Formal Foundations for Networks

Nate Foster
Cornell University
Proof Assistants

Coq
Proof Assistants

Coq

Arjun Guha
Postdoc → UMass

Mark Reitblatt
PhD student

Rebecca Coombes
Undergrad student
Networks in the News

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.
Networks in the News

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was performed [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.
Networks in the News

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was performed [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.

Service outage was due to a series of internal network events that corrupted router data tables.
Networks in the News

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network change was performed [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm.

Service outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems.
Networks in the News

We discovered a misconfiguration on this pair of switches that caused what's called a "bridge loop" in the network.

A network outage was due to a series of internal network events that corrupted router data tables.

Experienced a network connectivity issue […] interrupted the airline's flight departures, airport processing and reservations systems.
Service outage was due to a series of internal network events that corrupted router data tables. A network change was performed [...] executed incorrectly [...] more "stuck" volumes and added more requests to the re-mirroring storm.

Networks in the News

We discovered a misconfiguration on this pair of switches that creates a network connectivity loop in the network. Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems.
Networks Today

There are hosts...
Networks Today

Connected by switches...
Networks Today

There are also servers...
Networks Today

Connected by routers...
Networks Today

And a load balancer...
Networks Today

And a gateway router...
Networks Today

There are other ISPs...
So we need to run BGP...
Networks Today

And we need a firewall to filter incoming traffic...
Networks Today

There are also wireless hosts...
Networks Today

So we need wireless gateways...
Networks Today

And yet more middleboxes for lawful intercept...
Each color represents a different set of control plane protocols and algorithms... this is madness!
Software-Defined Networks

A clean-slate architecture that generalizes network hardware and decouples control and forwarding
Software-Defined Networks

Key Ideas
• Standardize network devices
• Separate forwarding and control
Software-Defined Networks

Key Ideas
- Standardize network devices
- Separate forwarding and control
Software-Defined Networks

Key Ideas

• Standardize network devices
• Separate forwarding and control

Benefits

• A flexible platform for deploying new functionality
• Enables precise reasoning about network behavior
Example

Security Policy
Block SSH traffic

Monitoring Policy
Log incoming HTTP requests

Routing Policy
Forward all other traffic between hosts \{1,2,3\}
Software-Defined Networking
Software-Defined Networking

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
<th>Stats</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>dstPort:22</td>
<td></td>
<td>2538</td>
</tr>
<tr>
<td>Medium</td>
<td>dstIP:h1</td>
<td>Forward 1</td>
<td>9623</td>
</tr>
<tr>
<td>Low</td>
<td>*</td>
<td>Controller</td>
<td>87</td>
</tr>
</tbody>
</table>
Software-Defined Networking

Network Events
- Topology changes
- Diverted packets
- Traffic statistics

Control Messages
- Install rule
- Uninstall rules
- Query counters

NOX

Controller

OpenFlow Switch

OpenFlow Switch

OpenFlow Switch
Issue #1: Replicated Functionality
# Issue #1: Replicated Functionality

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>High</td>
<td>dstPort:22</td>
<td></td>
</tr>
<tr>
<td>Medium</td>
<td>dstIP:h1, dstPort: 80</td>
<td>Forward 1, Forward 4</td>
</tr>
<tr>
<td>Medium</td>
<td>dstIP:h2, dstPort: 80</td>
<td>Forward 2, Forward 4</td>
</tr>
<tr>
<td>Medium</td>
<td>dstIP:h3, dstPort: 80</td>
<td>Forward 3, Forward 4</td>
</tr>
<tr>
<td>Low</td>
<td>dstIP:h1</td>
<td>Forward 1</td>
</tr>
<tr>
<td>Low</td>
<td>dstIP:h2</td>
<td>Forward 2</td>
</tr>
<tr>
<td>Low</td>
<td>dstIP:h3</td>
<td>Forward 3</td>
</tr>
</tbody>
</table>
def switch_join(sw):
  sw.flowMod(High, { dstPort: 22 }, [])
  sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
  sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
  sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
  sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
  sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
  sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
  sw.flowMod(High, { dstPort: 22 }, [])
  sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
  sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
  sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
  sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
  sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
  sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 })
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 })
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])

