ns-3 Direct Code Execution tutorial

Hajime Tazaki (University of Tokyo)

12th May, 2015

Disclaimer (excuse ?)

- This tutorial is based on dce-1.5 and ns-3.22, the contents would be changed as DCE right now has usability issues....
- Point here is *to smell the potentials of DCE*
- You can do whatever you want **IF** you devote certain amount of brain :)
- We will use the Live-CD ISO image, which includes ns-3.22 and dce-1.5 modules.

https://www.nsnam.org/wiki/AnnualTraining2015

Survey report

Q: Topic: What kind of topic are you interested in ?

Topic	count	ratio
TCP in general	4	66.7%
Routing protocols (e.g. quagga)	4	66.7%
Custom Linux kernel network stack	4	66.7%
FreeBSD kernel network stack	2	33.3%
Linux Multipath TCP	1	16.7%
Other	3	50%

Recipe

- DCE in a nutshell (10 mins)
- Showcases (10 mins)
- Hands-on (60 mins)
  - Quagga ospfd
  - Linux TCP congestion window trace
- Q&A (you can interrupt anytime)

Direct Code Execution in a nutshell

What is Direct Code Execution ?

- 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
  - Configuration different

Why DCE ?

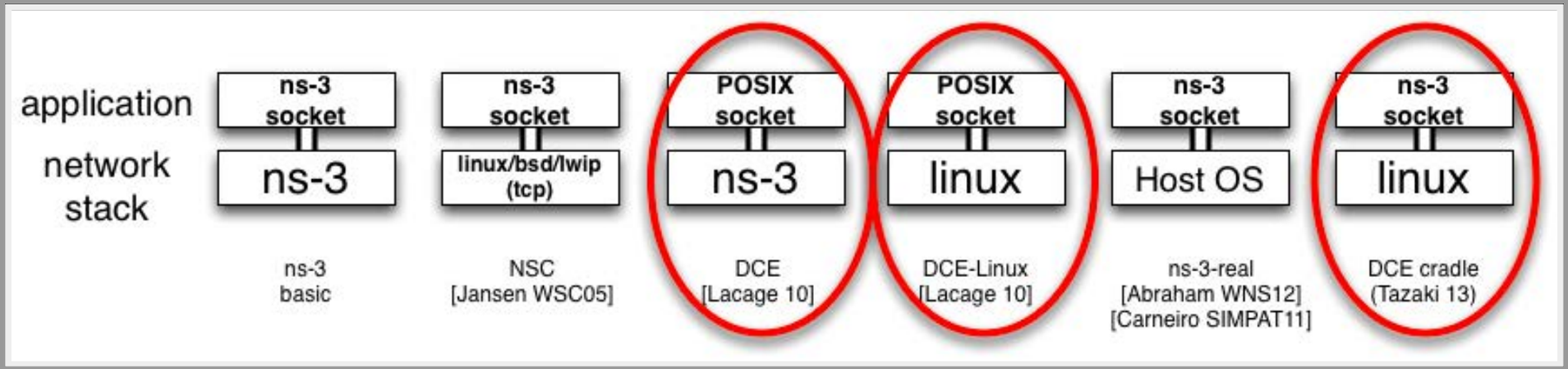
1. You want to investigate a protocol, but the **model isn't available**,
2. You don't want to **maintain two versions of implementation** btw/ ns-3 and (UNIX/POSIX) socket applications,
3. You want to evaluate a protocol implemented in Linux kernel, but
   - it **doesn't scale** much (computation resource of VM)
   - or you want to conduct a (fully) **reproducible experiment**

DCE helps you !

A brief history

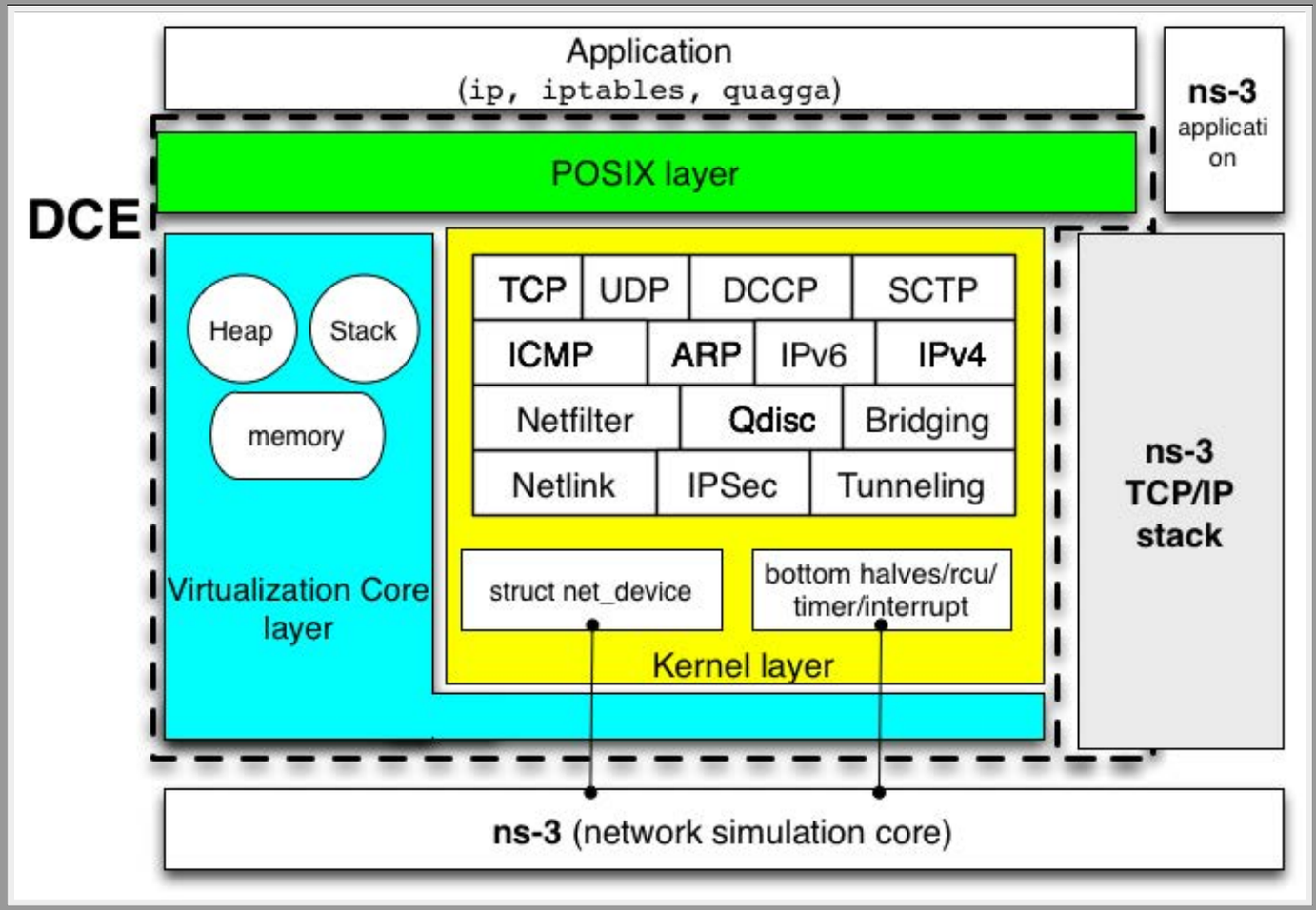
- Initial discussion (ns-3 goals)
  - Started around 2007 (Lacage)
  - GSoC 2008 (Quagga/Netlink bridge)
- almost 8 years old
  - 7 times official release since April 2013
  - along with ns-3 release (i.e., ns-3.23 == dce-1.6)

