ns-3 Direct Code Execution (DCE)

ns-3 Annual Meeting

June 2018

Outline

- DCE Introduction
- Download and Installation of DCE with Bake
- DCE examples with custom application:
  - iperf
  - ping
- Q&A

Credits

- DCE originated with Mathieu Lacage (INRIA), was substantially developed by Emilio Mancini and Frederic Urbani (INRIA), and finished off by Hajime Tazaki

- DCE leverages the bake build system substantially developed by Daniel Camara (INRIA)

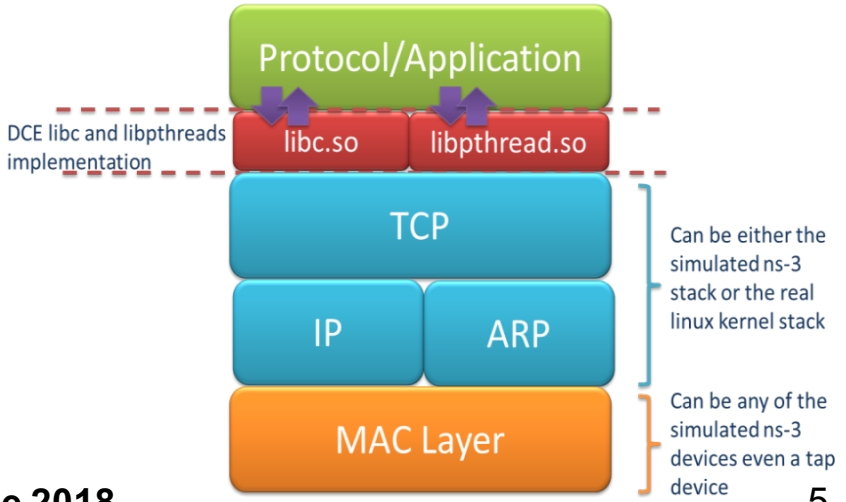
- DCE is presently maintained by Hajime Tazaki and Matthieu Coudron.

Goals

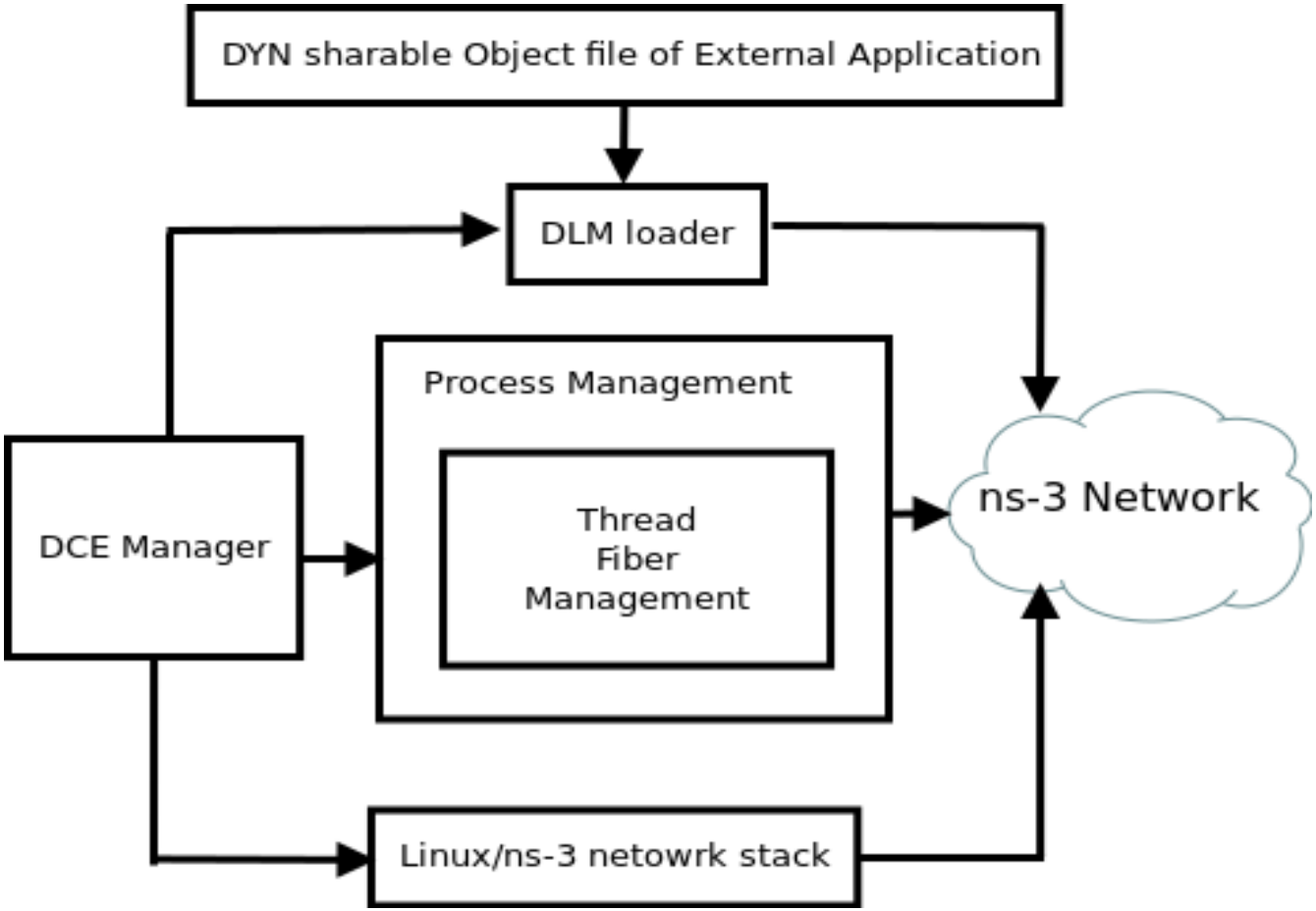
- Lightweight virtualization of kernel and application processes, interconnected by simulated networks
- Benefits:
  - Implementation realism in controlled topologies or wireless environments
  - Model availability
  - Debugging a whole network within a single process
- Limitations:
  - Not as scalable as pure simulation
  - Tracing more limited
  - C or C++ applications only

