DCE Cradle

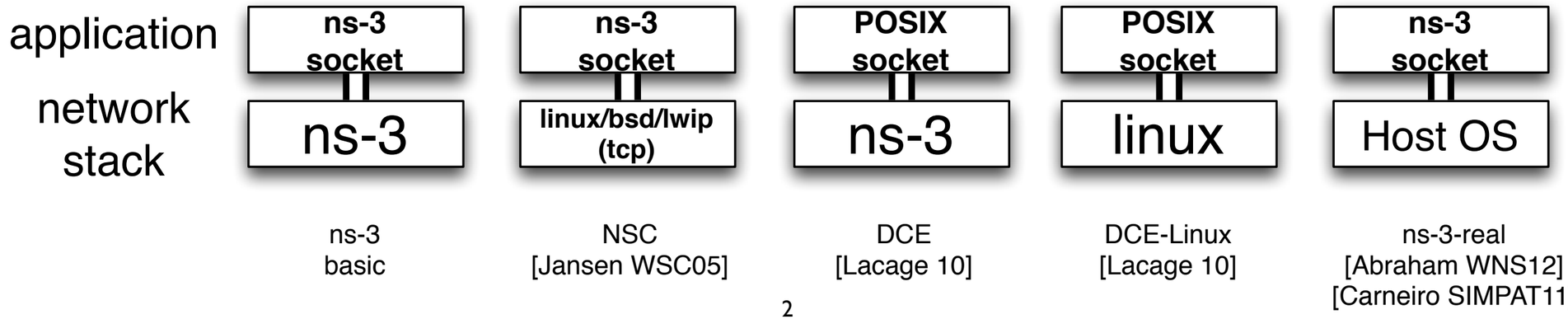
Simulate Network Protocols with Real Stacks for Better Realism

Hajime Tazaki (NICT, Japan)
Frédéric Urbani (INRIA, France)
Thierry Turletti (INRIA, France)

WNS3 2013, March, 2013, Cannes

Motivation

- ns-3 already has a wide range of network stacks
  - native ns-3 stack
  - FreeBSD/OpenBSD/lwip/Linux 2.6 (Network Simulation Cradle)
  - Direct Code Execution (Linux)
  - Using Host OS's stack
- Why current network stacks of ns-3 are not enough?



1) Adding new simulation models

- If we port SCTP or DCCP from other existing codes then,
  - Linux
    - 34K LOC (SCTP), 10K (DCCP)
  - ns-2.35
    - 20K LOC (SCTP), 9.5K (DCCP)
- Need much effort if we implement such models

2) Validation of new protocols

- Simulated protocols often only simplified model of a real one
- If protocol re-implemented scratch
  - need first to validate its implementation (complex task)
- Various misbehaviors of simulated models
  - Failure in neighbor discovery (IPv6) with specific option
  - Fixed size of TTL value
  - Different default values of simulated protocols

Using Direct Code Execution (DCE)

- A way to reuse running code
  - Linux network stack
- Without manual patching
  - Easy to track the latest version of the code

- Still ns-3 applications cannot use DCE
  - Only POSIX socket applications can benefit with DCE...

Outline

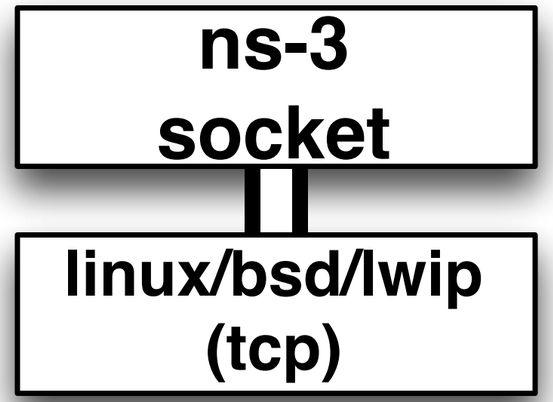
- Motivation
- Existing studies
- Design of DCE Cradle
- Evaluations
- Conclusion

How many kind of network stacks does ns-3 have ?

Network Simulation Cradle (NSC)

[Jansen WSC05]

- Utilize real TCP codes in network simulators (ns-2/ns-3)
- Various OSes
  - FreeBSD5, Linux-2.6, OpenBSD, lwip



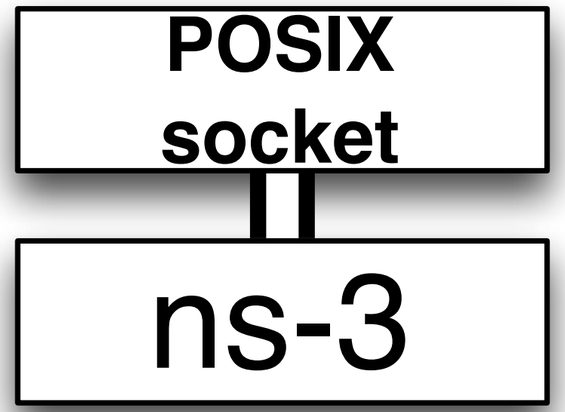
- Hard to extend (UDP, DCCP)
- Hard to track latest kernel

```
InternetStackHelper internet;
internet.SetTcp ("ns3::NscTcpL4Protocol",
                 "Library",
                 StringValue("liblinux2.6.26.so"));
PacketSinkHelper sink ("ns3::TcpSocketFactory",
  InetSocketAddress (Ipv4Address::GetAny (), port));
```

Direct Code Execution

[Lacage 10]

- POSIX socket applications run on ns-3 (without any modifications)
- Can utilize available network stacks in ns-3 (NSC, native)