    def packet_in(sw, pkt):
        actions = []
        if pkt.dstPort == 80:
            actions = actions + [Forward 4]
        if pkt.dstIP == h1:
            actions = actions + [Forward 1]
        if pkt.dstIP == h2:
            actions = actions + [Forward 2]
        if pkt.dstIP == h3:
            actions = actions + [Forward 3]
        sw.packet_out(pkt, actions)
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 })
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 })
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 })
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])

def packet_in(sw, pkt):
    actions = []
    if pkt.dstPort == 80:
        actions = actions + [Forward 4]
    if pkt.dstIP == h1:
        actions = actions + [Forward 1]
    if pkt.dstIP == h2:
        actions = actions + [Forward 2]
    if pkt.dstIP == 32:
        actions = actions + [Forward 3]
    sw.packet_out(pkt, actions)
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])

Issue #2: Control Message Reordering
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { 'dstPort': 22 }, [])
    sw.flowMod(Medium, { 'dstIP': 'h1', 'dstPort': 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { 'dstIP': 'h2', 'dstPort': 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { 'dstIP': 'h2', 'dstPort': 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { 'dstIP': 'h1' }, [Forward 1])
    sw.flowMod(Low, { 'dstIP': 'h2' }, [Forward 2])
    sw.flowMod(Low, { 'dstIP': 'h3' }, [Forward 3])

def switch_join(sw):
    sw.flowMod(High, { 'dstPort': 22 }, [])
    sw.flowMod(Medium, { 'dstIP': 'h1', 'dstPort': 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { 'dstIP': 'h2', 'dstPort': 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { 'dstIP': 'h2', 'dstPort': 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { 'dstIP': 'h1' }, [Forward 1])
    sw.flowMod(Low, { 'dstIP': 'h2' }, [Forward 2])
    sw.flowMod(Low, { 'dstIP': 'h3' }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 3, Forward 4])
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.barrierRequest()
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
    sw.barrierRequest()
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])

Issue #2: Control Message Reordering

Transient violation!
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.barrierRequest()
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
    sw.barrierRequest()
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, { dstPort: 22 }, [])
    sw.barrierRequest()
    sw.flowMod(Medium, { dstIP: h1, dstPort: 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { dstIP: h2, dstPort: 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { dstIP: h3, dstPort: 80 }, [Forward 3, Forward 4])
    sw.barrierRequest()
    sw.flowMod(Low, { dstIP: h1 }, [Forward 1])
    sw.flowMod(Low, { dstIP: h2 }, [Forward 2])
    sw.flowMod(Low, { dstIP: h3 }, [Forward 3])
def switch_join(sw):
    sw.flowMod(High, {\n        'dstPort': 22
    })
    sw.barrierRequest()
    sw.flowMod(Medium, {\n        'dstIP': h1,
        'dstPort': 80
    })
    sw.flowMod(Medium, {\n        'dstIP': h2,
        'dstPort': 80
    })
    sw.flowMod(Medium, {\n        'dstIP': h3,
        'dstPort': 80
    })
    sw.barrierRequest()
    sw.flowMod(Low, {\n        'dstIP': h1
    })
    sw.flowMod(Low, {\n        'dstIP': h2
    })
    sw.flowMod(Low, {\n        'dstIP': h3
    })
Issue #3: Malformed Patterns

def switch_join(sw):
    sw.flowMod(High, { 'ethType': 0x800, 'ipType': 6, 'dstPort': 22 }, [])
    sw.barrierRequest()
    sw.flowMod(Medium, { 'ethType': 0x800, 'proto': 6, 'dstIP': h1, 'dstPort': 80 }, [Forward 1, Forward 4])
    sw.flowMod(Medium, { 'ethType': 0x800, 'proto': 6, 'dstIP': h2, 'dstPort': 80 }, [Forward 2, Forward 4])
    sw.flowMod(Medium, { 'ethType': 0x800, 'proto': 6, 'dstIP': h3, 'dstPort': 80 }, [Forward 3, Forward 4])
    sw.barrierRequest()
    sw.flowMod(Low, { 'ethType': 0x800, 'dstIP': h1 }, [Forward 1])
    sw.flowMod(Low, { 'ethType': 0x800, 'dstIP': h2 }, [Forward 2])
    sw.flowMod(Low, { 'ethType': 0x800, 'dstIP': h3 }, [Forward 3])
# Controller Bugs

