A Network in a Laptop: Rapid Prototyping for Software-Defined Networks

Bob Lantz, Brandon Heller, Nick McKeown
Stanford University
HotNets 2010, 10/20/10
Wouldn’t it be amazing... 

if systems papers were runnable.
Wouldn’t it be amazing...

If systems papers made replicating their results, modifying the described system, and sharing it with others...

... as easy as downloading a file.
Wouldn’t it be amazing…

if network systems papers were more than runnable.

with no code changes!?!
Mininet: a platform for rapid network prototyping.

scales to usefully large nets runs unmodified applications provides path to hardware facilitates sharing
openflow.org/mininet

140+ users
45+ on mailing list
20+ institutions
open source (BSD license)

[don’t download now! save the WiFi!]
Demo
Demo Topology: Fat Tree

described in Scalable Commodity Data Center, SIGCOMM 2008, Al Fares et al.
(1) share-able
(2) runs on hardware
Date

Nov 2009: deadline in 3 months

[based on a true story]
Resources: a laptop
Goal: build/eval/demo a realistic new networked system
Why not a real system?

- as real as it gets
- a pain to reconfigure
Why not networked virtual machines?

- easier topology changes
- scalability

VM1 switch s1

VM2
host h2

VM3
host h3
Why not a simulator?

+ good visibility
- no path to hardware
Problem 1: 
Want scale with unmodified applications.
→ Use lightweight, OS-level virtualization.
OS-level Virtualization

Same system, different view. Almost zero overhead. ex. IMUNES, Emulab

one OS kernel
Problem 2: Want a smooth path to hardware deployment.

→ Use Software-Defined Networking.
Software-Defined Network

- OpenFlow
- Feature
- Feature
- Network OS
- Packet Forwarding
- Packet Forwarding
- Packet Forwarding
- Packet Forwarding
Mininet
Walkthrough
```bash
$> mn --topo minimal \
   --switch ovsk \ 
   --controller ref
```

run Mininet launcher

root network namespace

```
mn
```
$> \texttt{mn --topo minimal \ --switch ovsk \ --controller ref}

\textbf{Hosts}

\textbf{create bash processes}

\begin{tikzpicture}
\node[draw,rounded corners] (host1) at (0,0) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, below of=host1] (host2) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, above of=host1] (host3) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, below of=host1, xshift=-2cm] (host4) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, above of=host1, xshift=-2cm] (host5) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, below of=host1, xshift=2cm] (host6) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\node[draw,rounded corners, above of=host1, xshift=2cm] (host7) {
\begin{verbatim}
/host/bin/bash
\end{verbatim}
};
\draw[->,>=stealth, dashed, green] (host1) -- (host2);
\draw[->,>=stealth, dashed, green] (host1) -- (host3);
\draw[->,>=stealth, dashed, green] (host1) -- (host4);
\draw[->,>=stealth, dashed, green] (host1) -- (host5);
\draw[->,>=stealth, dashed, green] (host1) -- (host6);
\draw[->,>=stealth, dashed, green] (host1) -- (host7);
\end{tikzpicture}
$> mn --topo minimal \n   --switch ovsk \n   --controller ref

unshare(CLONE_NEWNET)
$> mn --topo minimal \n  --switch ovsk \n  --controller ref

`ip link add`

```
root namespace

  namespace

  network

  namespace

Links

veth pairs

veth0
veth1
veth2
veth3

/min/bash

h2 namespace

/min/bash

h3 namespace
```
$> \text{mn} \quad \text{--topo} \quad \text{minimal} \quad \text{\backslash} \\
\quad \text{--switch} \quad \text{ovsk} \quad \text{\backslash} \\
\quad \text{--controller} \quad \text{ref}
$> \textit{mn --topo minimal \ switch ovsk \ controller ref}$

