Towards Correct Network Virtualization

Soudeh Ghorbani
Brighten Godfrey

UIUC

HotSDN 2014
Virtualization

Hypervisor

App
VM
App
VM
App
VM

x86
Virtualization

Hypervisor

Physical Network

Network Virtualization

Firewall
Load-balancer
Router

L2 bridge

x86

Soudeh Ghorbani and Brighten Godfrey
HotSDN 2014
Virtualization

Diagram inspired by Teemu Koponen’s NSDI 2014 talk on “Network Virtualization in Multi-tenant Datacenters”.
Is the physical implementation a faithful reproduction of the virtual network?
Virtual firewall

Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.
Virtual firewall

Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.
Virtual firewall

Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.
Virtual firewall

Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.
Virtual firewall

Policy: permit an external server to talk to an internal client if and only if the client has sent a request to the server.
## Virtual firewall app

<table>
<thead>
<tr>
<th>Priority</th>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>srcip=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>
### Virtual firewall app

<table>
<thead>
<tr>
<th>Priority</th>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>srcip=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>

Soudeh Ghorbani and Brighten Godfrey                HotSDN 2014
## Virtual firewall app

<table>
<thead>
<tr>
<th>Priority</th>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>srcip=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>

Soudeh Ghorbani and Brighten Godfrey                HotSDN 2014
Virtual firewall app

<table>
<thead>
<tr>
<th>Priority</th>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>srcip=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>

switch(msg.getType()) {
    case PACKET_IN:
        if (internal.contains(msg.srcMAC())) {
            whitelisted[msg.dstMAC()][msg.srcMAC()] = true;
        } else {
            if (whitelisted[msg.srcMAC()][msg.dstMAC()]) {
                whitelist(sw, msg);
            } else {
                blacklist(sw, msg);
            }
        }
}
Packet-in from an internal client? Save state: dst server is allowed to send back.

```
switch(msg.getType()) {
    case PACKET_IN:
        if (internal.contains(msg.srcMAC())) {
            whitelisted[msg.dstMAC()][msg.srcMACA()] = true;
        } else {
            if (whitelisted[msg.srcMAC()][msg.dstMAC()]) {
                whitelist(sw, msg);
            } else {
                blacklist(sw, msg);
            }
        }
    }
```
**Virtual firewall app**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>srcip=130.126.**.*</td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>0</td>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>

Switch(msg.getType()) {
    case PACKET_IN:
        if (internal.contains(msg.srcMAC())) {
            whitelisted[msg.dstMAC()][msg.srcMACA()] = true;
        } else {
            if (whitelisted[msg.srcMAC()][msg.dstMAC()]) {
                whitelist(sw, msg);
            } else {
                blacklist(sw, msg);
            }
        }
}

Packet-in from an external server?
- If the server is allowed to send, install rules to allow bidirectional traffic.
- Else, blacklist the external server.
Virtual firewall

Firewall App
Virtual firewall

Firewall

App

1

2
Virtual firewall

1. Network traffic enters the virtual firewall.
2. The firewall inspects the traffic and decides whether to allow or block it.
3. The firewall allows the traffic to pass through to its destination.

Firewall App
Virtual firewall

1. Access from local network
2. Request to firewall
3. Firewall lookup
4. Route to external network

Firewall
App
Virtual firewall

1. Network access
2. Firewall processing
3. Allowance or denial
4. Connection establishment
5. Traffic monitoring
Firewall + virtualization = bug
Firewall + virtualization = bug

<table>
<thead>
<tr>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>
Firewall + virtualization = bug

<table>
<thead>
<tr>
<th>Flow</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=130.126.<em>.</em></td>
<td>Send to controller, fwd(1)</td>
</tr>
<tr>
<td>*</td>
<td>Send to controller</td>
</tr>
</tbody>
</table>

Flow Action

* Send to controller
Firewall + virtualization = bug
Firewall + virtualization = bug
Firewall + virtualization = bug
## Network virtualization: What could go wrong?

<table>
<thead>
<tr>
<th>App</th>
<th>Virtualization technique</th>
<th>Incorrect-behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stateful firewall</td>
<td>One-to-many mapping</td>
<td>Blacklisting the legitimate hosts</td>
</tr>
<tr>
<td>NAT</td>
<td>One-to-many mapping</td>
<td>Dropping requested packets</td>
</tr>
<tr>
<td>Load-balancer</td>
<td>One-to-many mapping</td>
<td>Overloading some servers and leaving some underutilized</td>
</tr>
<tr>
<td>Firewall &amp; router</td>
<td>Many-to-one mapping</td>
<td>Blacklisting the legitimate hosts</td>
</tr>
</tbody>
</table>
Related work

• Incorrect behavior caused by moving, observed in:


• These existing solutions are:
  ◦ Only a short-term fix while virtual network is being moved.
  ◦ Infeasible when incorrect behavior is permanent rather than transient.
Root-cause of the incorrect behavior
Firewall + virtualization = bug
Firewall + virtualization = bug
Firewall + virtualization = bug

Root-cause: forwarding decision has some dependency on the **history**, the sequence of previous ‘send’ and ‘receive’ events.
Who programs the network?

