Direct Code Execution

Mathieu Lacage, Hajime Tazaki

March, 4th 2013
Simulations are great but...

Need Model Implementation
  It’s costly
  Are they correct?
Useless for Real Implementation
  Debugging
  Valgrinding
  Correctness testing
  Regressions testing
  Fuzz testing
  Code coverage
  etc.
Reuse model as the real implementation
It’s rare to have a model only
Somewhat lacks runtime efficiency
Reuse real implementation as the model
Painful Manual modifications
Synchronization with changes
Many (sucky) solutions
  Deployments with testbeds (PlanetLab, cluster)
  Emulation with VMs, containers
  Synchronized emulation with Xen
But, really, it’s painful...
  Reproducibility
  Setup complexity (NEPI ?)
  Complex debugging, tracing
What is Direct Code Execution?

Recompile

Userspace as Position Independent Executable
Kernelspace as shared library

Run within ns-3

Simulation models for layers 1/2 and/or 3/4/5
Userspace with libc & pthread replacements
Kernelspace with kernel services replacements

Debug with gdb, valgrind!
What it works with

quagga (RIPv2/ng, OSPFv2/3, BGP)
umip (Mobile IPv6)
ccnx (CCN)
libtorrent rasterbar
thttpd (http server)
bind9, unbound (DNS/DNSSEC)
iperf, ping, ping6
net-next (DCCP, TCP, IPv6/4)
What you can use it for

A development tool
  Easy distributed debugging
  Easy distributed valgrinding
  Easy distributed reproducible testing

A simulation tool
  Closer to the real implementations
  No need to design/implement/test a model
DCE as a development tool

DCE as a simulation tool
Typical development tasks:
- Debug our kernel code
- Valgrind our kernel code
- Setup regression tests
- Setup fuzz testing (regression tests with trinity)
- Track test coverage
Distributed debugging within a single process

(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  0x00007fff2831418 in ipv6_raw_deliver (skb=0x7ffff7cde8b0, nexthdr=135) at net/ipv6/raw.c:199
#2  0x00007fff2831697 in raw6_local_deliver (skb=0x7ffff7cde8b0, nexthdr=135) at net/ipv6/raw.c:232
#3  0x00007fff27e6068 in ip6_input_finish (skb=0x7ffff7cde8b0) at net/ipv6/ip6_input.c:197
(More stack frames follow...)
Just run it, and...

tcp_input.c:3782: touch un-initialized value
af_key.c:2143: touch un-initialized value

Still exists in 3.7.0
Regression testing

For example, bug\(^1\) introduced in Kernel 3.3

Table: Regression test results vs. kernel versions.

<table>
<thead>
<tr>
<th>Test Suite</th>
<th>Linux 2.6.34</th>
<th>Linux 3.4.0</th>
<th>Linux 3.7.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>test-raw-socket</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-tcp-socket</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-radvd (icmp6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-ripd (udp)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-ripngd (udp6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-bgpd (tcp)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-bgpd+ (tcp6)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>test-cmip6 (mip6)</td>
<td>FAIL</td>
<td>FAIL</td>
<td></td>
</tr>
<tr>
<td>test-nemo (nemo)</td>
<td>FAIL</td>
<td>FAIL</td>
<td></td>
</tr>
</tbody>
</table>

\(^1\)http://www.wakoond.hu/2012/07/message-corruption-with-hao-and-route2.html
Code coverage (gcov+lcov) is easier:
  Reproducible
  Sender & Receiver
We get higher coverage:

Table: Coverage of network test with DCE in Linux 3.7.0.

<table>
<thead>
<tr>
<th>Directory</th>
<th>Coverage</th>
<th>Functions</th>
<th>Branches</th>
</tr>
</thead>
<tbody>
<tr>
<td>net/core</td>
<td>31.8% (+9.2%)</td>
<td>38.2% (+12.3%)</td>
<td>22.1% (+7.5%)</td>
</tr>
<tr>
<td>net/ipv4</td>
<td>38.2% (+4.5%)</td>
<td>47.6% (+6.3%)</td>
<td>27.2% (+5.2%)</td>
</tr>
<tr>
<td>net/ipv6</td>
<td>41.1% (+32.8%)</td>
<td>51.9% (+39.5%)</td>
<td>29.9% (+25.0%)</td>
</tr>
<tr>
<td>net/netlink</td>
<td>55.7% (+24.1%)</td>
<td>68.3% (+30.1%)</td>
<td>40.5% (+25.6%)</td>
</tr>
<tr>
<td>net/packet</td>
<td>13.4% (+11.8%)</td>
<td>18.4% (+15.4%)</td>
<td>7.8% (+6.9%)</td>
</tr>
<tr>
<td>net/xfrm</td>
<td>36.4% (+36.0%)</td>
<td>48.2% (+47.9%)</td>
<td>25.3% (+25.0%)</td>
</tr>
</tbody>
</table>
DCE as a development tool

DCE as a simulation tool
Mobile IP with handoff

Scenario
- ns-3 MAC/PHY wifi + mobility
- kernel tunneling
- umip signaling

Pros
- No need to re-implement IPv6 handoff signaling
- Greater realism than pure simulation
Huge scale experiment

Highlight
- Minimized virtualization
- High controlability

Example: HANA\(^2\)
- Assign IP addresses to all routers in the world
- Scaling VMs to this scale is not trivial
- Caida AS topology (36k ASes)
- MPI-based distributed simulation
  - partitioning: Metis
  - visualization: gephi

\(^2\)Fujikawa et al. *The Basic Procedures of Hierarchical Automatic Locator Number Allocation Protocol HANA*
Direct Code Execution allows
  Control of network conditions
  Reproducibility
  Debuggability
  Automation

For
  Userspace
  Kernelspace

Protocol implementations
http://www.nsnam.org/projects/direct-code-execution/
Thank you!

Questions?
Contact

mathieu.lacage@alcmeon.com

tazaki@nict.go.jp