High-level Overview



- POSIX application on ns-3 [Lacage 10]
- Linux/FreeBSD kernel on ns-3 [Lacage 10]
- ns-3 Application with Linux/FreeBSD kernel [Tazaki 13]

Internals



- Virtualization Core Layer
- Kernel layer
- POSIX glue layer

How it works ?

- Recompile your code
  - Userspace as Position Independent Executable (PIE)
  - Kernel space code as shared library (libsim-linux.so)
- Run with ns-3
  - Load the executables (binary, library) in an isolated environment among nodes
  - synchronize simulation clocks with apps/kernels clock

What to start with ?

- Quick start with *bake* tool
  - https://www.nsnam.org/docs/dce/release/1.5/manual/html/getting-started.html
- 3 modes
  - dce-ns3-|version| (basic mode)
  - dce-linux-|version| (advanced mode)
  - dce-freebsd-|version| (*experimental*)

```
mkdir dce
cd dce
bake.py configure -e dce-linux-1.5
bake.py download
bake.py build
```

How to use it ?

- DceManagerHelper / DceApplicationHelper
- (option) LinuxStackHelper
- (option) other submodule helper (QuaggaHelper, Mip6dHelper, etc)
- (option) Custom command execution

Code snippet

```
// configure DCE with Linux network stack
DceManagerHelper dce;
dce.SetNetworkStack ("ns3::LinuxSocketFdFactory",
                     "Library", StringValue ("liblinux.so"));
dce.Install (nodes);

// run an executable at 1.0 second on node 0
DceApplicationHelper process;
ApplicationContainer apps;
process.SetBinary ("your-great-server");
apps = process.Install (nodes.Get (0));
apps.Start (Seconds (1.0));
```

Recent News

- release (1.6)
  - will be released very soon
- Linux upstream
  - posted lkml on March 2015, on the review
  - LWN featured (http://lwn.net/Articles/639333/)

Call for Maintainers

we want to have new maintainer

- maintain a release of DCE
- fix known bugs in Bugzilla
- introduce new features

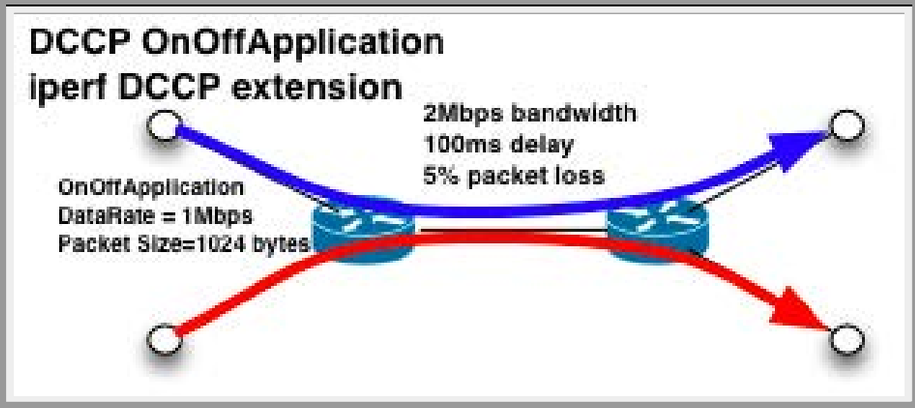
contact Hajime (thehajime at gmail.com) directly