Direct Code Execution

- DCE/ns-3 framework requires the virtualization of a series of services
  - Multiple isolated instances of the same protocol on the same machine
- System calls are captured and treated by DCE
- Network stack protocols calls are captured and redirected
- To perform its work, DCE re-implements the Linux program loader and parts of *libc* and *libpthread*



DCE Components



DCE modes

- DCE modes in context of possible approaches

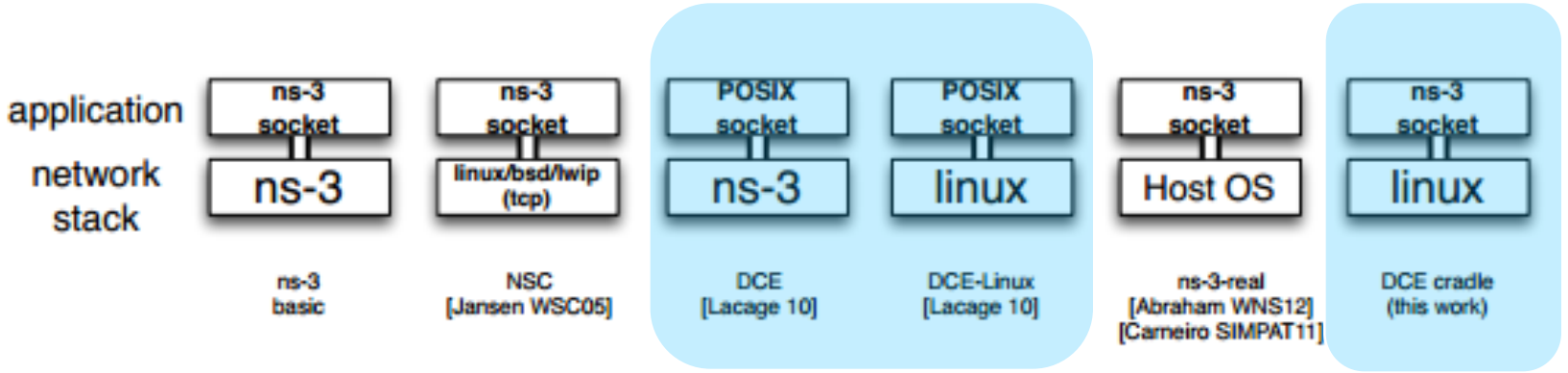


Figure 1: Current possible combinations of network stacks and applications.