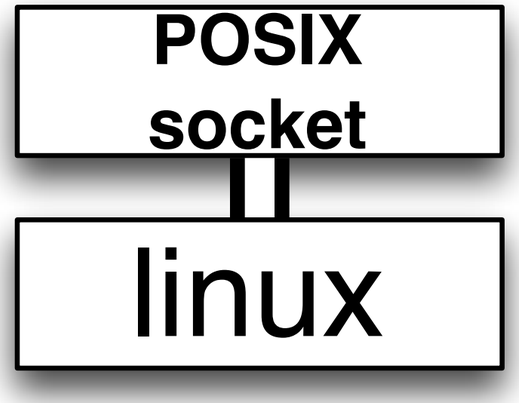



```
DceApplicationHelper dce;
dce.SetBinary ("tcp-loopback");
dce.Install (nodes.Get (0));
```

DCE Linux kernel module

[Lacage 10]

- Allow using Linux network stack
  - with DCE-enabled POSIX socket application
- Designed to track latest kernel
  - virtually introduced “sim” architecture in kernel
- It’s not available for ns-3 applications

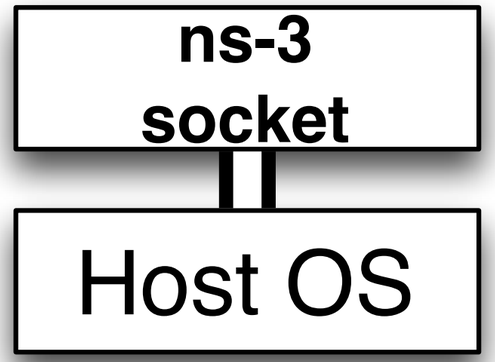



```
DceManagerHelper dceMng;
dceMng.SetNetworkStack ("ns3::LinuxSocketFdFactory",
                        "Library",
                        StringValue ("liblinux.so"));
dceMng.Install (nodes);
DceApplicationHelper dce;
dce.SetBinary ("tcp-loopback");
dce.Install (nodes.Get (0));
```

Using Host-OS's stack

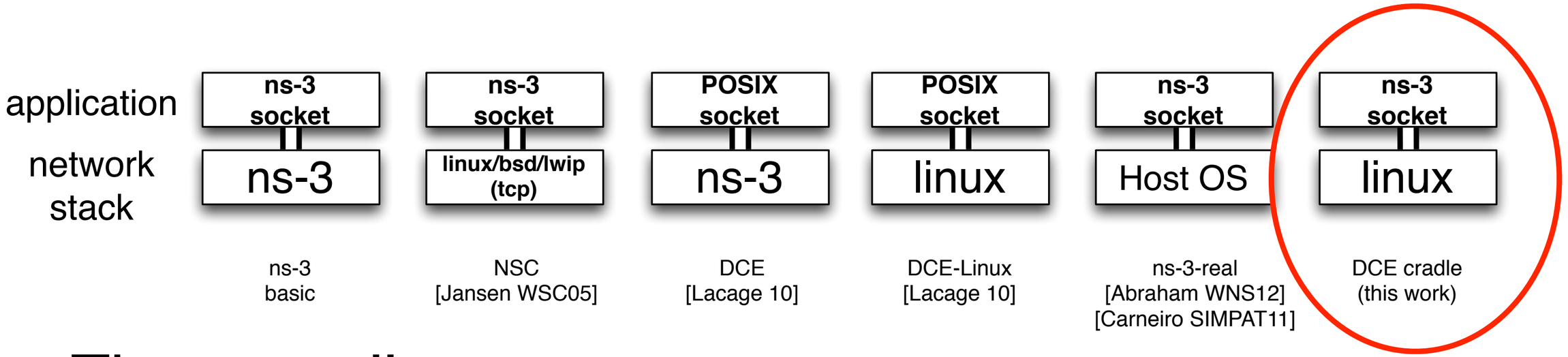
[Abraham WNS12][Carneiro SIMPAT11]

- Using hosted OS network stack from ns-3
- ns-3 applications can run
  - on ns-3 (as simulations)
  - on (directly) host OS
- Avoid writing application codes twice




```
PacketSinkHelper
("ns3 :: RealTcpSocketFactory" ,
InetSocketAddress (Ipv4Address::GetAny (), port));
```

What is possible today?



- There is still a gap
  - ns-3 applications (e.g., PacketSink/OnOff/Bulk application) + **latest** Linux kernel via DCE

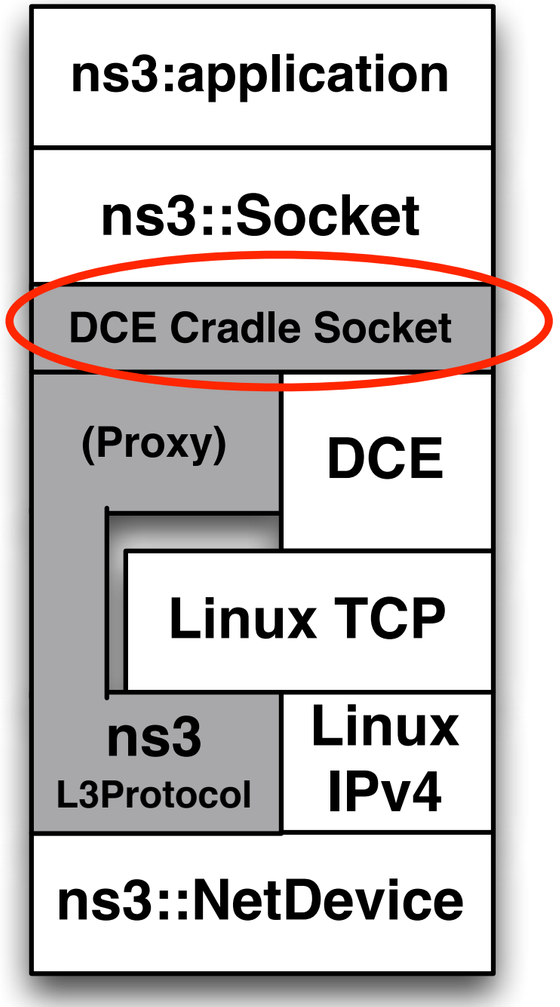
How to fill the gap?