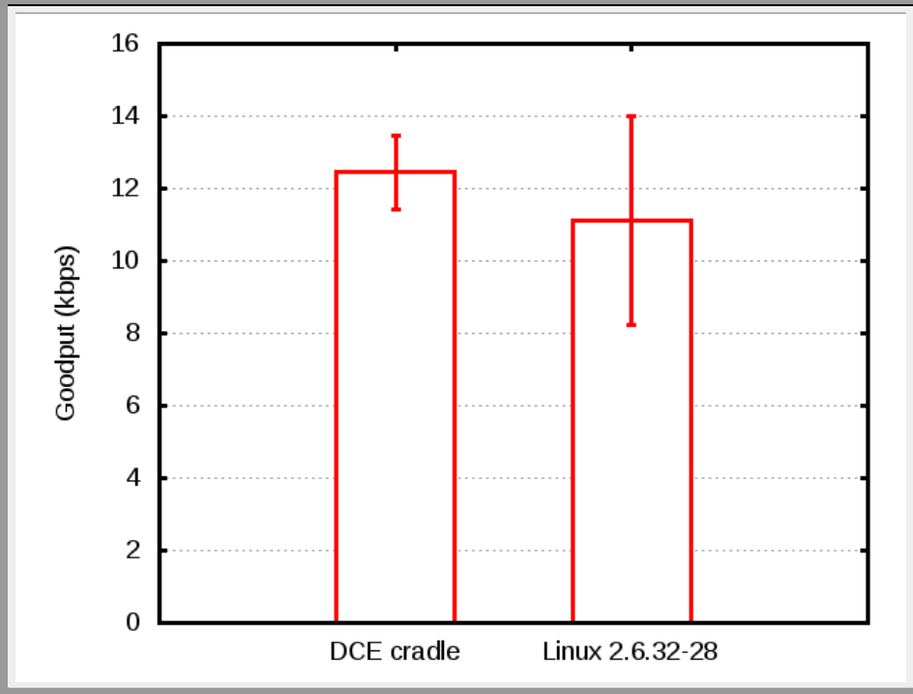
DCE Showcases

Features/Functions

1. **less development effort**
   - quagga, umip, ccnx, bind9, etc
   - Linux kernel (SCTP, DCCP, IPv4/v6), FreeBSD 10.0 kernel (partialy)
   - Out-of-tree Linux kernel Multi-path TCP
2. **reproducible environment** for Linux kernel experiment
3. **development platform**
   - debugging facility (gdb, valgrind)
   - code coverage (gcov, etc)

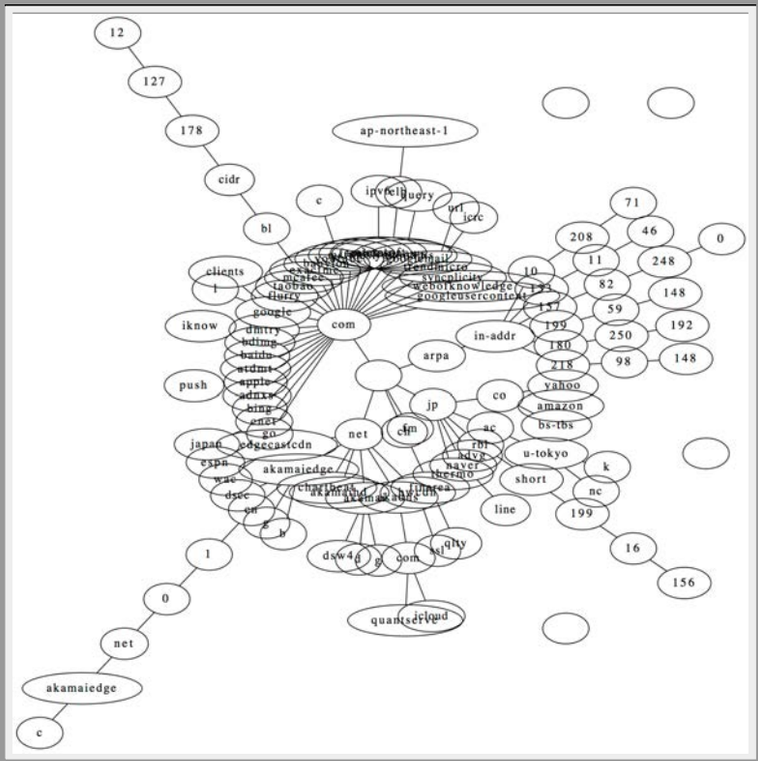
Less development effort (DCCP)

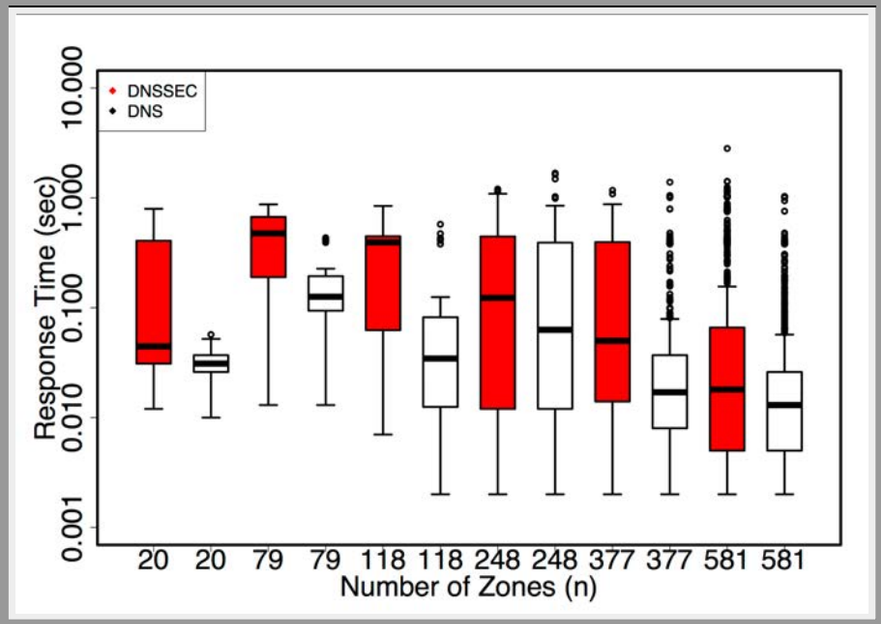




- **10K LoC** of Linux DCCP can be used on ns-3
- with a small amount of code (~10 LoC)!
- and the results are promising

