFi
  - DSDV routing for IPv4
  - PhySim for WiFi



# Other announced projects

---

- Wireless jamming model
- MPLS
- VANET mobility model
- TDMA
- TCP Vegas
- DSR routing
- SimpleWireless model
- Zigbee, WPAN, and 6LOWPAN
- Chord/DHash DHT
- Synchronized emulation
- 802.11n
- TMix and DelayBox
- ns-3-simu
- multi-core parallelization

# Modularity and model store

---

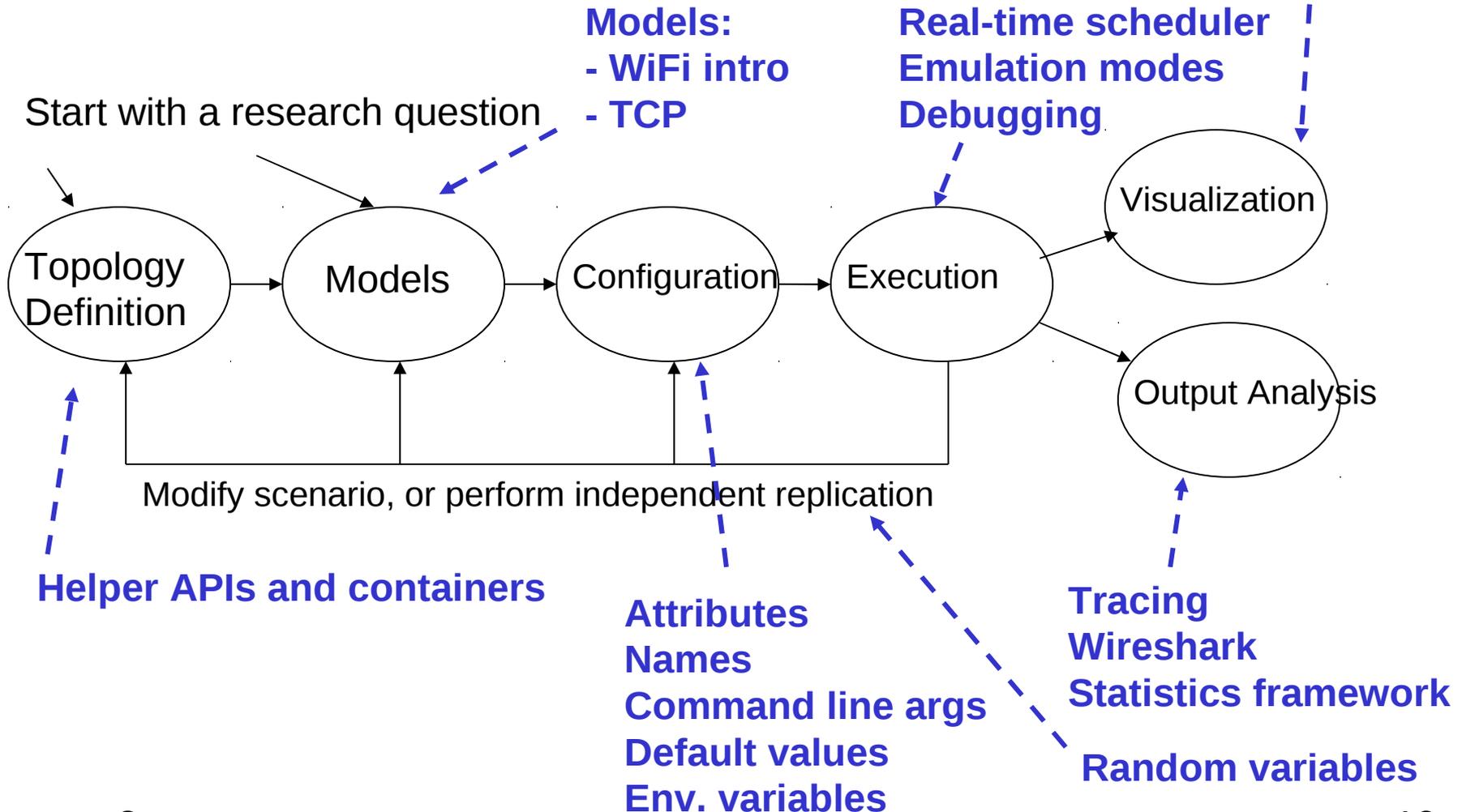
- Moving to a modular build and package management system
  - ns-3 project maintains the core
  - models may be enabled/disabled
  - other research groups may separately maintain their own models
    - maintainers can still provide reviews
    - common package metadata format used to inform ns-3 build system