<table>
<thead>
<tr>
<th>Violation</th>
<th>Guilty Controllers</th>
</tr>
</thead>
<tbody>
<tr>
<td>Handles PacketIn incorrectly</td>
<td>PANE (absolved)</td>
</tr>
<tr>
<td>Omits barrier messages</td>
<td>NetCore, PANE, Nettle-FRP</td>
</tr>
<tr>
<td>Generates malformed patterns</td>
<td>NetCore, PANE, Nettle-FRP†</td>
</tr>
<tr>
<td>Bogus flow table optimizations</td>
<td>NetCore (absolved)</td>
</tr>
</tbody>
</table>

† Nettle only affected by missing protocol numbers
Existing Tools

There is a cottage industry in configuration-checking tools...
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
- VeriFlow [HotSDN ’12]
There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
- VeriFlow [HotSDN ’12]
- and many others...
Existing Tools

There is a cottage industry in configuration-checking tools...

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
- VeriFlow [HotSDN ’12]
- and many others...

These are all great tools!

But they are expensive to run, and each builds on a custom (typically ad hoc) mathematical foundation
A Different Approach

- Write programs in a declarative network programming language
- Reason about the behavior of the network at a suitably high level of abstraction
- Use a compiler and run-time system to generate low-level OpenFlow messages
- Establish the correctness of the compiler and run-time system formally using a mechanized proof assistant
Certified Software Systems

Recent Successes
- seL4 [SOSP ’09]
- CompCert [CACM ’09]
- F* [ICFP ‘11, POPL ’12, ’13]

Tools

Textbooks

Certified Programming with Dependent Types
Recent Successes
• seL4 [SOSP ’09]
• CompCert [CACM ’09]
• F* [ICFP ‘11, POPL ’12, ’13]

Tools

Textbooks
Recent Successes

- seL4 [SOSP ’09]
- CompCert [CACM ’09]
- F* [ICFP ’11, POPL ’12, ’13]

Tools

Textbooks
Certified Software Systems

Recent Successes
- seL4 [SOSP ’09]
- CompCert [CACM ’09]
- F* [ICFP ‘11, POPL ’12, ’13]

Tools
- Isabelle
- ACL2

Textbooks
- Certified Programming with Dependent Types

Write code
Prove correct
Extract code
Recent Successes

- seL4 [SOSP ’09]
- CompCert [CACM ’09]
- F* [ICFP ‘11, POPL ’12, ’13]

Tools

- Isabelle
- ACL2

Textbooks

- Certified Programming with Dependent Types

Certified Software Systems

Write code
Prove correct
Extract code

Certified binary
Certified Network Controller

NetCore

Compiler

Flow tables

Run-time system

OpenFlow messages

Featherweight OpenFlow

Optimizer
Certified Network Controller

- Each level of abstraction formalized in Coq
Certified Network Controller

- Each level of abstraction formalized in Coq
- Machine-checked proofs that the transformations between levels preserve semantics
Certified Network Controller

- Each level of abstraction formalized in Coq
- Machine-checked proofs that the transformations between levels preserve semantics
- Code extracted to OCaml and deployed with real switch hardware

Diagram:
- NetCore
  - Compiler
  - Flow tables
  - Run-time system
  - OpenFlow messages
  - Featherweight OpenFlow

Optimizer
Certified Network Controller

- Each level of abstraction formalized in Coq
- Machine-checked proofs that the transformations between levels preserve semantics
- Code extracted to OCaml and deployed with real switch hardware
Syntax