- Goal of **DCE Cradle**
  - ns-3 applications + **latest** Linux kernel network stack
  - make ns-3 scenarios/applications **transparent**

- 3 Key components
  - Socket wrapper
  - Linux stack helper, Layer-3 proxy
  - DCE scheduler (blocking vs non-blocking)

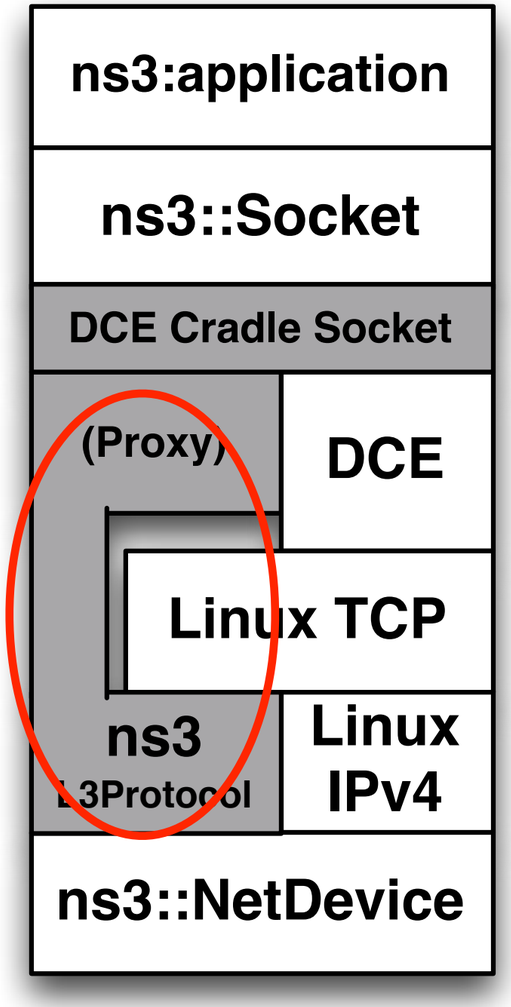
Socket wrapper

- A series of socket factories
  - ns3::LinuxIpv4RawSocketFactory
  - ns3::LinuxTcpSocketFactory
  - ns3::LinuxUdpSocketFactory
  - ns3::LinuxDccpSocketFactory
  - (can be extended)
- ns-3 applications should only care about the socket name



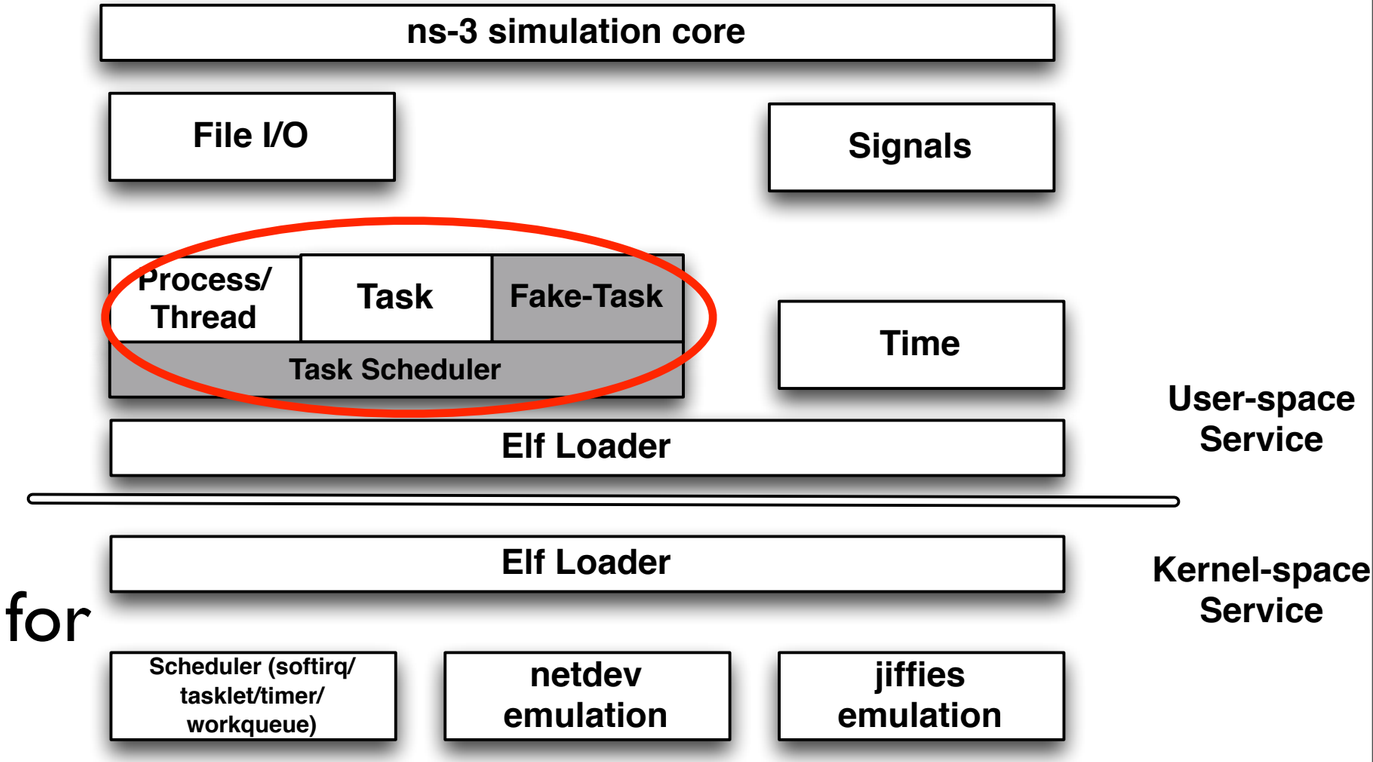
Layer-3 Proxy

- ns3::LinuxStackHelper class
  - Replacement of ns3::InternetStackHelper
  - ns3::Ipv4Linux class in the backend
    - to configure IP addresses and routes
- Useful to make the ns-3 applications transparent to network stacks



