Splendid Isolation

A Slice Abstraction for Software Defined Networks

Joint work with:

Stephen Gutz

Alec Story

Nate Foster

PRINCETON UNIVERSITY

Aug. 2012

HotSDN

Cornell University
• How does one read the state of the network?

• How does one **read** the state of the network?
  

• How does one **write** the state of the network?
  
• How does one **read** the state of the network?

• How does one **write** the state of the network?

• How does one **define** a new (virtual) network?
  [Coming soon!]
• How does one **read** the state of the network?  

• How does one **write** the state of the network?  

• How does one **define** a new (virtual) network?  
  [Coming soon!]

• How does one **compose** two network programs?  
  [This talk.]
• How does one *compose* two network programs?
• How does one **compose** two network programs?

- Define a new slice abstraction.
- Lift slices (and isolation) into the language.
Data Center Isolation

Topology
Policy 2

Client 2
**Client 1 + Client 2**

Policy 1 + Policy 2
Client 2 injects packets into Client 1’s section of the network!
Client 2 intercepts packets from Client 1’s section of the network!
Our Approach

• Make isolation **part of the language.**
  – For *security* and *modularity*.

• Give each client a **slice** of the network which they can assume complete control over, as if they were alone on the network.

• Given a set of slices and a policy for each slice, **compile** them into one whole-network program that enforces isolation.
Slices

A_1 \quad A_2

R_1 \quad R_2 \quad R_3
# topology
topo = nxtopo.NXTopo()
topo.add_switch(name="R1",ports=[1,2,3,4])
topo.add_switch(name="R2",ports=[1,3,4])
topo.add_switch(name="A1",ports=[1,2])
topo.add_switch(name="A2",ports=[1,2])
Slices

# slice entries and exits
edges = [ ("R1", 1, tpDst 80, tpDst 80),
         ("R1", 2, tpDst 80, tpDst 80),
         ("R2", 1, tpDst 80, tpDst 80) ]

Predicate on incoming packets
Predicate on outgoing packets
# slice constructor
slice = Slice(topo, phys_topo, edges)
Isolation as Modularity

Multi-part Controller Program
Isolation as Modularity

ARP

C1_1  C1_2  C1_3  C2
Isolation as Modularity

ARP

MAC Learning
Isolation as Modularity

ARP
MAC Learning
Traffic Monitoring
Implementation

Input: a set of slices and NetCore policies.

(Must be VLAN-independent.)
Implementation

**Input:** a set of slices and NetCore policies.

*(Must be VLAN-independent.)*
Implementation

**Input:** a set of slices and NetCore policies.

**Output:** a single, global NetCore policy.
Implementation

**Input:** a set of slices and policies.  
(Must be VLAN-independent.)

**Output:** a single, global NetCore policy.
FlowVisor

Controller 1

Controller 2

A

R

H

H

A

R

H

H

H

H

H

H

H

FlowVisor
Verification
Verification

Model NetCore policies in SMT (Z3).
Verification

1. Model NetCore policies in SMT (Z3).

2. Verify isolation.
Verification

1. Model NetCore policies in SMT (Z3).

2. Verify isolation.

1. Verify semantic equivalence.

\[ \phi \land \phi' \]

\[ \psi \land \psi' \]

\[ A_1 \quad A_2 \]

Slice Compiler
Contributions

• A new language for slices.
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
Contributions

• A **new language** for slices.
• A **compiler** that enforces isolation.
• A **verifier** that guarantees:
Contributions

• A **new language** for slices.
• A **compiler** that enforces isolation.
• A **verifier** that guarantees:
  – isolation

\[ \text{NetCore} \]

\[ \text{NOX, Nettle, Floodlight, etc.} \]
Contributions

• A **new language** for slices.
  – Security
  – Modularity
• A **compiler** that enforces isolation.
• A **verifier** that guarantees:
  – isolation, and
  – semantic equivalence.
Thank you!

Read the paper:
frenetic-lang.org/papers

Get the code:
github.com/frenetic-lang/netcore

See the demo:
Find me after the talk!

We wish to thank Shrutarshi Basu, Arjun Guha, Josh Reich, Mark Reitblatt, Jennifer Rexford, and David Walker for many helpful comments and suggestions.
THE END
Re-imagining the fundamentals of network implementation from a programming languages point of view:

• How does one *read* the state of the network?
• How does one *write* the state of the network?
• How does one *define* a new (virtual) network?
• How does one *compose* two network programs?

Frenetic is a new programming language we are creating to explore these questions and more
One program to rule them all ...
NetCore: Program Composition

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPort = 2</td>
<td>Forward 1</td>
</tr>
<tr>
<td>inPort = 1</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpSrc = 22</td>
<td>Drop</td>
</tr>
<tr>
<td>tpSrc = 80</td>
<td>Drop</td>
</tr>
</tbody>
</table>
NetCore: Program Composition

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPort = 2</td>
<td>Forward 1</td>
</tr>
<tr>
<td>inPort = 1</td>
<td>Forward 2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>tpSrc = 22</td>
<td>Drop</td>
</tr>
<tr>
<td>tpSrc = 80</td>
<td>Drop</td>
</tr>
</tbody>
</table>

NetCore

<table>
<thead>
<tr>
<th>Repeater</th>
<th>Monitor</th>
</tr>
</thead>
<tbody>
<tr>
<td>inPort 2 == forward [1] &lt;+&gt;</td>
<td>tpSrc 22 == Query_1 &lt;+&gt;</td>
</tr>
<tr>
<td>inPort 1 == forward [2]</td>
<td>tpSrc 80 == Query_2</td>
</tr>
</tbody>
</table>
VLAN-based Isolation
VLAN-based Isolation