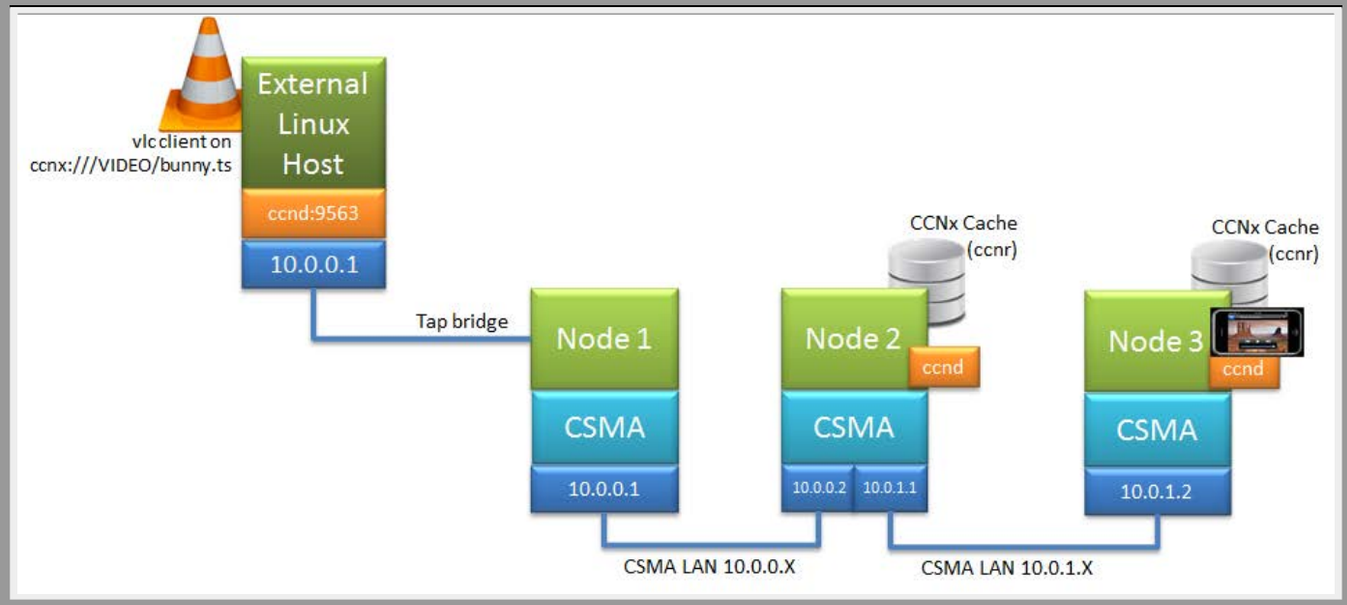
Less development effort (DNS/DNSSEC)





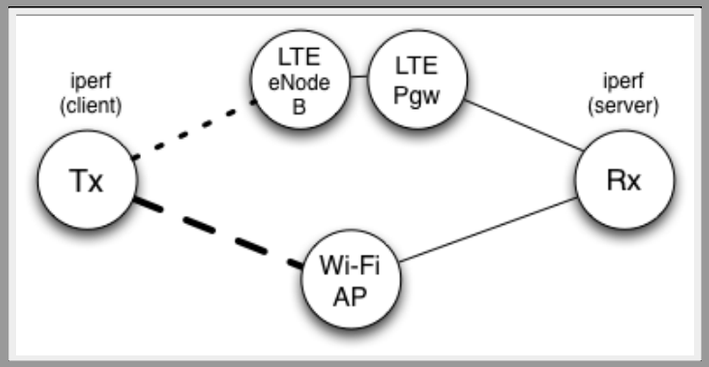
- bind9 zone file => named.conf/unbound.conf (by *createzones*)
- run bind9/unbound (*named*, *dig*, etc) with key-signed operation
- see how response time will be changed
- available at a private repository (http://dnssec.sekiya-lab.info)

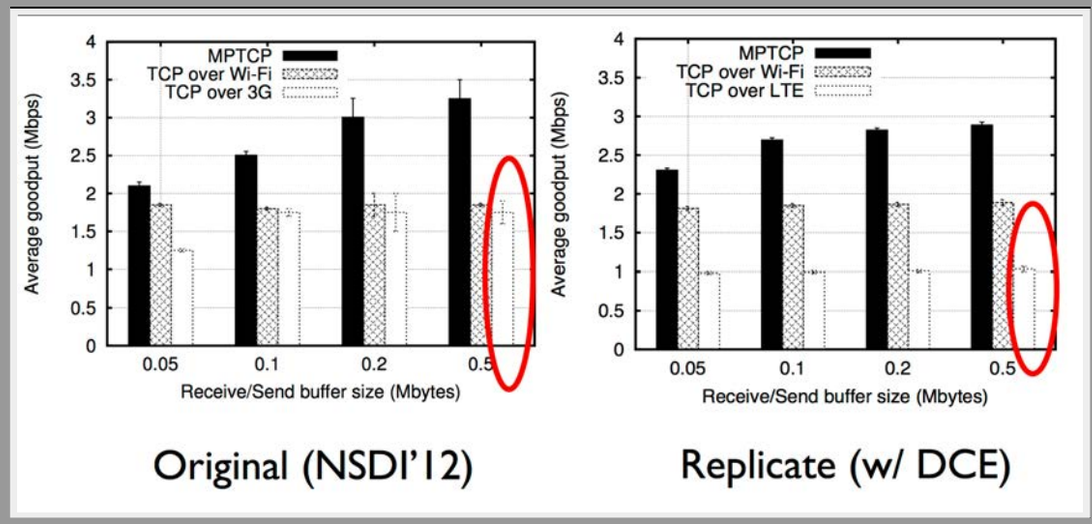
Less development effort (ccnx)