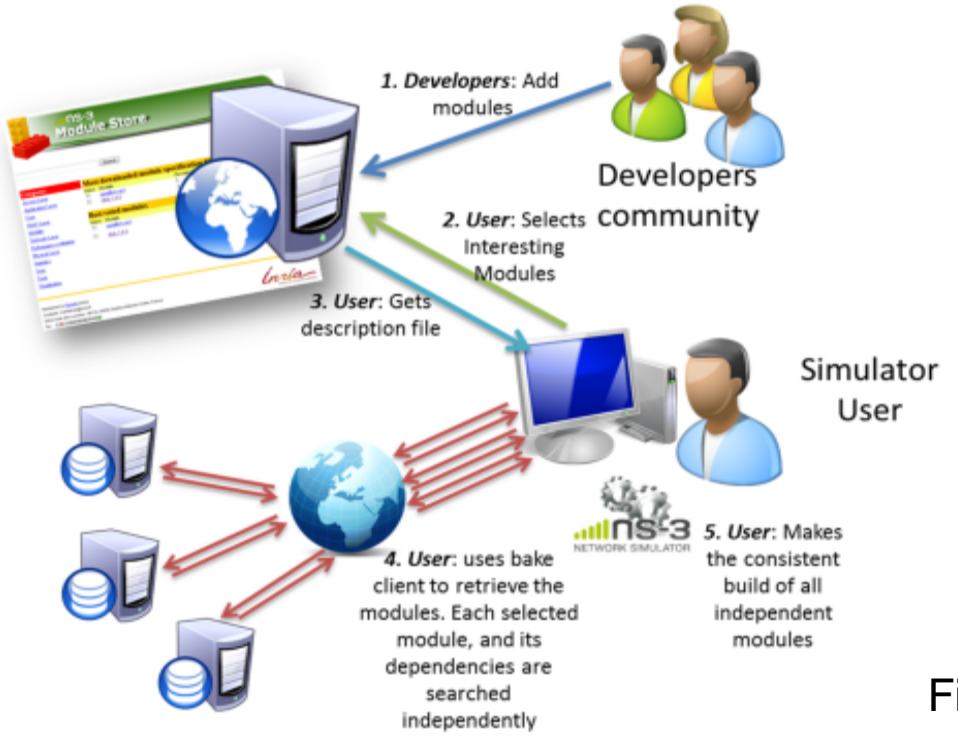
Figure source: DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism, Tazaki et al, WNS3 2013.

References

- Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments
  - Tazaki et al, CONEXT 2013
  - http://hal.archives-ouvertes.fr/docs/00/88/08/70/PDF/con013-hal.pdf
- DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism
  - Tazaki et al, WNS3 2013
  - http://hal.archives-ouvertes.fr/docs/00/78/15/91/PDF/wns3-2013.pdf
- NetDev 1.1 talk (Feb. 2016): "LibOS as a regression test framework for Linux networking"
  - https://www.youtube.com/watch?v=FNGkMFmboEc

bake overview

- Open source Integration tool to simplify the build process by automatically builds the software with required dependencies.



"bake" tool
(Lacage and Camara)

Components:

- build client
- "module store" server
- module metadata

Figure source: Daniel Camara

bake basics

- bake can be used to build the Python bindings toolchain, Direct Code Execution, Network Simulation Cradle, etc.
- Steps to setup DCE and using Bake

```
export BAKE_HOME=`pwd`/bake          // Set environment variables
export PATH=$PATH:$BAKE_HOME         // to locate bake
export PYTHONPATH=$PYTHONPATH:$BAKE_HOME
```

```
./bake.py configure -e <module>
./bake.py check      // bake tool dependency
./bake.py show       // <module> related dependency
./bake.py download   // Download <module> specific source
./bake.py build      // Build the source code
```