\begin{center}
\textbf{Links}
\end{center}

\begin{center}
\textbf{ip link set netns}
\end{center}

\begin{center}
\textbf{root network namespace}
\end{center}

\begin{center}
\begin{tikzpicture}

\node [rectangle, draw] (m) at (0,0) {\textit{m\text{n}}};
\node [rectangle, draw] (s1) at (3,1) {\textit{s1-eth1}};
\node [rectangle, draw] (s2) at (3,-1) {\textit{s1-eth2}};
\node [rectangle, draw] (h2) at (-3,0) {\textit{h2-eth0}};
\node [rectangle, draw] (h3) at (3,0) {\textit{h3-eth0}};
\node [rectangle, draw] (h2b) at (-3,-1.5) {\textit{/bin/bash}};
\node [rectangle, draw] (h3b) at (3,-1.5) {\textit{/bin/bash}};
\node [rectangle, draw] (h2n) at (-3,-3) {\textit{h2 namespace}};
\node [rectangle, draw] (h3n) at (3,-3) {\textit{h3 namespace}};
\node [rectangle, draw] (r) at (0,0) {\textit{pipes}};

\draw [dotted, thick] (m) -- (r);
\draw [dotted, thick] (m) -- (s1);
\draw [dotted, thick] (m) -- (s2);
\draw [dotted, thick] (r) -- (h2);
\draw [dotted, thick] (r) -- (h3);
\draw [dotted, thick] (h2) -- (h2b);
\draw [dotted, thick] (h3) -- (h3b);
\draw [dotted, thick] (h2) -- (h2n);
\draw [dotted, thick] (h3) -- (h3n);

\end{tikzpicture}
\end{center}
$> mn --topo minimal \n   --switch ovsk \n   --controller ref

create OpenFlow Switch
$> mn --topo minimal \n   --switch ovsk \n   --controller ref

create controller
Virtual Machine

root network namespace

ofdatapath

ofprotocol

unix socket

controller

pipes

s1-eth1

s1-eth2

veth pairs

h2-eth0

h3-eth0

h2 namespace

h3 namespace

/bin/bash

/input

/nombre

pipes

ofprotocol

unix socket

controller

/s1-eth1

/s1-eth2

veth pairs

h2-eth0

h3-eth0

h2 namespace

h3 namespace

/bin/bash
Mininet example commands

Create a network using mn launcher:

```bash
mn --switch ovsk --controller nox --topo \ tree,depth=2,fanout=8 --test pingAll
```

Interact with a network using CLI:

```bash
mininet> h2 ping h3
mininet> h2 py dir(locals())
```

Customize a network w/API:

```python
from mininet.net import Mininet
from mininet.topolib import TreeTopo

tree4 = TreeTopo(depth=2,fanout=2)
net = Mininet(topo=tree4)
net.start()

h1, h4 = net.hosts[0], net.hosts[3]
print h1.cmd('ping -c1 %s' % h4.IP())
net.stop()
```
Apps made with the Mininet API
Evaluation
### Startup/Shutdown/Memory

<table>
<thead>
<tr>
<th>Topology</th>
<th>H</th>
<th>S</th>
<th>Setup(s)</th>
<th>Stop(s)</th>
<th>Mem(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Minimal</td>
<td>2</td>
<td>1</td>
<td>1.0</td>
<td>0.5</td>
<td>6</td>
</tr>
<tr>
<td>Linear(100)</td>
<td>100</td>
<td>100</td>
<td>70.7</td>
<td>70.0</td>
<td>112</td>
</tr>
<tr>
<td>VL2(4, 4)</td>
<td>80</td>
<td>10</td>
<td>31.7</td>
<td>14.9</td>
<td>73</td>
</tr>
<tr>
<td>FatTree(4)</td>
<td>16</td>
<td>20</td>
<td>17.2</td>
<td>22.3</td>
<td>66</td>
</tr>
<tr>
<td>FatTree(6)</td>
<td>54</td>
<td>45</td>
<td>54.3</td>
<td>56.3</td>
<td>102</td>
</tr>
<tr>
<td>Mesh(10, 10)</td>
<td>40</td>
<td>100</td>
<td>82.3</td>
<td>92.9</td>
<td>152</td>
</tr>
<tr>
<td>Tree(4^4)</td>
<td>256</td>
<td>85</td>
<td>168.4</td>
<td>83.9</td>
<td>233</td>
</tr>
<tr>
<td>Tree(16^2)</td>
<td>256</td>
<td>17</td>
<td>139.8</td>
<td>39.3</td>
<td>212</td>
</tr>
<tr>
<td>Tree(32^2)</td>
<td>1024</td>
<td>33</td>
<td>817.8</td>
<td>163.6</td>
<td>492</td>
</tr>
</tbody>
</table>