---

# automation

# Overview of ns-3 features

Examples



# Motivation

---

Network simulation is no easy business. One must:

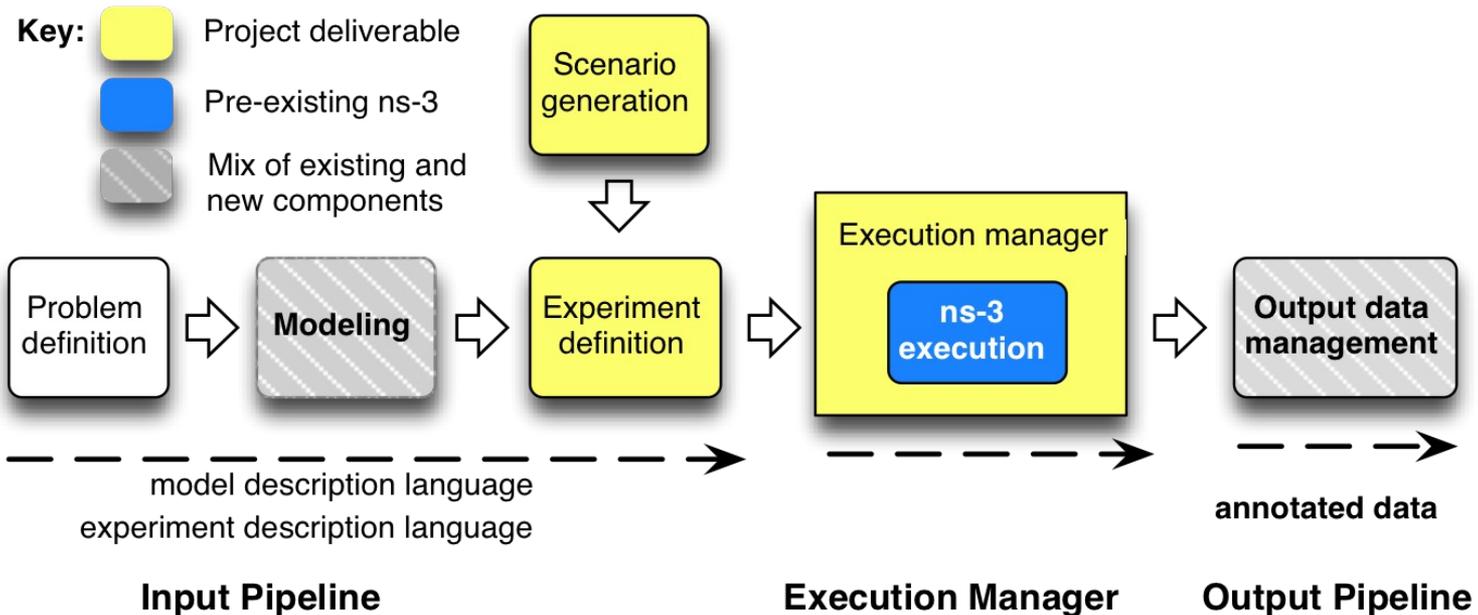
- Build a model that is consistent.
- Describe the simulation model for a given simulator.
- Design and execute experiments.
- Process output data using sound methodologies.

We raise the level of abstraction on the user interface to the network simulator to support the needs of both experts and novices.

We can address issues that undermine credibility.

# Frameworks for ns-3

- NSF CISE Community Research Infrastructure
  - University of Washington (Tom Henderson), Georgia Tech (George Riley), Bucknell Univ. (Felipe Perrone)
  - Project timeline: 2010-14



# Automation

---

- Task led by Felipe Perrone, Bucknell Univ.
- Inspired by SWAN-Tools and ANSWER frameworks.
- User interfaces, description languages, and tools for automation of experiments.
- Model composition, structural validation, control of experiments, data processing and storage, and archiving experimental setup.