Placeholder slide for demoing bake

Demo: `bake.py configure -e ns-allinone-3.25`
Look at: `build/` libraries and executables

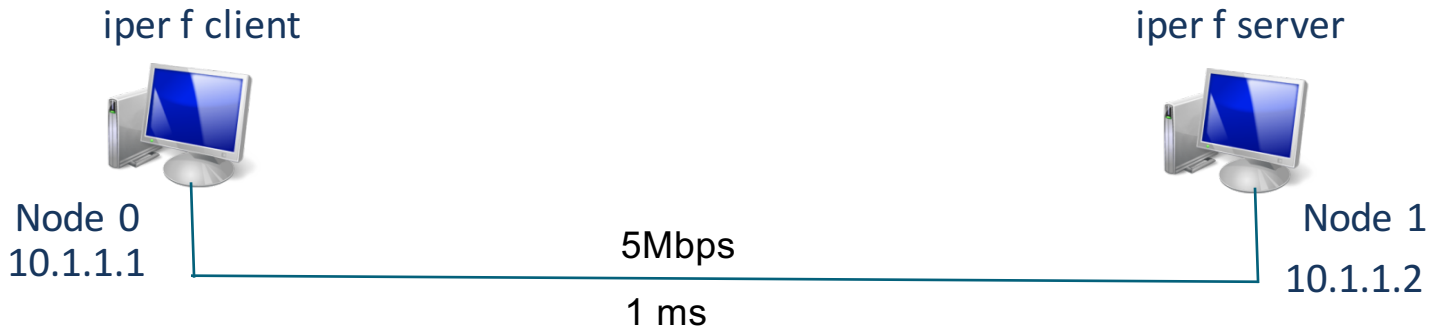
Manual available at:
https://www.nsnam.org/docs/bake/tutorial/html/index.html

bakeconf example

```
        <module name="ns-allinone-3.28">
        <source type="none"/>
        <depends_on name="netanim-3.108" optional="True"/>
        <depends_on name="nsc-0.5.3" optional="True"/>
        <depends_on name="pybindgen-ns3.27-castxml"
optional="True"/>
        <depends_on name="pyviz-prerequisites"
optional="True"/>
        <depends_on name="click-ns-3.25" optional="True"/>
        <depends_on name="openflow-ns-3.25" optional="True"/>
        <depends_on name="pygccxml-1.9.1" optional="True"/>
        <depends_on name="BRITE" optional="True"/>
        <depends_on name="ns-3.28" optional="False"/>
        <build type="none"/>
        </module>
```

Iperf example in DCE

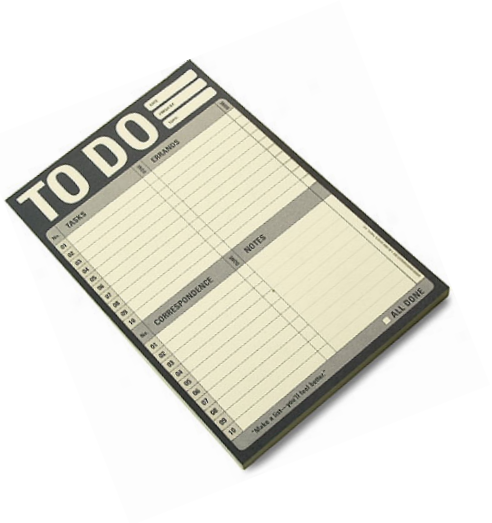
- ns-3-dce/example/dce-iperf.cc



iperf: it is a widely used tool for network performance measurement and tuning.
[text source: Wikipedia]

Step by step example - What we need to do!

1. Create the nodes
2. Create stack
3. Create devices
4. Set addresses
5. Connect devices
6. Create DCE
7. Configuration the applications to run
8. Set start time for server and client
9. Set simulation time
10. Start simulation



Step by step example - What we need to do!