DCE core extension

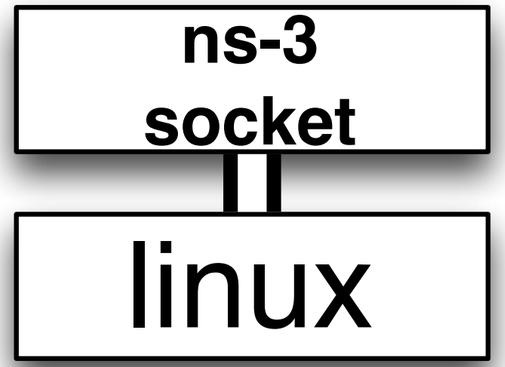
- Design consideration
  - ns-3 asynchronous socket API vs POSIX socket API
- Introduce Fake-task
  - won't ***schedule*** tasks for DCE Cradle operation
- Set (statically) ***O_NONBLOCK*** for DCE Cradle Linux sockets



Architecture of Direct Code Execution

How DCE Cradle looks like ?

- Specifying
  - library name of network stack (e.g., liblinux.so)
  - socket names of DCE Cradle (ns3::LinuxYYYYSocketFactory)
- That's it



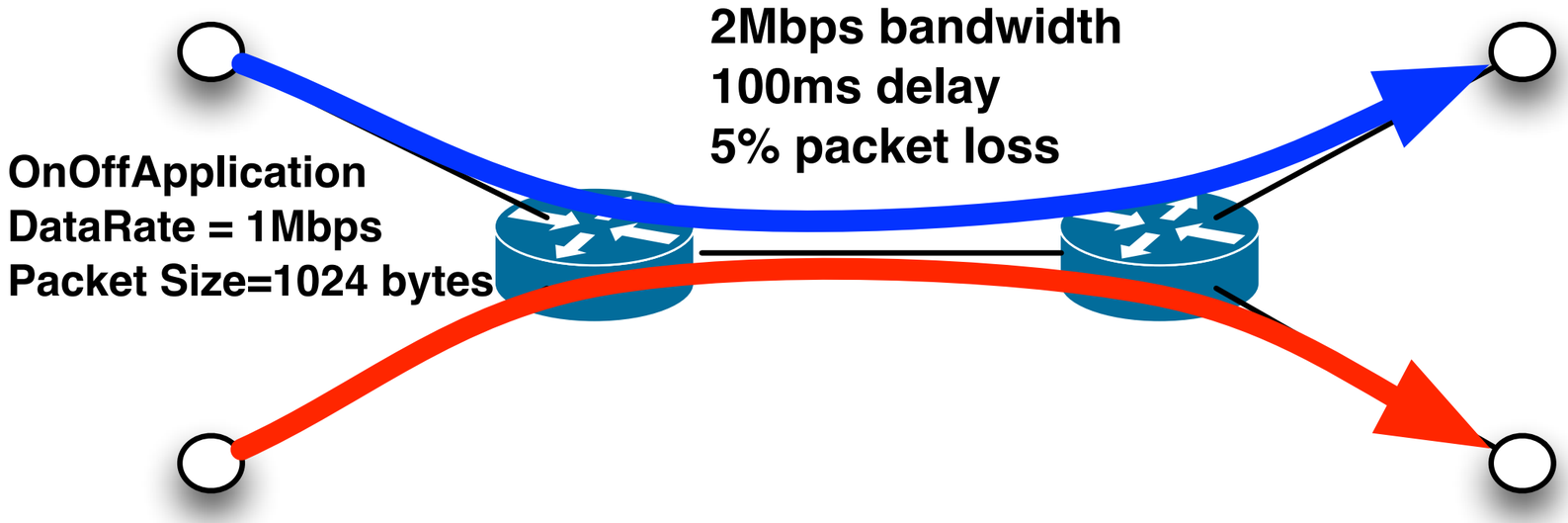

```
DceManagerHelper dceMng;
dceMng.SetNetworkStack ("ns3::LinuxSocketFdFactory",
                        "Library",  StringValue ("liblinux.so"));
dceMng.Install (nodes);

PacketSinkHelper sink = PacketSinkHelper
("ns3::LinuxIpv4RawSocketFactory",
     InetSocketAddress (Ipv4Address::GetAny (), 9));
```

Evaluation

- How **similar** DCE Cradle is to other approaches ?
- How much **overhead** does DCE Cradle introduce ?
- How is DCE Cradle able to **reduce** implementation cost ?