- an alternative for ndn-sim

Replication of a past experiment (MPTCP)



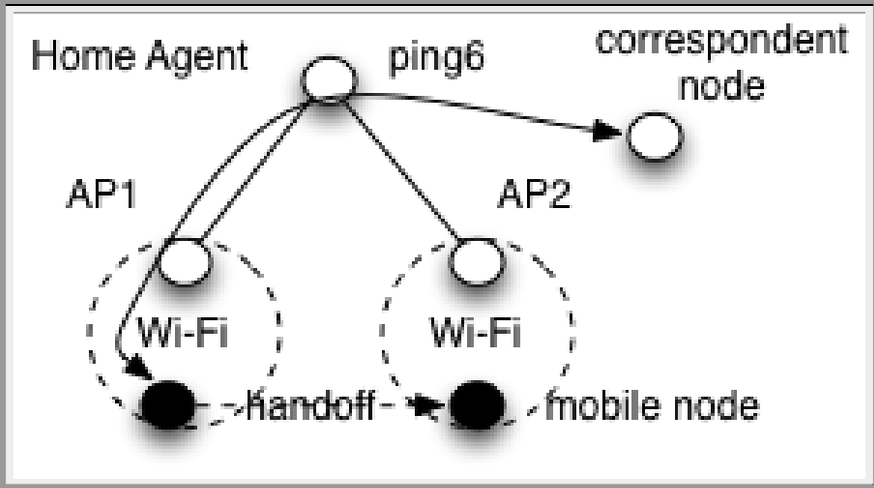


1. picked one figure from a paper (NSDI 2012)
2. replicate an experiment with available information on the paper
3. configure an ns-3 scenario **with** the same software (Linux/iperf)

- no significant goodput improvement with buffer size when DCE in single TCP
- Max goodput range: 2.2 - 2.9Mbps (DCE) 2.0 - 3.2Mbps (NSDI)

See more detail in our CoNEXT paper (2013) [2]

Development platform (gdb)




```
(gdb) b mip6_mh_filter if dce_debug_nodeid()==0
Breakpoint 1 at 0x7ffff287c569: file net/ipv6/mip6.c, line 88.
<continue>
(gdb) bt 4
#0  mip6_mh_filter
    (sk=0x7ffff7f69e10, skb=0x7ffff7cde8b0)
    at net/ipv6/mip6.c:109
#1  0x00007ffff2831418 in ipv6_raw_deliver
    (skb=0x7ffff7cde8b0, nexthdr=135)
    at net/ipv6/raw.c:199
#2  0x00007ffff2831697 in raw6_local_deliver
    (skb=0x7ffff7cde8b0, nexthdr=135)
    at net/ipv6/raw.c:232
#3  0x00007ffff27e6068 in ip6_input_finish
    (skb=0x7ffff7cde8b0)
    at net/ipv6/ip6_input.c:197
```

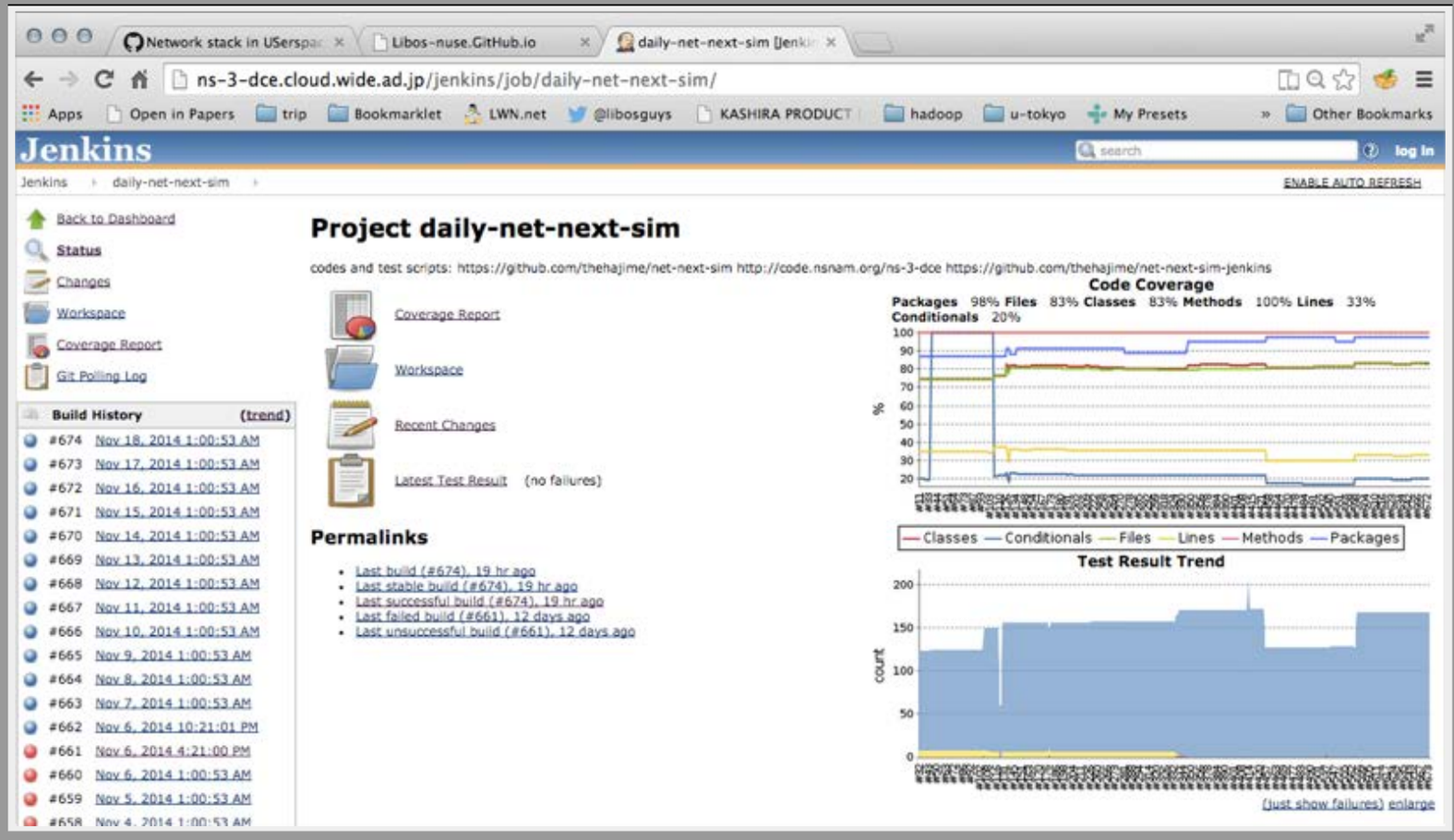
- Inspect codes during experiments
  - among distributed nodes
  - in a single process
- perform a simulation to reproduce a bug
- see how badly handling a packets in Linux kernel