- Classify packets by location and contents
  
  ```
  switch s inport n srcIP h dstP h
  ```

- Apply transformations and query packets
  
  ```
  Forward n Modify h n Query x
  ```

- Combine predicates using boolean operators
  
  ```
  e1 || e2 e1 && e2 !e
  ```

- Compose policies in parallel and condition on predicate
  
  ```
  p1 || p2 e => p
  ```
NetCore

Syntax

• Classify packets by location and contents

```
switch s inport n srcIP h dstP h
```

• Apply transformations and query packets

```
Forward n Modify h n Query x
```

• Combine predicates using boolean operators

```
e1 || e2 e1 && e2 !e
```

• Compose policies in parallel and condition on predicate

```
p1 || p2 e => p
```

Example

```
!(dstPort 22) =>
(dstIP h1 => Forward 1 ||
dstIP h2 => Forward 2 ||
dstIP h3 => Forward 3 ||
dstPort 80 => Forward 4)
```
NetCore Semantics

**Semantics**

- $p$ – NetCore program
- $M$ – multiset of “located” packets
- $lp$ – located packet processed in this step

\[
[p](lp) = M'
\]

\[
p \vdash \{lp\} \uplus M \xrightarrow{lp} M \uplus M'
\]
NetCore Semantics

Semantics

- \( p \) – NetCore program
- \( M \) – multiset of “located” packets
- \( lp \) – located packet processed in this step

\[
\llbracket p \rrbracket (lp) = M'
\]

\[
p \vdash \{lp\} \uplus M \xrightarrow{lp} M \uplus M'
\]

Features

- Models hop-by-hop forwarding behavior of the network
- Abstracts away from the underlying distributed system
- Enables simple reasoning about network-wide properties
Flow Tables

- An intermediate language like RTL in traditional compilers
- Idealized version of hardware tables maintained by switches

Syntax

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, ipProto: 6, dstPort:22</td>
<td></td>
</tr>
<tr>
<td>ethType:0x800, ipProto: 6, dstIP:H1, dstPort: 80</td>
<td>Forward 1, Forward 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Flow Tables

- An intermediate language like RTL in traditional compilers
- Idealized version of hardware tables maintained by switches

Syntax

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, ipProto: 6, dstPort:22</td>
<td></td>
</tr>
<tr>
<td>ethType:0x800, ipProto: 6, dstIP:H1, dstPort: 80</td>
<td>Forward 1, Forward 4</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Semantics

\[
\llbracket ft \rrbracket (lp) = M'
\]

\[
ft \vdash \llbracket lp \rrbracket \cup M \xrightarrow{lp} M \cup M'
\]
NetCore Compiler

• Translates from NetCore programs down to flow tables
• “Flattens” logical operator to linear sequence of forwarding rules
• Key operation: flow table intersection
NetCore Compiler

• Translates from NetCore programs down to flow tables
• “Flattens” logical operator to linear sequence of forwarding rules
• Key operation: flow table intersection

Example

dstIP: h1 => Forward 1
NetCore Compiler

- Translates from NetCore programs down to flow tables
- “Flattens” logical operator to linear sequence of forwarding rules
- Key operation: flow table intersection

Example

dstIP: h1 => Forward 1

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, dstIP: h1</td>
<td>Forward 1</td>
</tr>
</tbody>
</table>
NetCore Compiler

- Translates from NetCore programs down to flow tables
- "Flattens" logical operator to linear sequence of forwarding rules
- Key operation: flow table intersection

**Example**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, dstIP: h1</td>
<td>Forward 1</td>
</tr>
</tbody>
</table>

- dstIP: h1 => Forward 1
- dstPort: 80 => Forward 4
NetCore Compiler

- Translates from NetCore programs down to flow tables
- “Flattens” logical operator to linear sequence of forwarding rules
- Key operation: flow table intersection

Example

![Pattern Actions Table]

- \( \text{Pattern: } \text{ethType: 0x800}, \text{ dstIP: h1 } \rightarrow \text{Actions: Forward 1} \)
- \( \text{Pattern: } \text{dstPort: 80 } \rightarrow \text{Actions: Forward 4} \)

![Pattern Actions Table]