- The entities that can make or influence the forwarding decisions:
  - **Controller**
  - **Switch**: random forwarding like ECMP
  - **Data packet**: indirectly through local state, e.g., idle-timers
Who programs the network?

- The entities that can make or influence the forwarding decisions:
  - **Controller**
  - **Switch**: random forwarding like ECMP
  - **Data packet**: indirectly through local state, e.g., idle-timers
Can existing correctness definitions detect the incorrect behavior?

Correctness conditions:

1. Per-packet/flow consistency: prevents loops, black-holes, ...
   
   Consensus Routing [NSDI’08], Consistent Updates [SIGCOMM’12]

2. Congestion freedom
   
   zUpdates [SIGCOMM’13], SWAN [SIGCOMM’13], On Consistent Updates in Software-Defined Networks [HotNets’13]
Can existing correctness definitions detect the incorrect behavior?

1. Per-packet/flow consistency: prevents loops, black-holes,...
   Consensus Routing [NSDI’08], Consistent Updates [SIGCOMM’12]

2. Congestion freedom
   zUpdates [SIGCOMM’13], SWAN [SIGCOMM’13], On Consistent Updates in Software-Defined Networks [HotNets’13]

None of these conditions were violated in our examples!
Can existing correctness definitions detect the incorrect behavior?

Correctness conditions:

1. Per-packet/flow consistency: prevents loops, blackholes,…
   Consensus Routing [NSDI’08], Consistent Updates [SIGCOMM’12]

2. Congestion freedom
   zUpdates [SIGCOMM’13], SWAN [SIGCOMM’13],
   On Consistent Updates in Software-Defined Networks [HotNets’13]

None of these conditions were violated in our examples!

“Correctness is what users want.”
Leslie Lamport

[SIGCOMM’12]
Can existing correctness definitions detect the incorrect behavior?

1. **None** of these conditions were violated in our examples!

2. “Correctness is what users want.”
   Leslie Lamport

   \[\text{[SIGCOMM’12]}\]

3. Techniques designed to preserve those correctness conditions could **break the otherwise correct behavior**.

   \[\text{On Consistent Updates in Software-Defined Networks [HotNets’13]}\]

Soudeh Ghorbani and Brighten Godfrey

HotSDN 2014
Can existing correctness definitions detect the incorrect behavior?

1. None of these conditions were violated in our examples!

2. "Correctness is what users want." Leslie Lamport

3. Techniques designed to preserve those correctness conditions could break the otherwise correct behavior.

4. We need new definitions of correctness and new techniques to achieve those.
A new correctness condition: End-to-end correctness
A new correctness condition: End-to-end correctness
A new correctness condition: End-to-end correctness
A new correctness condition: End-to-end correctness

- A mapping of a logical network $L$ to a physical network $P$ is said to be end-to-end correct iff $Pr_L[E] \approx Pr_P[E]$ where $E$ is the partially ordered set of ‘send’ and ‘receive’ events.
A new correctness condition: End-to-end correctness

- A mapping of a logical network $L$ to a physical network $P$ is said to be end-to-end correct iff $Pr_L[E] \approx Pr_P[E]$ where $E$ is the partially ordered set of ‘send’ and ‘receive’ events.

- Key features:
  - distinguishes between events that happen always, sometimes, and never.
A new correctness condition: End-to-end correctness

• A mapping of a logical network L to a physical network P is said to be **end-to-end correct** iff

\[ Pr_L[E] \approx Pr_P[E] \]

where E is the partially ordered set of ‘send’ and ‘receive’ events.

• Key features:
  • distinguishes between events that happen **always**, **sometimes**, and **never**.
  • permissive of the **differences in packet loss or timing** that do not affect correctness.
A new correctness condition: End-to-end correctness

- A mapping of a logical network $L$ to a physical network $P$ is said to be end-to-end correct iff $P_{r_L}[E] \approx P_{r_P}[E]$ where $E$ is the partially ordered set of ‘send’ and ‘receive’ events.

- Key features:
  - distinguishes between events that happen always, sometimes, and never.
  - permissive of the differences in packet loss or timing that do not affect correctness.
  - permissive of the legitimate differences in orderings of events.
So far:

1. We identified the **problem**: incorrect application-level behavior under the existing virtualization techniques.

2. We identified its **root-cause**: dependence on the history.

3. We developed an analytical **framework** to reason about the problem.

Research Vision:

4. Developing a general **algorithm**.

5. Proving its **correctness**.

6. Developing a correct virtualization **System**.
Thanks!

Questions?