Development platform (valgrind)

```
==5864== Memcheck, a memory error detector
==5864== Copyright (C) 2002-2009, and GNU GPL'd, by Julian Seward et al.
==5864== Using Valgrind-3.6.0.SVN and LibVEX; rerun with -h for copyright info
==5864== Command: ../build/bin/ns3test-dce-vdl --verbose
==5864==
==5864== Conditional jump or move depends on uninitialised value(s)
==5864==    at 0x7D5AE32: tcp_parse_options (tcp_input.c:3782)
==5864==    by 0x7D65DCB: tcp_check_req (tcp_minisocks.c:532)
==5864==    by 0x7D63B09: tcp_v4_hnd_req (tcp_ipv4.c:1496)
==5864==    by 0x7D63CB4: tcp_v4_do_rcv (tcp_ipv4.c:1576)
==5864==    by 0x7D6439C: tcp_v4_rcv (tcp_ipv4.c:1696)
==5864==    by 0x7D447CC: ip_local_deliver_finish (ip_input.c:226)
==5864==    by 0x7D442E4: ip_rcv_finish (dst.h:318)
==5864==    by 0x7D2313F: process_backlog (dev.c:3368)
==5864==    by 0x7D23455: net_rx_action (dev.c:3526)
==5864==    by 0x7CF2477: do_softirq (softirq.c:65)
==5864==    by 0x7CF2544: softirq_task_function (softirq.c:21)
==5864==    by 0x4FA2BE1: ns3::TaskManager::Trampoline(void*) (task-manager.cc:261)
==5864==  Uninitialised value was created by a stack allocation
==5864==    at 0x7D65B30: tcp_check_req (tcp_minisocks.c:522)
==5864==
```

- Memory error detection
  - among distributed nodes
  - in a single process
- Use **Valgrind**

Development platform (Jenkins CI)