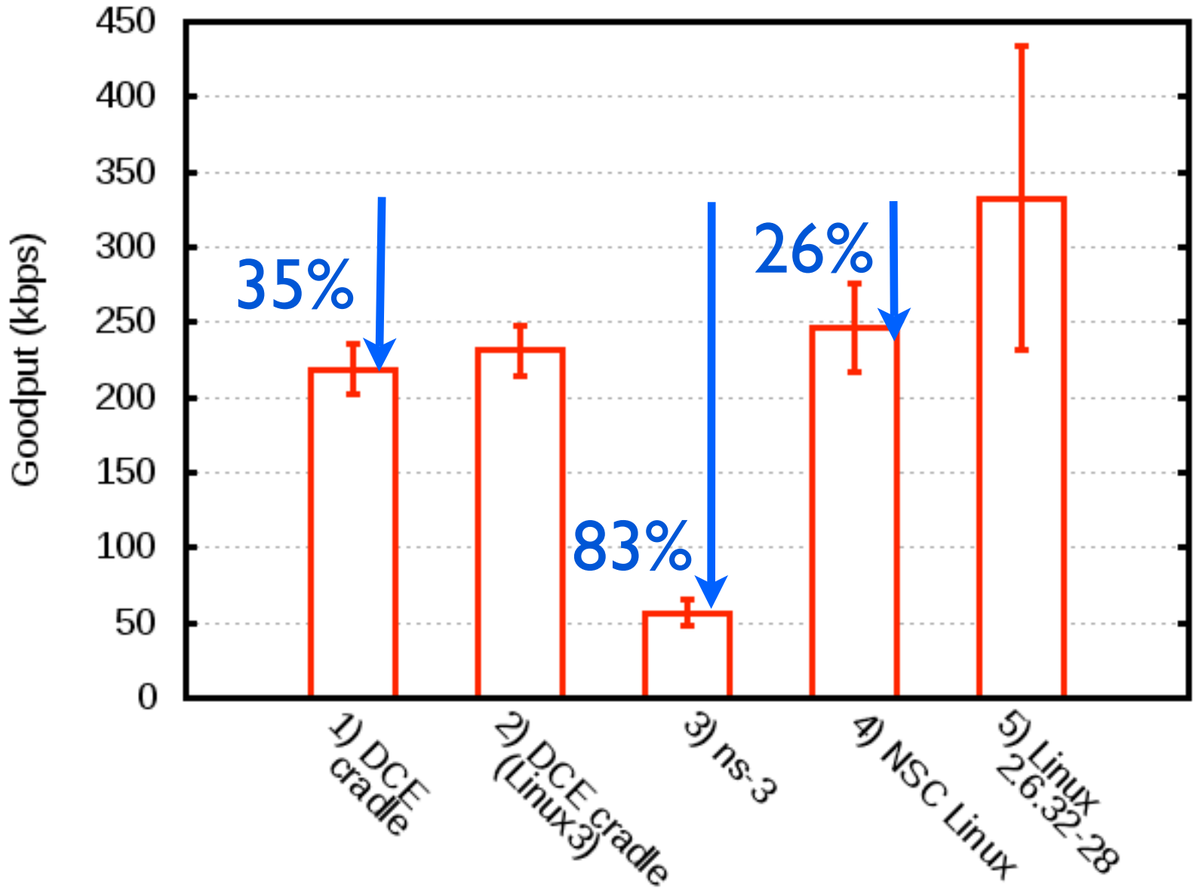
Network Configuration



- Simple dumbbell topology
  - Multiple traffic flows through a bottleneck link

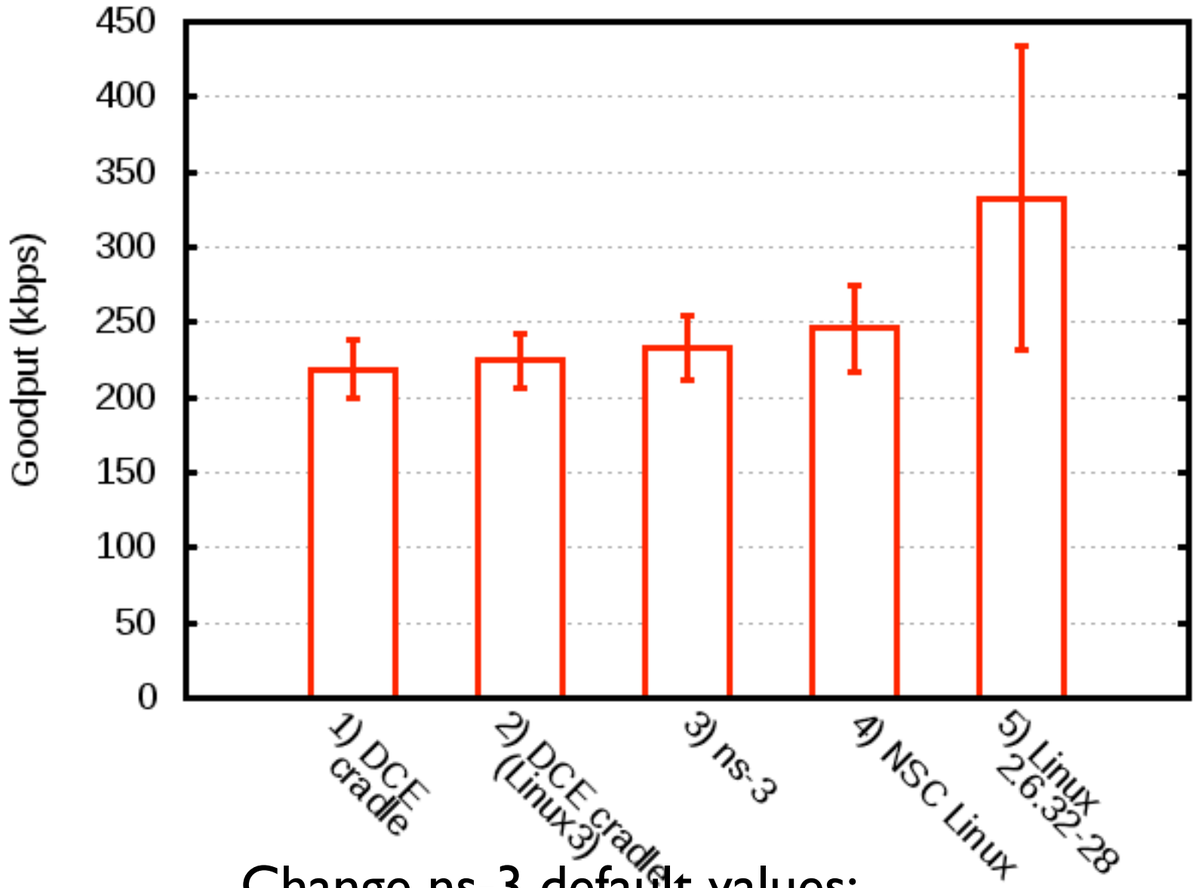
Goodput measurement

- Variants
  - 1) DCE Cradle (Linux 2.6.36)
  - 2) DCE Cradle (3.4.5)
  - 3) ns-3 native
  - 4) NSC (Linux 2.6.26)
  - 5) Real Linux (2.6.32-28)
- Traffic generator
  - OnOffApplication (1-4)
  - iperf (5)



Goodput measurement

- Variants