1. Create the nodes
2. Create stack
3. Create devices
4. Set addresses
5. Connect devices
6. Create DCE
7. Configuration the applications to run
8. Set start time for server and client
9. Set simulation time
10. Start simulation

Standard ns-3 procedures
DCE specific

Application configuration in DCE

```
DceApplicationHelper dce;

dce.SetStackSize (1 << 20);  // 1MB, default: 8kB


dce.SetBinary ("application");
dce.ResetArguments ();
dce.ResetEnvironment ();


dce.AddArgument ("first");
    dce.AddArguments ("second");
dce.AddArguments ("third");
dce.Install (nodes.Get (0));
```

Sample Command: **application first second third**

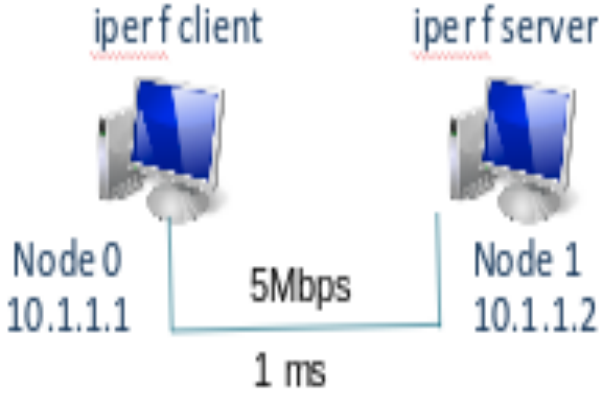
Step by step example iperf

```
int main (int argc, char *argv[])
{

    // Select network stack
    std::string stack = "ns3";

    // Node Container creation
    NodeContainer nodes;
    nodes.Create (2);

        // Device and channel  creation
        PointToPointHelper pointToPoint;
    NetDeviceContainer devices;
```



Step by step example iperf

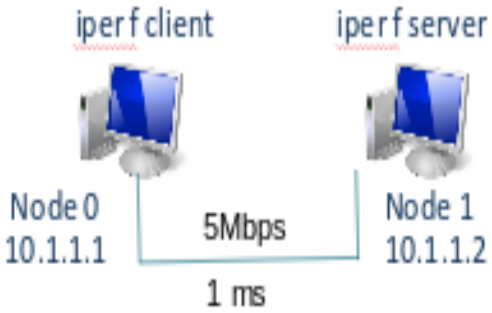
```
// Select Stack type
if (stack == "ns3") {
  InternetStackHelper stack;
  stack.Install (nodes);
  dceManager.Install (nodes);

 } else if (stack == "linux") {

     dceManager.SetNetworkStack (
   "ns3::LinuxSocketFdFactory ", "Library",
   StringValue ("liblinux.so"));

   dceManager.Install (nodes);
   LinuxStackHelper stack;
   stack.Install (nodes);
 } else if (stack == "freebsd") {
                ...
  }
```

Step by step example iperf




```
//  Node0-Node1 setup
  Ipv4AddressHelper address;
  //  Node0-Node1 addresses
  address.SetBase ("10.1.1.0", "255.255.255.252

  NetDeviceContainer devices;
  // connecting nodes
  devices = p2p.Install (nodes.Get (0), nodes.Get (1));

  //  assign addresses
  Ipv4InterfaceContainer interfaces = address.Assign
(devices);

  // setup ip routes
  Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
  if (stack == "linux")
        {
           LinuxStackHelper::PopulateRoutingTables ();
    }
```

Step by step example iperf

```
DceManagerHelper dceManager;
dceManager.Install (nodes);

DceApplicationHelper dce;
ApplicationContainer apps;
 dce.SetStackSize (1 << 20); // 1MB stack

dce.SetBinary ("iperf"); // Launch iperf client on node 0
dce.ResetArguments ();  // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-c");  // client
dce.AddArgument ("10.1.2.2"); //target machine address
dce.AddArgument ("-i"); // interval
dce.AddArgument ("1");
dce.AddArgument ("--time"); // how long
dce.AddArgument ("10");
 apps = dce.Install (nodes.Get (0)); //install application
apps.Start (Seconds (0.7)); //start at 0.7 simulation time
apps.Stop (Seconds (20)); //stop at 20s        simulation time

dce.SetBinary ("iperf"); // Launch iperf server on node 2
dce.ResetArguments (); // clean arguments
dce.ResetEnvironment (); // clean environnent
dce.AddArgument ("-s"); // server
dce.AddArgument ("-P"); // number of paralell servers
dce.AddArgument ("1");
apps = dce.Install (nodes.Get (2));
apps = dce.Install (nodes.Get (2));
apps.Start (Seconds (0.6));
```