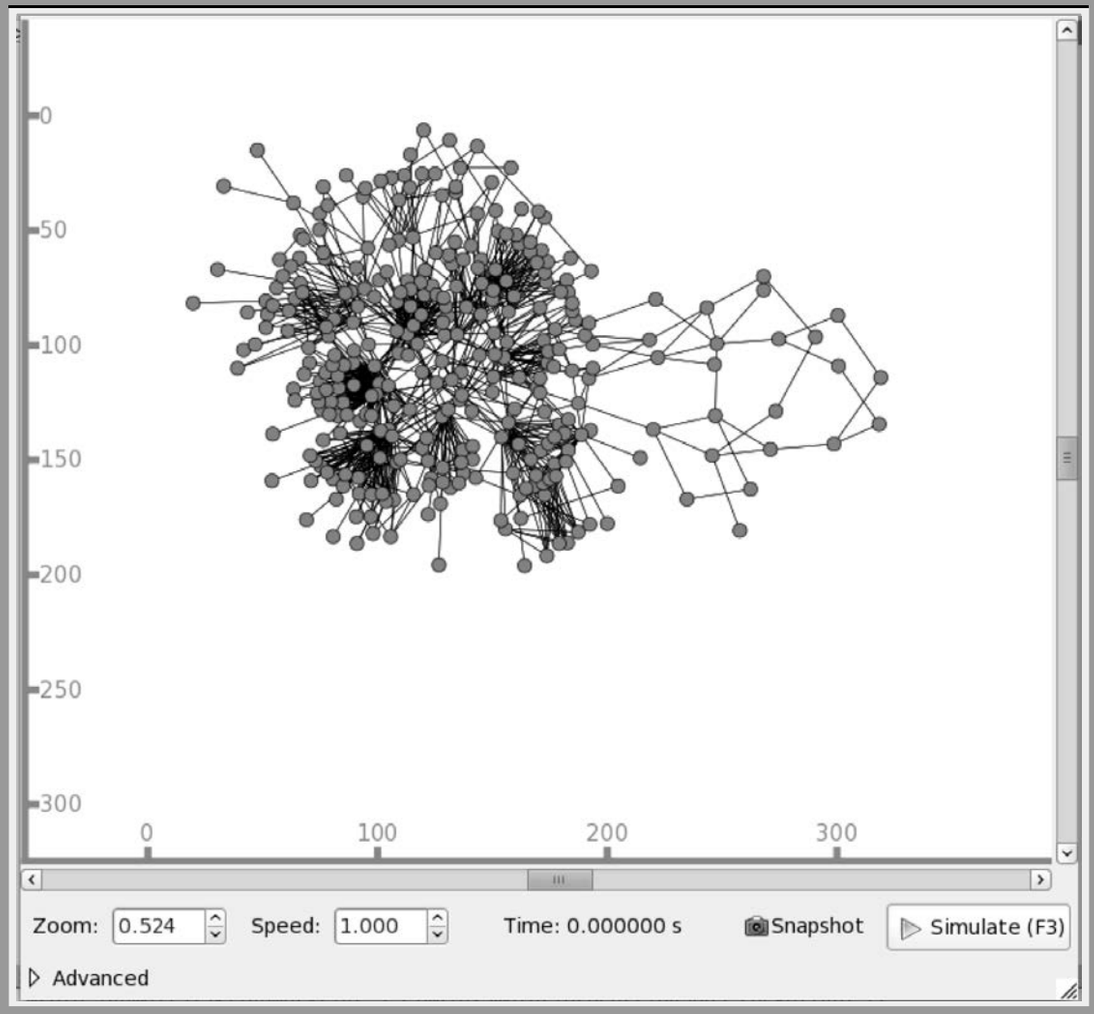


Hands-on 1

OSPF routing in a real ISP topology

Hands-on 1 (30 mins)

ospfd on rocketfuel topology



Workflow

Real topology, real routing protocol

- dce-quagga-ospfd-rocketfuel.cc
- Topology: Rocketfuel topology, AS3967, 79 nodes
- Routing: OSPFv2 (quagga ospfd), all area 0
- Traffic: ns3::V4Ping

Script review

```
gedit ~/training/bake/source/ns-3-dce/myscripts/\
ns-3-dce-quagga/example/dce-quagga-ospfd-rocketfuel.cc
```

http://code.nsnam.org/thehajime/ns-3-dce-quagga/file/eafaa128a2fe/example/dce-quagga-ospfd-rocketfuel.cc

Run simulation

```
cd training/bake/source/ns-3-dce
./waf --run dce-quagga-ospfd-rocketfuel --vis
```

You should see the result something like below.

```
scanning topology: 79 nodes...
scanning topology: calling graphviz layout
scanning topology: all done.
PING  10.0.2.18.26 56(84) bytes of data.
64 bytes from 10.0.2.18: icmp_seq=32 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=33 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=34 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=35 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=36 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=37 ttl=63 time=8 ms
64 bytes from 10.0.2.18: icmp_seq=38 ttl=63 time=8 ms
```

What happened ?

Look at the output files generated by *simulated* processes

```
ls files-0/var/log/(pid)/
```

- under files-X/ directories
  - cmdline: executed command with the arguments
  - stdout: log output to *stdout* by the process
  - stderr: log output to *stderr* by the process
  - status: internal status information
  - syslog: (optional) syslog info if the process uses

```
cat files-0/var/log/*/cmdline
```

Speedup the simulation

- Use custom *elf-loader*, instead of the system's one
- pass **--dlm** option to the waf

```
./waf --run ABC --dlm
```

- You should get

Loader+Fiber	**Time (MM:SS.ss)**
Cooja (non-vdl) + Pthread (default)	11:49.81
Dlm (vdl) + Ucontext (with --dlm)	2:29.21

Customize the script

- Change the topology file

```
head -8 ~/training/bake/source/ns-3-dce/myscripts/ns-3-dce-quagga/\
example/3967.weights.intra > topo8.dat
```

- Then run the simulation again

```
cd training/bake/source/ns-3-dce
./waf --run "dce-quagga-ospfd-rocketfuel --topoFile=topo8.dat" --vis
```

Summary

- Quagga ospfd with QuaggaHelper
- Import a Rocketfuel topology (AS3967)
- Verify the connectivity with ns3::V4Ping

Further steps

- Use distributed simulator (MPI) for the speedup
- Larger topology files

Hands-on 2

TCP congestion window trace with Linux kernel

Hands-on 2 (30 mins)

- Customize DCE environment from bake installation
- **TCP cwnd** trace with Linux kernel
  - latest Linux (Linux 4.1.0-rc1)
  - ss (in iproute2)
  - an example of DCE simulation *extending* pre-installed modules

Setup (Installation)

- get the latest net-next-sim and iproute2 code
- *iproute2* at least requires Linux 3.18.0 or later

```
git clone --depth=1 \
 https://github.com/direct-code-execution/net-next-sim -b wns3-2015
git clone --depth=1 \
 https://git.kernel.org/pub/scm/linux/kernel/git/shemminger/iproute2.git
```

- obtain a script

```
wget https://gist.githubusercontent.com/thehajime/5e9e05ea2df08141ae47/raw
a23387aa97b58bc13ccf96a1a208ddd387c9646f/nat-dce-tcp-ns3-nsc-comparison.cc
mv nat-dce-tcp-ns3-nsc-comparison.cc ns-3-dce/myscripts
```

you will see nat-dce-tcp-ns3-nsc-comparison.cc

build net-next-sim

```
cd net-next-sim
make defconfig ARCH=lib
make library ARCH=lib
cp libsim-linux-4.1.0-rc1.so ~/training/bake/build/bin_dce/
cd ~/training/bake/build/bin_dce/
ln -s -f libsim-linux-4.1.0-rc1.so liblinux.so
```

*1 ARCH=lib has been changed: we used ARCH=sim in the past.

build iproute2

```
cd iproute2
./configure
CFLAGS+=-fpic CFLAGS+=" -D_GNU_SOURCE" CFLAGS+=" -O0" CFLAGS+="\
 -U_FORTIFY_SOURCE" CFLAGS+=" -g" LDFLAGS=-pie \
 LDFLAGS+=" -rdynamic" make
cp misc/ss ~/training/bake/build/sbin/
```