- \( \text{Pattern: } \text{ethType: 0x800}, \text{ proto: 6}, \text{ dstPort: 80 } \rightarrow \text{Actions: Forward 4} \)
NetCore Compiler

- Translates from NetCore programs down to flow tables
- “Flattens” logical operator to linear sequence of forwarding rules
- Key operation: flow table intersection

Example

dstIP: h1 => Forward 1

dstPort: 80 => Forward 4

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, dstIP: h1</td>
<td>Forward 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, proto: 6, dstPort: 80</td>
<td>Forward 4</td>
</tr>
</tbody>
</table>
NetCore Compiler

- Translates from NetCore programs down to flow tables
- “Flattens” logical operator to linear sequence of forwarding rules
- Key operation: flow table intersection

Example

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, dstIP: h1</td>
<td>Forward 1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ethType:0x800, proto: 6, dstIP: h1, dstPort: 80</td>
<td>Forward 1, Forward 4</td>
</tr>
<tr>
<td>ethType:0x800, dstIP: h1</td>
<td>Forward 1</td>
</tr>
<tr>
<td>ethType:0x800, proto: 6, dstPort: 80</td>
<td>Forward 4</td>
</tr>
</tbody>
</table>
Compiler Correctness

Optimizer

- Naive compiler produces exponentially large flow tables
- Optimizer removes “empty” and “shadowed” rules
Compiler Correctness

Optimizer

• Naive compiler produces exponentially large flow tables
• Optimizer removes “empty” and “shadowed” rules

Correctness Theorem

\[
\text{Theorem } \text{compile\_correct} : \\
\forall \text{opt pol sw pt pk bufId}, \\
\text{SemanticsPreserving opt } \rightarrow \\
\text{netcore\_eval pol sw pt pk bufId} = \\
\text{flowtable\_eval (compile pol sw) sw pt pk bufId}.
\]
Compiler Correctness

Optimizer
- Naive compiler produces exponentially large flow tables
- Optimizer removes “empty” and “shadowed” rules

Correctness Theorem

Theorem compile_correct :
  \(\forall\) opt pol sw pt pk bufId, SemanticsPreserving opt ->
  netcore_eval pol sw pt pk bufId =
  flowtable_eval (compile pol sw) sw pt pk bufId.

Formalization Highlights
- Library of algebraic properties of flow tables
- New tactic for proving equalities on bags
- Key invariant: all intermediate patterns “natural”
Natural Patterns

\[
\text{Inductive ValidPattern : pattern -> Prop :=}
| \text{ValidPat_{TCPUDP} : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos tpSrc tpDst inPort nwProto,}
| \quad \text{In nwProto SupportedNwProto ->}
| \quad \text{ValidPattern (Pattern dlSrc dlDst (WildcardExact Const_0x800) dlVlan dlVlanPcp nwSrc nwDst}
| \quad \quad \quad \quad \text{(WildcardExact nwProto) nwTos tpSrc tpDst inPort)}
| \text{| ValidPat_{ARP} : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst inPort,}
| \quad \text{ValidPattern (Pattern dlSrc dlDst (WildcardExact Const_0x806) dlVlan dlVlanPcp}
| \quad \quad \quad \text{nwSrc nwDst WildcardAll WildcardAll WildcardAll WildcardAll WildcardAll inPort)}
| \text{| ValidPat_{IP\_other} : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos inPort nwProto,}
| \quad \text{~ In nwProto SupportedNwProto ->}
| \quad \text{ValidPattern (Pattern dlSrc dlDst (WildcardExact Const_0x800) dlVlan dlVlanPcp nwSrc nwDst}
| \quad \quad \quad \text{(WildcardExact nwProto) nwTos WildcardAll WildcardAll WildcardAll inPort)}
| \text{| ValidPat_{IP\_NoNwProto} : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos inPort,}
| \quad \text{ValidPattern (Pattern dlSrc dlDst (WildcardExact Const_0x800) dlVlan dlVlanPcp nwSrc nwDst}
| \quad \quad \quad \text{WildcardAll nwTos WildcardAll WildcardAll WildcardAll inPort)}
| \text{| ValidPat_{OtherFrameTyp} : forall dlSrc dlDst dlVlan dlVlanPcp inPort frameTyp,}
| \quad \text{~ In frameTyp SupportedDlTyp ->}
| \quad \text{ValidPattern (Pattern dlSrc dlDst (WildcardExact frameTyp) dlVlan dlVlanPcp WildcardAll WildcardAll}
| \quad \quad \quad \text{WildcardAll WildcardAll WildcardAll WildcardAll inPort)}
| \text{| ValidPat_{AnyFrameTyp} : forall dlSrc dlDst dlVlan dlVlanPcp inPort,}
| \quad \text{ValidPattern (Pattern dlSrc dlDst WildcardAll dlVlan dlVlanPcp WildcardAll WildcardAll}
| \quad \quad \quad \text{WildcardAll WildcardAll WildcardAll WildcardAll inPort)}
| \text{| ValidPat_{None} : forall pat,}
| \quad \text{Pattern.is_empty pat = true ->}
| \quad \text{ValidPattern pat.}
\]
NetCore