DCE Setup

Step by step example iperf

```
  // Simulation stop time
  Simulator::Stop (Seconds (40.0));

  // Run
  Simulator::Run ();

  // Stop
  Simulator::Destroy ();

  return 0;
}
```

With/without linux kernel

- ns-3 stack
  - `./waf --run "dce-iperf --stack=ns3"`
- Linux stack
  - `./waf --run "dce-iperf --stack=linux"`
- More options
  - `./waf --run "dce-iperf --PrintHelp"`

Step by step example – iperf, ns-3

- Generated
  - elf-cache – program files
  - exitprocs – execution process information
  - files-0 files-1 – execution filesystem
- files-x
  - files-x corresponds to “/” of the machine x
  - files-x/var/log/<pid>/
    - cmdline – command executed
    - status – execution information
    - stderr – standard error output
    - stdout – standard output
    - syslog – syslog output

Build Custom Application for DCE: Ping

- Collect the source code and untar:

  wget http://www.skbuff.net/iputils/iputils-s20101006.tar.bz2

  tar xvjf iputils-s20101006.tar.bz2

- Add extra LDFLAGS in Makefile

  LDFLAGS+=-pie -rdynamic

- Build ping application

  make CFLAGS=-fPIC ping

- Check binary is read to execute in DCE

  readelf -h ping

Header Section of Ping binary

```
Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              DYN (Shared object file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
```

Running Ping Example

- Ping example: ns-3-dce/myscripts/ping/dce-ping.cc

  ./waf --run dce-ping

- Need to update DCE_PATH to external application

  assert failed. cond="exeFullPath.length () > 0", msg="Executable 'ping' not found ! Please check your DCE_PATH and DCE_ROOT environment variables.", file=../model/dce-manager.cc, line=262

- Fix the DCE_PATH

  export DCE_PATH=$DCE_PATH:/path/to/ping

Tuning DCE

DCE is configurable with NS3 Attributes. Refer to the following table:

ATTRIBUTE NAME	DESCRIPTION	VALUES	EXAMPLES

FiberManagerType	The TaskManager is used to switch the execution context between threads and processes.	**UcontextFiberManager** the more efficient.
		
		PthreadFiberManager helpful with **gdb** to see the threads. This is the de fault.
			`--ns3::TaskManager::FiberManagerType=UcontextFiberManager`
			
			`dceManager.SetTaskManagerAttribute("FiberManagerType", StringValue("UcontextFiberManager"));`
			
			`--ns3::TaskManager::FiberManagerType=PthreadFiberManager`

LoaderFactory	The LoaderFactory is used to load the hosted binaries.	**CoojaLoaderFactory** is the default and the only one that supports `fork`.
		
		DlmLoaderFactory is the more efficient. To use it you have two ways:
		
		1. use `dce-runner`
		
		2. link using `ldso` as default interpreter.
			`--ns3::DceManagerHelper::LoaderFactory=ns3::CoojaLoaderFactory[]`
			
			`$ dce-runner my-dce-ns3-script`
			
			OR
			
			`gcc -o my-dce-ns3-script Wl,--dynamic-linker=PATH2LDSO/ldso ...`
			
			`$ my-dce-ns3-script --ns3::DceManagerHelper::LoaderFactory=ns3::Dl mLoaderFactory[]`
			
			`dceManager.SetLoader("ns3::DlmLoaderFactory");`

Practical issues

- All software must be re-compiled with –fpic and linked with –pie to generate the code as Position Independent Code (PIC)
- Executables need to be installed to a place where DCE can find it
- Applications may require system dependencies
- Requires coverage of selected POSIX calls
- Execution must be managed (e.g. start/stop time)
- How much stack size to allocate to programs?
- Programs generate output (stdout, stderr, files)

Advantages of DCE

- Simulation with Linux network stack.
- Allow userspace applications to run inside ns-3.
- Debug posix based applications in single address space.

Future Work

- Support for latest linux kernel using Linux Kernel Library (LKL).