Slice Compiler

VLAN=2

S_1 -> S_2 -> S_3 -> S_4

VLAN=1

S_1 -> S_2 -> S_3 -> S_4

VLAN=2

S_1 -> S_2 -> S_3 -> S_4

VLAN=1

S_1 -> S_2 -> S_3 -> S_4
VLAN-based Isolation

Slice Compiler
Data Center Isolation

Topology
Client 2 injects packets into Client 1’s section of the network!
Client 2 intercepts packets from Client 1’s section of the network!
Controller 1

Controller 2

Hypervisor

PORT == 80
PORT != 80
Isolation as Modularity

Multi-part Controller Program
Isolation as Modularity

ARP

A₁ A₂ A₃ A₄ B
Isolation as Modularity
Isolation as Modularity

ARP
LLDP
NAT
Isolation as Modularity

ARP
LLDP
NAT
IP Routing
Read-only Slices

- Network monitoring
- Usage-based billing
Precise Semantics
Contributions
Contributions

• A **new language** for slices.
Slices

A_1 -- R_1 -- A_2

R_2

R_3
# topology

topo = nxtopo.NXTopo()
topo.add_switch(name="R1", ports=[1, 2, 3, 4])
topo.add_switch(name="R2", ports=[1, 3, 4])
topo.add_switch(name="A1", ports=[1, 2])
topo.add_switch(name="A2", ports=[1, 2])
# slice entries and exits

edges = \[
([ (p, Top(), Top()) for p in topo.edge_ports("R1") ] +
[ (p, Top(), Top()) for p in topo.edge_ports("R2") ])

# slice constructor
slice = Slice(topo, phys_topo, edges)
Programming with Slices

# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)
Programming with Slices

# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)

# IP module
ip_slice = Slice(phys_topo, phys_topo, ip_edges)
ip_policy = gen_routing_policy(ip_slice)
Programming with Slices

# ARP module
arp_slice = Slice(phys_topo, phys_topo, arp_edges)
arp_policy = gen_arp_policy(arp_slice)

# IP module
ip_slice = Slice(phys_topo, phys_topo, ip_edges)
ip_policy = gen_routing_policy(ip_slice)

# Network-wide policy
slices = [(arp_slice, arp_policy), (ip_slice, ip_policy)]
whole_policy = compile(slices)
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
VLAN-based Isolation
VLAN-based Isolation

VLAN=2

VLAN=1

VLAN-based Isolation
VLAN-based Isolation
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
Contributions

• A **new language** for slices.
• A **compiler** that enforces isolation.
Contributions

• A new language for slices.
• A compiler that enforces isolation.
Contributions

• A new language for slices.
• A compiler that enforces isolation.
• A verifier that guarantees:
Contributions

• A new language for slices.
• A compiler that enforces isolation.
• A verifier that guarantees:
  – isolation
Contributions

- A **new language** for slices.
- A **compiler** that enforces isolation.
- A **verifier** that guarantees:
  - isolation, and
  - semantic equivalence.
Verifying the Results

<table>
<thead>
<tr>
<th>NetCore</th>
<th>Encoded using SMT (Z3)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A₁ A₂ A₃ A₄ B</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Encoded using model checking (NuSMV)</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image-url" alt="Diagram" /></td>
</tr>
</tbody>
</table>
Contributions

• A **new language** for slices.

• A **compiler** that enforces isolation.

• A **verifier** that guarantees:
  – isolation, and
  – semantic equivalence.
Thank you!

Read the paper:
frenetic-lang.org/papers

Get the code:
github.com/frenetic-lang/netcore
1. Slice 2 cannot inject packets into Slice 1.
1. Slice 2 cannot **inject** packets into Slice 1.
Slice 2 cannot inject packets into Slice 1.
1 Slice 2 cannot inject packets into Slice 1.
Confidentiality

Slice 2 cannot siphon packets from Slice 1.
Confidentiality

Slice 2 cannot siphon packets from Slice 1.
Confidentiality

Slice 2 cannot siphon packets from Slice 1.
Confidentiality

Slice 2 cannot siphon packets from Slice 1.
Establishing Isolation

• Confidentiality and integrity are global, end-to-end properties.
• But we can establish isolation by enforcing simple, local properties.
Implementation

• VLAN-based implementation.
Verification

• SAT encoding (with Z3)
• **Separate:** given two compiled slices, guarantee that they are separate.
• **Semantics-preserving:** given a source slice + program and a compiled program, verify that they are semantically equivalent.
Read the paper:
http://www.cs.princeton.edu/~cschlesi

Download the code:
https://github.com/frenetic-lang/netcore
Isolation as Modularity

- Queries/network monitoring.
- ARP
One approach: like SFI

- Draw a box around each network program and prevent them from broaching their respective boxes (slices).
  - Absolute.
  - Says nothing about what happens within a slice.
- FlowVisor takes this approach.
Problem: Very Coarse-grained

• 1: We want isolation and semantics preserving by construction.

• 2: We want read-only slices.
  – Consider an admin/billing slice that monitors use. Isolation is too strong, but without isolation, what do we have?

• 3: Isolation as modularity.
Limitations

• Isolation as modularity.

• Inter-slice interaction.

• Intra-slice semantics.
Splendid Isolation

Cole Schlesinger

HotSDN

Joint work with:

Cornell University
Stephen Gutz
Alec Story
Nate Foster

PRINCETON UNIVERSITY

Aug. 2012

One-color printing

When only one-color printing is available, the insignia, logotype, and name of college, school, or unit should be printed positive—in black or Cornell Red. The insignia, logotype, and name of college, school, or unit also can be reversed out of any color to white, as shown.