  - 1) DCE Cradle (Linux 2.6.36)
  - 2) DCE Cradle (3.4.5)
  - 3) ns-3 native
  - 4) NSC (Linux 2.6.26)
  - 5) Real Linux (2.6.32-28)
- Traffic generator
  - OnOffApplication (1-4)
  - iperf (5)

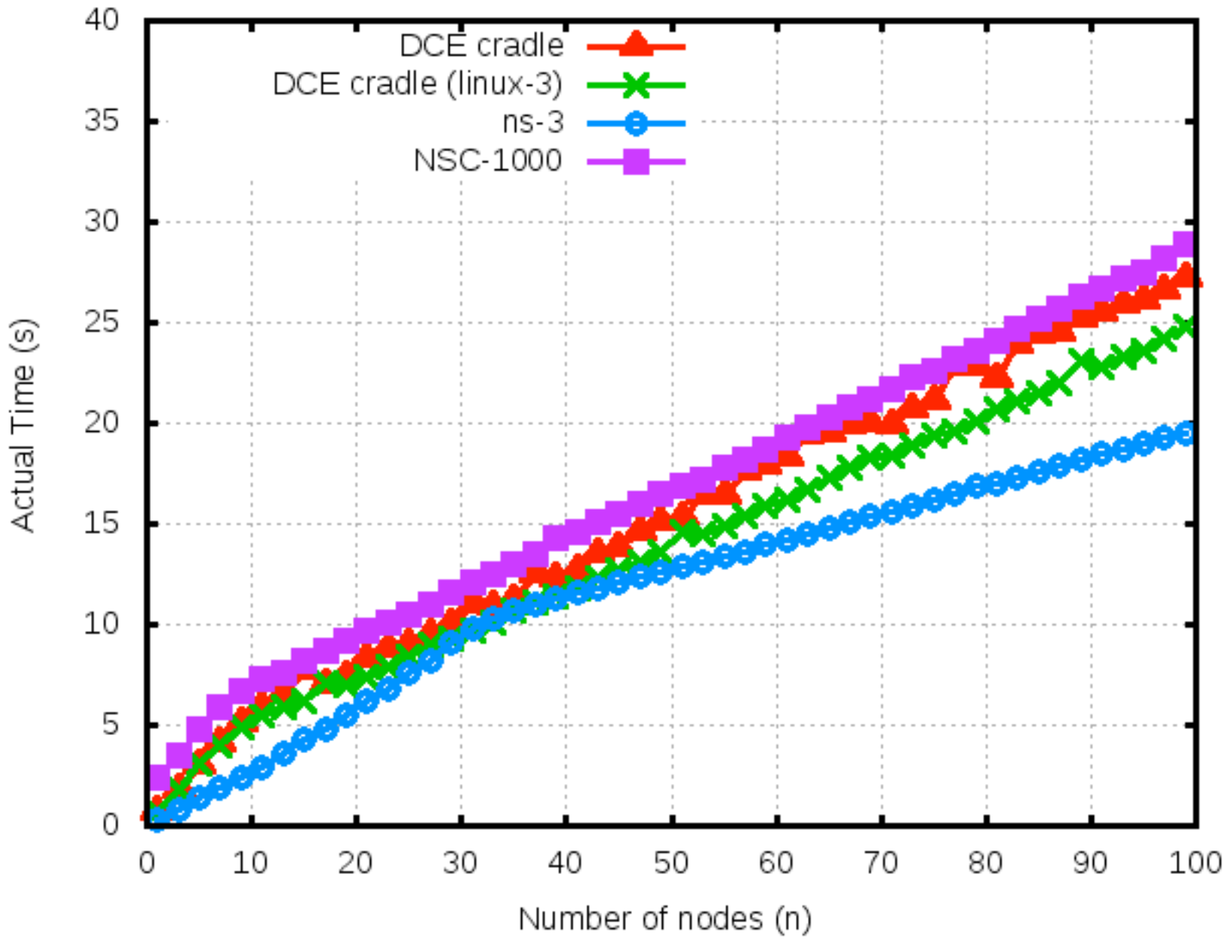


Change ns-3 default values:
TCP/SegmentSize (MSS) **576=>1448**
TCP/DelAckCount , **2=>1**

Overhead (vs #nodes)

- Varying the number of Tx/Rx nodes
- Fixing simulation time
  - 64 seconds
- Measure execution time