# Topology generation

---

- Integrate BRITE topology generator (Boston Univ.) into framework.
- BRITE is downloaded into distribution and compiled by the ns-3 build system.
- The ns-3 simulation script uses a topology *helper* which reads a BRITE configuration file, receives results from BRITE, and builds the ns-3 topology.

# Broader use case

- Provide a model and a description of experiment.
- Framework generates design of experiment space, distribute simulation runs to machines, collect results and archive in persistent storage.
- User mines storage to find, extract, and visualize results.

The image shows two screenshots of web-based configuration interfaces for ns-3 simulations, both accessed via http://localhost:3000.

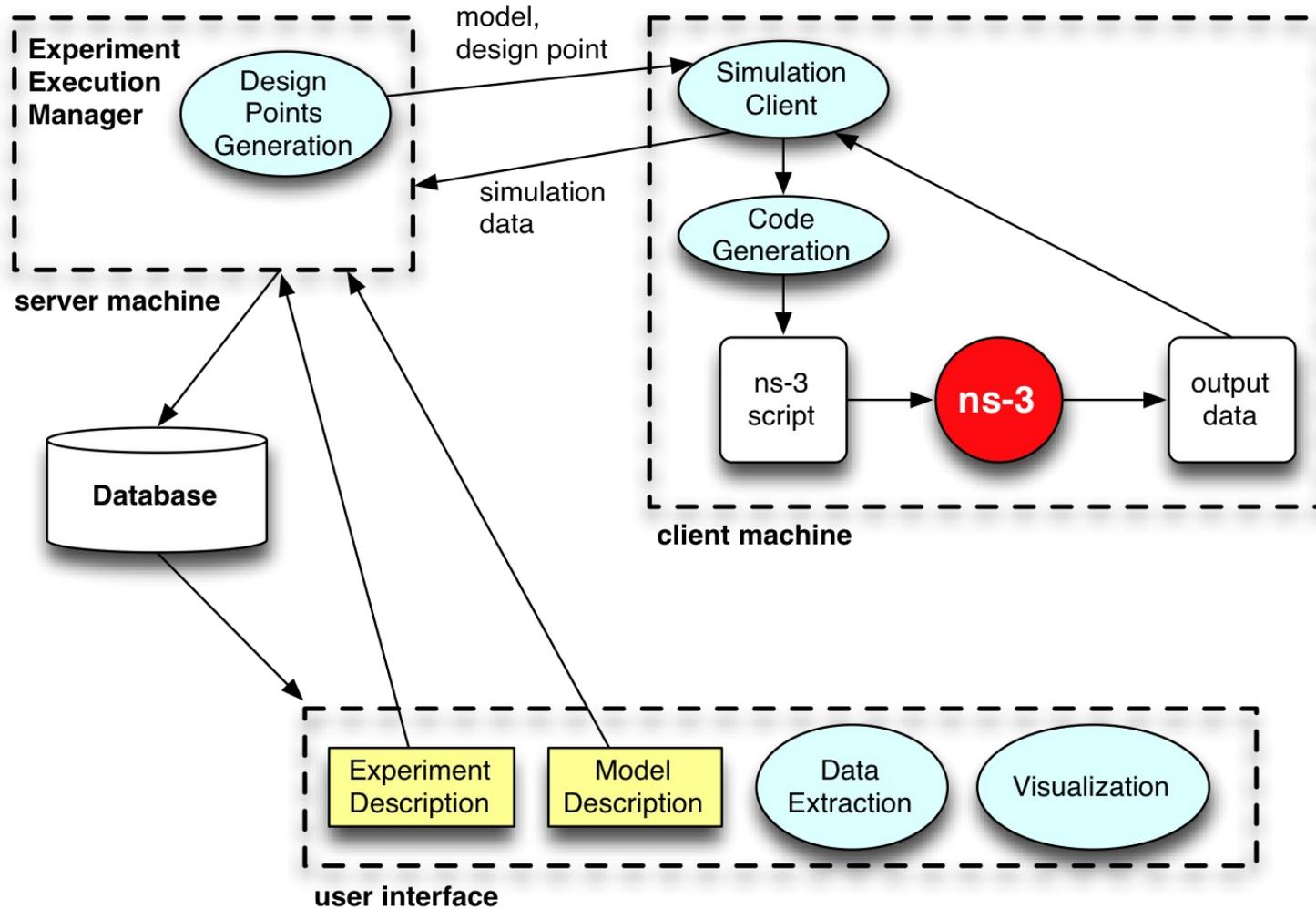
The top screenshot is the **AODV Model Configuration Interface**. It features several configuration options:

- Packet priority: 0
- Local repair: true/false (dropdown)
- Expanding ring search: true/false (dropdown)
- Active route timeout: 10 seconds
- Route request retries: 2
- ... (more options)
- Buttons: **Inspect/Configure Defaults** and **Done**

The bottom screenshot is the **Model Composition Interface**. It is divided into several sections:

- Initial Node Deployment:** User Assigned, Regular Grid, Random Uniform, Random Cluster, Random Triangular (dropdown)
- Mobility:** Stationary, Random Waypoint, Brownian, Gauss-Markov (dropdown)
- Radio propagation:** Friis Free Space, 2-Ray, Lognormal Shadowing, Okumura-Hata, Walfish-Ikegami (dropdown)
- Wireless Node:** 802.11PHY, 802.16PHY, 802.11MAC, 802.16MAC, IPv4, IPv6, ARP, ICMPv4, ICMPv6, AODV, DSR, UDP, TCP, CBR source, VBR source (checkboxes)
- Terrain:** Flat, Elevation Map (dropdown)
- Button: **Validate Model Structure**

# General architecture



---

# Direct Code Execution

---

# Virtual machines and ns-3

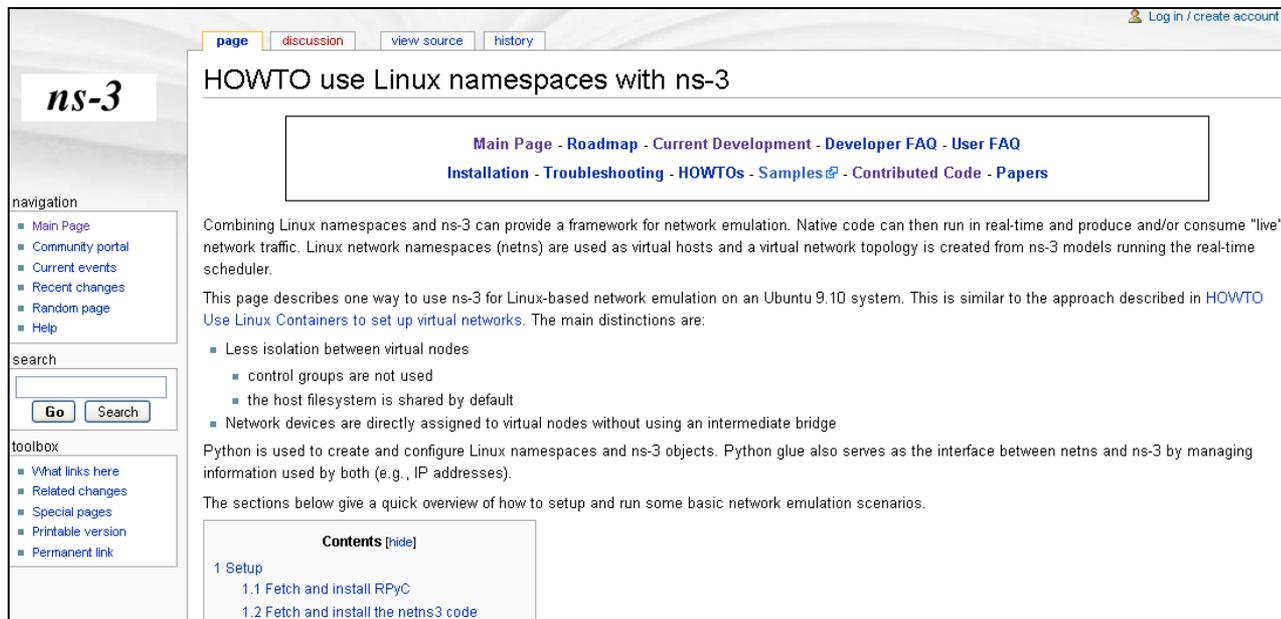
# Goals

---

- 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

# 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



The screenshot shows a web browser displaying a wiki page for ns-3. The page title is "HOWTO use Linux namespaces with ns-3". The page content includes a navigation menu, a search box, and a list of links: "Main Page - Roadmap - Current Development - Developer FAQ - User FAQ", "Installation - Troubleshooting - HOWTOs - Samples - Contributed Code - Papers". The main text describes how to use ns-3 for Linux-based network emulation on an Ubuntu 9.10 system. It mentions that Python is used to create and configure Linux namespaces and ns-3 objects. The page also includes a "Contents" section with links to "1 Setup", "1.1 Fetch and install RPyC", and "1.2 Fetch and install the netns3 code".

# netns3 demo

```
File Edit View Terminal Help
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

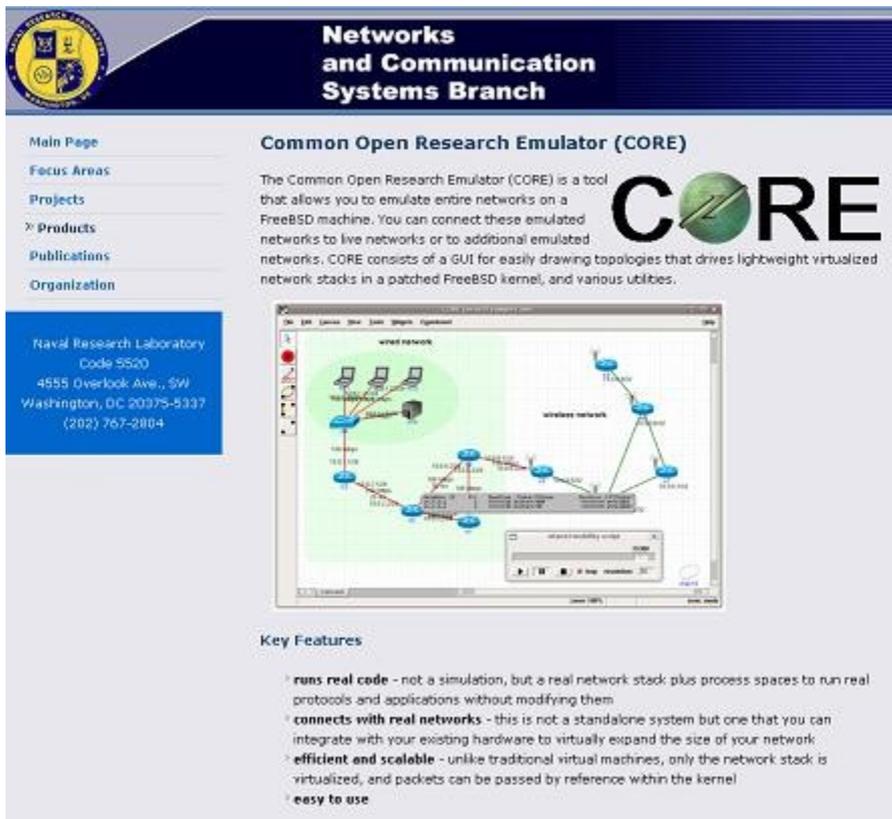
def createnet(self, devhelper, nodecontainer,
              addrhelper, mask, ifnames = []):
    tapbridge = ns3.TapBridgeHelper()
    devices = devhelper.Install(nodecontainer)
    for i in xrange(nodecontainer.GetN()):
        n = nodecontainer.Get(i)
        dev = devices.Get(i)
        tap = tapbridge.Install(n, 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%
```

# CORE is an open source project

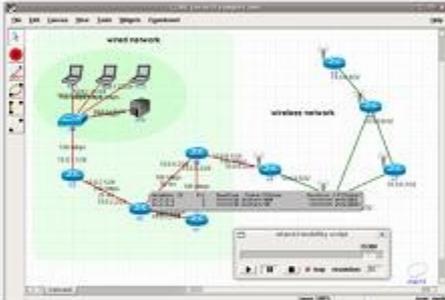
- Web site and code repository hosted by NRL ITD



**Networks and Communication Systems Branch**

**Common Open Research Emulator (CORE)**

The Common Open Research Emulator (CORE) is a tool that allows you to emulate entire networks on a FreeBSD machine. You can connect these emulated networks to live networks or to additional emulated networks. CORE consists of a GUI for easily drawing topologies that drives lightweight virtualized network stacks in a patched FreeBSD kernel, and various utilities.