 Compiler

 Flow tables

 Run-time system

 OpenFlow messages

 Featherweight OpenFlow

 Optimizer
OpenFlow 1.0 Specification

42 pages...

...of informal prose

...diagrams and flow charts

...and C struct definitions
**Key Features:**
- Models all features related to packet forwarding and all essential asynchrony
- Supports arbitrary controllers
/* Fields to match against flows */

struct ofp_match {
    uint32_t wildcards; /* Wildcard fields. */
    uint16_t in_port; /* Input switch port. */
    uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan; /* Input VLAN. */
    uint8_t dl_vlan_pcp; /* Input VLAN priority. */
    uint16_t dl_type; /* Ethernet frame type. */
    uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad2[2]; /* Align to 64-bits. */
    uint32_t nw_src; /* IP source address. */
    uint32_t nw_dst; /* IP destination address. */
    uint16_t tp_src; /* TCP/UDP source port. */
    uint16_t tp_dst; /* TCP/UDP destination port. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */
struct ofp_match {
    uint32_t wildcards; /* Wildcard fields. */
    uint16_t in_port; /* Input switch port. */
    uint16_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
    uint16_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
    uint16_t dl_vlan; /* Input VLAN. */
    uint16_t dl_vlan_pcp; /* Input VLAN priority. */
    uint16_t dl_type; /* Ethernet frame type. */
    uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits). */
    uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
    uint8_t pad1[1]; /* Align to 64-bits. */
    uint8_t pad2[2]; /* Align to 64-bits. */
    uint8_t nw_src; /* IP source address. */
    uint8_t nw_dst; /* IP destination address. */
    uint16_t tp_src; /* TCP/UDP source port. */
    uint16_t tp_dst; /* TCP/UDP destination port. */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);
/* Fields to match against flows */

struct ofp_match {
  uint32_t wildcards; /* Wildcard fields. */
  uint16_t in_port; /* Input switch port. */
  uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address. */
  uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address. */
  uint16_t dl_vlan; /* Input VLAN. */
  uint8_t dl_vlan_pcp; /* Input VLAN priority. */
  uint8_t dl_type; /* Ethernet frame type. */
  uint8_t nw_tos; /* IP ToS (DSAP field, 6 bits). */
  uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode. */
  uint8_t nw_src; /* IP source address. */
  uint8_t nw_dst; /* IP destination address. */
  uint16_t tp_src; /* TCP/UDP source port. */
  uint16_t tp_dst; /* TCP/UDP destination port. */
};

OFP_ASSERT(sizeof(struct ofp_match) == 40);
Forwarding

/* Fields to match against flows */
struct ofp_match {
  
  uint32_t wildcards; /* Wildcard fields */
  uint16_t in_port; /* Input port */
  uint8_t dl_src[OFP_ETH_ALEN]; /* Ethernet source address */
  uint8_t dl_dst[OFP_ETH_ALEN]; /* Ethernet destination address */
  uint16_t dl_vlan; /* Input VLAN */
  uint8_t dl_vlan_pcp; /* Input VLAN priority */
  uint16_t dl_type; /* Ethernet frame type */
  uint8_t nw_tos; /* IP ToS (DSCP field, 6 bits) */
  uint8_t nw_proto; /* IP protocol or lower 8 bits of ARP opcode */
  uint8_t nw Src; /* IP source address */
  uint8_t nw_dst; /* IP destination address */
  uint16_t tp Src; /* TCP/UDP source port */
  uint16_t tp dst; /* TCP/UDP destination port */
};
OFP_ASSERT(sizeof(struct ofp_match) == 40);