lots of switches & hosts w/ reasonable amounts of memory
### Microbenchmarks

<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create a node (host/switch/controller)</td>
<td>10</td>
</tr>
<tr>
<td>Run command on a host (‘echo hello’)</td>
<td>0.3</td>
</tr>
<tr>
<td>Add link between two nodes</td>
<td>260</td>
</tr>
<tr>
<td>Delete link between two nodes</td>
<td>416</td>
</tr>
<tr>
<td>Start user space switch (OpenFlow reference)</td>
<td>29</td>
</tr>
<tr>
<td>Stop user space switch (OpenFlow reference)</td>
<td>290</td>
</tr>
<tr>
<td>Start kernel switch (Open vSwitch)</td>
<td>332</td>
</tr>
<tr>
<td>Stop kernel switch (Open vSwitch)</td>
<td>540</td>
</tr>
</tbody>
</table>

Link management is slow.
Bandwidth

<table>
<thead>
<tr>
<th>$S$ (Switches)</th>
<th>User (Mbps)</th>
<th>Kernel (Mbps)</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>445</td>
<td>2120</td>
<td>~5x</td>
</tr>
<tr>
<td>10</td>
<td>49.9</td>
<td>940</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>25.7</td>
<td>573</td>
<td></td>
</tr>
<tr>
<td>40</td>
<td>12.6</td>
<td>315</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>6.2</td>
<td>267</td>
<td></td>
</tr>
<tr>
<td>80</td>
<td>4.15</td>
<td>217</td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>2.96</td>
<td>167</td>
<td>~50x</td>
</tr>
</tbody>
</table>

**usable amount of bandwidth**
Case Studies
Research Examples

• Ripcord: modular data center
• Asterix: wide-area load balancing
• SCAFFOLD: new internet architecture
• Distributed snapshot demo
Unexpected Uses

• Tutorials
• Whole-network regression suites
• Bug replication
Limitations
Inherent Limitations

• OS-level virtualization → one kernel only
• Linux containers → Linux programs only
• Cannot match the introspection of an event-driven simulation
Performance Fidelity

issues:
- performance
- predictability
- isolation

\[ p \gg c \] time multiplexing

\textbf{issues: performance predictability, isolation}
Wouldn’t it be amazing...

if systems papers were runnable.
Wouldn’t it be amazing...

If systems papers made replicating their results, modifying the described system, and sharing it with others...

... as easy as downloading a file.
Wouldn’t it be amazing... if network systems papers were more than runnable.

- idea → prototype
- share with others
- deploy on hardware

with no code changes!??
“A Network in a Laptop…”
is a runnable paper

...which itself describes how to make other runnable papers.
Mininet

- Rapid prototyping
- Scalable
- Shareable
- Functionally correct
- Path to hardware

enables “runnable papers” for a subset of networking

openflow.org/mininet
The SDN Approach

Separate control from the datapath
  – i.e. separate policy from mechanism

**Datapath:** Define minimal network instruction set
  – A set of “plumbing primitives”
  – A narrow interface: e.g. OpenFlow

**Control:** Define a network-wide OS
  – An API that others can develop on
How to get performance fidelity?

- Careful process-to-core allocation
- Bandwidth limits
- Scheduling priorities
- Real-time scheduling
- Scheduling groups w/resource isolation
Why not processes?

- scales better
- breaks applications