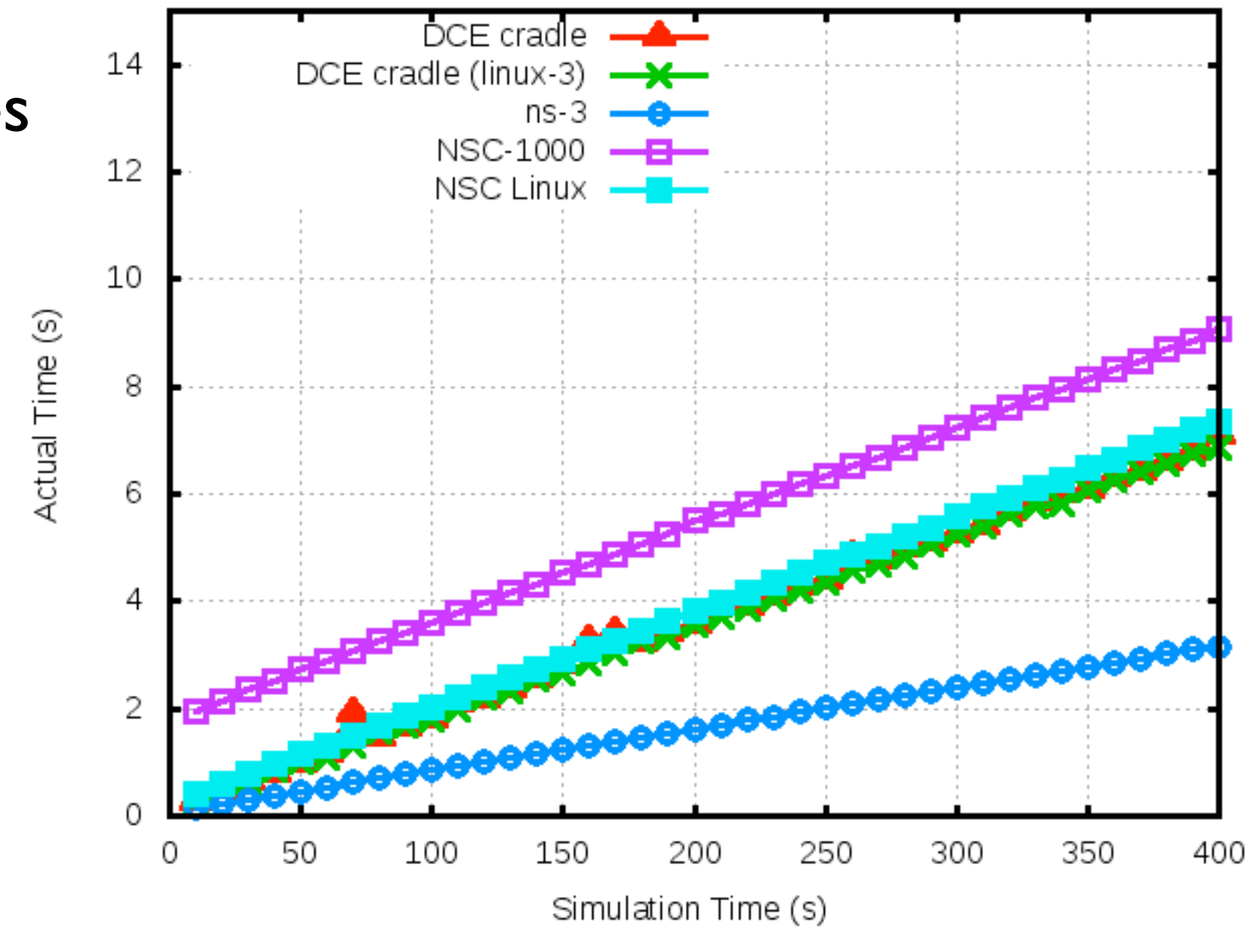
- NSC = (DCE Cradle * 1.05)
- ns-3 native is faster



Overhead (v.s. simulation time)

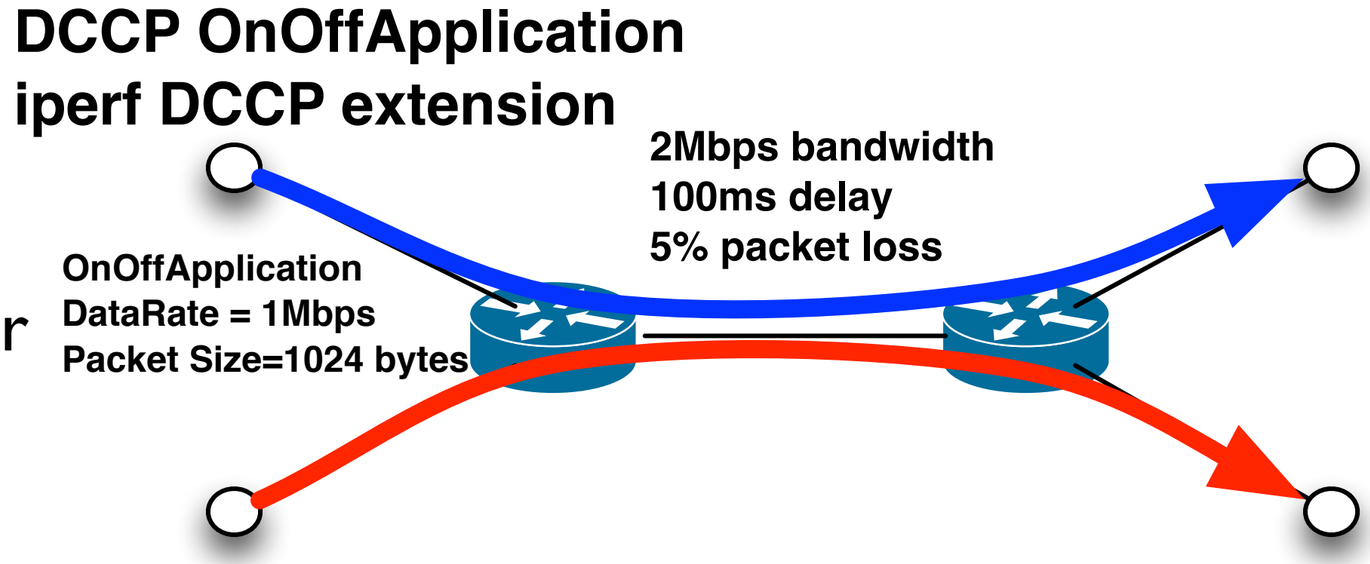
- Fixing the number of Tx/Rx nodes (2)
- Varying simulation time: 5 - 400 seconds
- Measure execution time