- the latest iproute2 (newer than 3.18.0) has full TCP_INFO support *via netlink*

Re-build DCE

DCE requires to rebuild to adapt newer API of net-next-sim.

```
cd ns-3-dce
hg pull -u
./waf configure --prefix=/home/ns3/training/bake/build \
 --with-ns3=/home/ns3/training/bake/build \
 --with-elf-loader=/home/ns3/training/bake/build/lib \
 --with-libaspect=/home/ns3/training/bake/build \
 --enable-kernel-stack=/mnt/net-next-sim/arch
./waf
```

- you need to update the latest ns-3-dce (dce-1.6)
  - to run with the latest Linux kernel with DCE patch

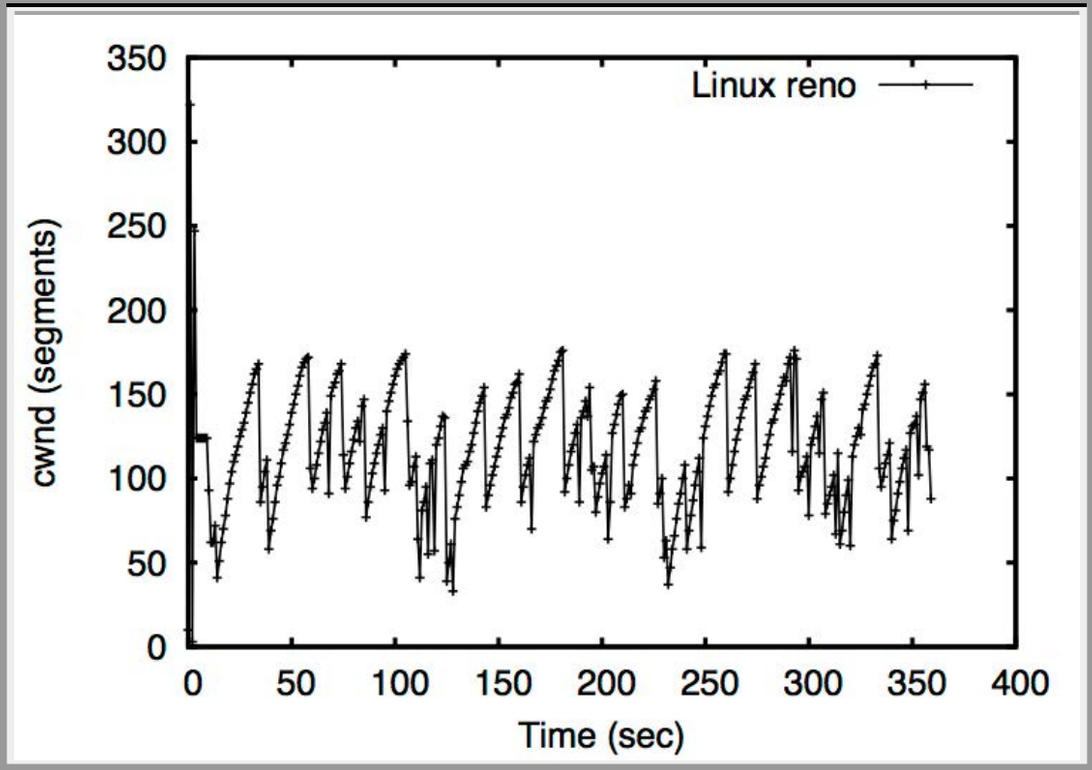
Run script

```
rm -rf files-*
./waf --run "nat-dce-tcp-ns3-nsc-comparison \
  --transport_prot=TcpNewReno --error_p=0.001"
```

- The simulation generates TCP traffic (via LinuxSocketFdFactory: DCE Cradle)
- 10Mbps, 45msec delay on PointToPointNetdevice

Generate a plot

```
grep cwnd files-0/var/log/*/stdout | sed "s/.*\(cwnd:.*\s\)ssth.*/\1/" | \
   sed "s/.*\(cwnd:.*\s\)send.*/\1/" | \
   sed "s/.*\(cwnd:.*\s\)lastsnd.*/\1/" | \
   sed "s/cwnd://"  > /tmp/cwnd.dat
gnuplot
gnuplot> plot "/tmp/cwnd.dat" using 0:($1*1446) \
        w lp title "Linux reno"
```



Voila !

Summary

- Congestion window trace with Linux TCP stack
- How to customize DCE which was installed by bake
- this contents will be included in future DCE release

The end of tutorial

- Walk-through tutorial of Direct Code Execution
- Documents, papers, examples
- Hands-on (Quagga ospfd, Linux TCP)

End notes

- Be patient :)
  - Real-world program is not always familiar with ns-3
  - not integrated with (ns-3) script/helper
  - sparse documents/tools/output
  - but you will get certain benefits !

References

- [1] Lacage. Experimentation Tools for Networking Research. PhD thesis, 2010
- [2] Tazaki et al. Direct Code Execution: Revisiting Library OS Architecture for Reproducible Network Experiments, CoNEXT 2013
- [3] Tazaki et al. DCE Cradle: Simulate Network Protocols with Real Stacks for Better Realism, WNS3 2013
- [4] Camara et al. DCE: Test the Real Code of Your Protocols and Applications over Simulated Networks, IEEE Communications Magazine, March 2014
- [5] Sekiya et al. DNSSEC simulator for realistic estimation of deployment impacts. IEICE ComEX(3):10, 2014.

Contact

- Web (https://www.nsnam.org/overview/projects/direct-code-execution/)
- Mailing list (ns-3-users@googlegroups.com)
- Github (https://github.com/direct-code-execution)

Questions ?