**Key Features**

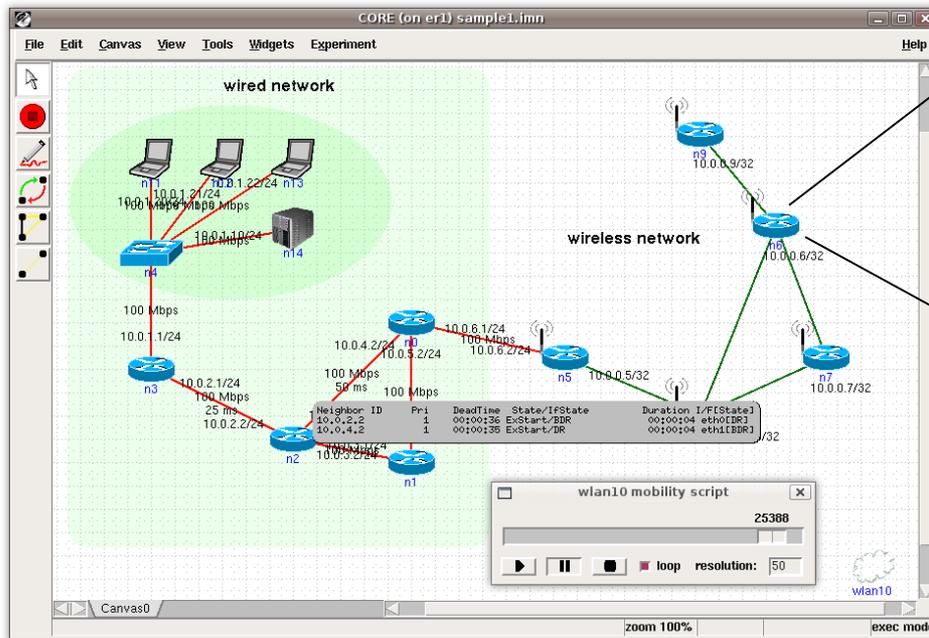
- **runs real code** - not a simulation, but a real network stack plus process spaces to run real protocols and applications without modifying them
- **connects with real networks** - this is not a standalone system but one that you can integrate with your existing hardware to virtually expand the size of your network
- **efficient and scalable** - unlike traditional virtual machines, only the network stack is virtualized, and packets can be passed by reference within the kernel
- **easy to use**

Naval Research Laboratory  
Code 5520  
4555 Overlook Ave., SW  
Washington, DC 20375-5337  
(202) 767-2804

- 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

# Future work: Integrating ns-3 and GUIs

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



Object Attributes	Attribute Value
ns3::NodeListPriv	
NodeList	
0	
DeviceList	
0	
Address	00:00:00:00:00:01
EncapsulationMode	Llc
SendEnable	true
ReceiveEnable	true
DataRate	5000000bps
TxQueue	
1	
ApplicationList	
ns3::PacketSocketFactory	
ns3::Ipv4L4Demux	
ns3::Tcp	
ns3::Udp	
ns3::Ipv4	
ns3::ArpL3Protocol	
ns3::Ipv4L3Protocol	

# 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

# Scaling time in virtualized environments

---

- Synchronized Network Emulation - RWTH Aachen University
  - Modified Xen
- VAN Testbed – Telcordia/CERDEC
  - Modified Xen
- Linux Time namespace - Jeff Dike (UML creator)
  - Add a time namespace to the Linux kernel, allowing for `gettimeofday()` offsets

---

# NEPI

# Resources

---

Web site:

<http://www.nsnam.org>

Mailing list:

<http://mailman.isi.edu/mailman/listinfo/ns-developers>

IRC: #ns-3 at freenode.net

Tutorial:

<http://www.nsnam.org/docs/tutorial/tutorial.html>

Code server:

<http://code.nsnam.org>

Wiki:

[http://www.nsnam.org/wiki/index.php/Main\\_Page](http://www.nsnam.org/wiki/index.php/Main_Page)