- NSC = (DCE Cradle * 1.3)
- ns-3 = (DCE Cradle / 2.2)



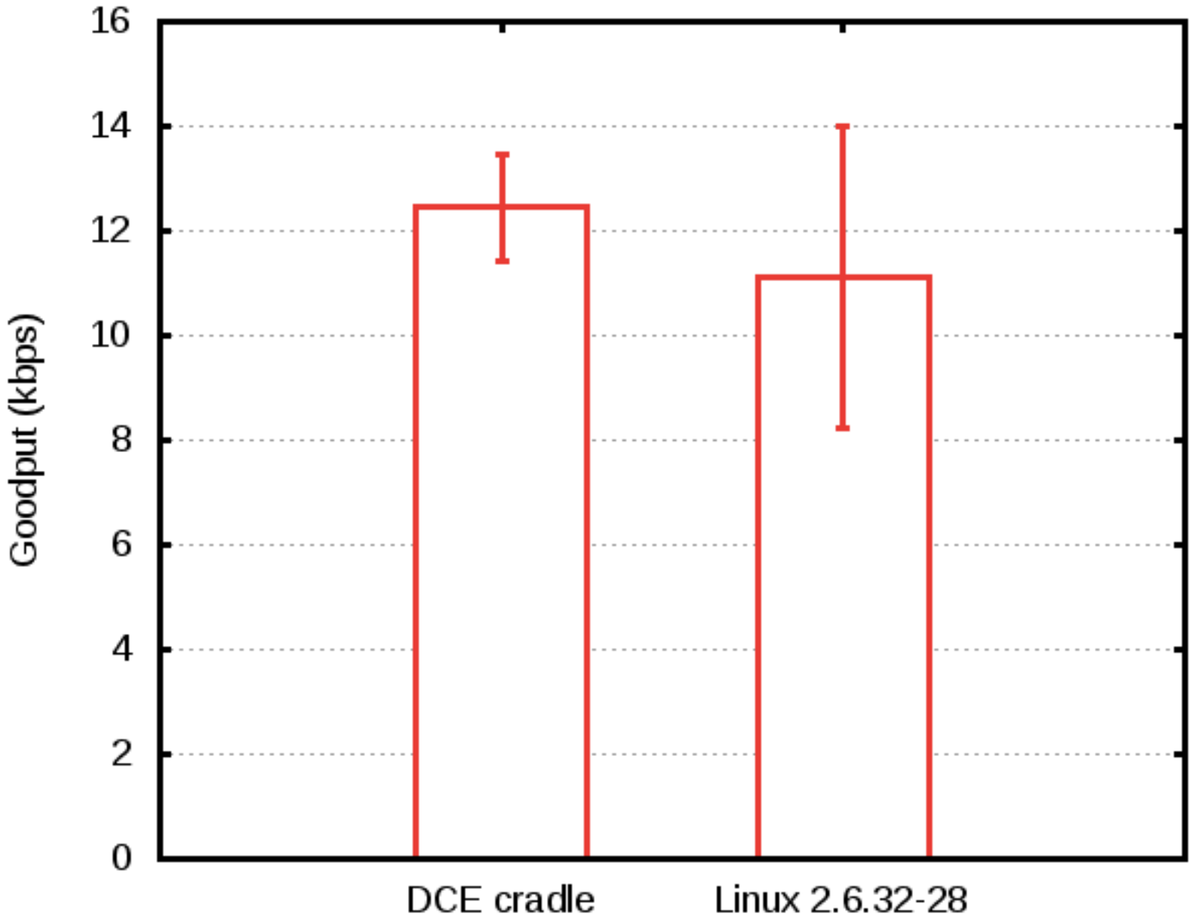
Linux DCCP with ns-3

- Build DCE enabled kernel
  - with CONFIG_NET_DCCP option
  - and a bit of glue code
- DCE Cradle provided wrapper socket (ns3::LinuxDccpSocketFactory)



DCCP simulation

- Difference
  - DCE Cradle/Linux 2.6.32 = 1.11
- Successfully simulate DCCP
- Similar behavior to real network code



Summary

- DCE is able to use with ns-3 applications
  - Linux TCP, as well as UDP and DCCP
- Performance overhead is negligible
- The code will be merged ns-3-dce soon

Thanks

Latest information will be at DCE Web page
http://www.nsnam.org/projects/direct-code-execution/

Lessons learnt

- Importance of reproducible result
  - Easily and quickly reproduce the results
  - All instructions described in the paper are available
    - http://www.nsnam.org/~thehajime/ns-3-dce-doc/dce-cradle-usecase.html
- Default values should carefully be chosen when comparing 2 implementations of a same protocol

Future plan

- Development version is available
  - will be announced soon
- 1st Release
  - IPv4 (UDP/TCP/DCCP/raw socket)
- 2nd
  - SCTP (depends on DCE feature)
  - IPv6

Latest information will be at DCE Web page

http://www.nsnam.org/projects/direct-code-execution/