Definition Pattern_inter (p : Pattern) :=
let d1Src := Wildcard_inter EthernetAddress.eqdec (ptrnD1Src p) (ptrnD1Src p') in
let d1Dst := Wildcard_inter EthernetAddress.eqdec (ptrnD1Dst p) (ptrnD1Dst p') in
let d1Type := Wildcard_inter Word16.eqdec (ptrnD1Type p) (ptrnD1Type p') in
let d1Vlan := Wildcard_inter Word16.eqdec (ptrnD1Vlan p) (ptrnD1Vlan p') in
let d1VlanPcp := Wildcard_inter Word16.eqdec (ptrnD1VlanPcp p) (ptrnD1VlanPcp p') in
let nwSrc := Wildcard_inter Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
let nwDst := Wildcard_inter Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
let nwProto := Wildcard_inter Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
let nwTos := Wildcard_inter Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
let tpSrc := Wildcard_inter Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
let tpDst := Wildcard_inter Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
let inPort := Wildcard_inter Word16.eqdec (ptrnInPort p) (ptrnInPort p') in
MkPattern d1Src d1Dst d1Type d1Vlan d1VlanPcp
  nwSrc nwDst nwProto nwTos
tpSrc tpDst
inPort.

Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
MkPattern
  (WildcardExact (pktD1Src pk)) (WildcardExact (pktD1Dst pk))
  (WildcardExact (pktD1Type pk))
  (WildcardExact (pktD1Vlan pk)) (WildcardExact (pktD1VlanPcp pk))
  (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pk))
  (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pk))
  (Wildcard_of_option (pktTpSrc pk)) (Wildcard_of_option (pktTpDst pk))
  (WildcardExact pt).

negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
Forwarding

```c
uint16_t
  tp_dst;
uint16_t
  tp_src;
uint32_t
  nw_dst;
uint32_t
  nw_src;

struct
/*
  Fields to match against flows */
  ofp_match
  {
    ow table update
    Detailed model of matching, forwarding, and flow table update

let d1Dest := Wildcard_inter EthernetAddress.eqdec (ptrnD1Dest p) (ptrnD1Dest p') in
let d1Type := Wildcard_inter Word16.eqdec (ptrnD1Type p) (ptrnD1Type p') in
let d1Vlan := Wildcard_inter Word16.eqdec (ptrnD1Vlan p) (ptrnD1Vlan p') in
let d1VlanPcp := Wildcard_inter Word8.eqdec (ptrnD1VlanPcp p) (ptrnD1VlanPcp p') in
let nwSrc := Wildcard_inter Word32.eqdec (ptrnNwSrc p) (ptrnNwSrc p') in
let nwDst := Wildcard_inter Word32.eqdec (ptrnNwDst p) (ptrnNwDst p') in
let nwProto := Wildcard_inter Word8.eqdec (ptrnNwProto p) (ptrnNwProto p') in
let nwTos := Wildcard_inter Word8.eqdec (ptrnNwTos p) (ptrnNwTos p') in
let tpSrc := Wildcard_inter Word16.eqdec (ptrnTpSrc p) (ptrnTpSrc p') in
let tpDst := Wildcard_inter Word16.eqdec (ptrnTpDst p) (ptrnTpDst p') in
let inPort := Wildcard_inter Word16.eqdec(inPort p) (ptrnInPort p') in
MkPattern d1Src d1Dest d1Type d1Vlan d1VlanPcp
  nwSrc nwDst nwProto nwTos
  tpSrc tpDst
  inPort.

Definition exact_pattern (pk : Packet) (pt : Word16.T) : Pattern :=
  MkPattern
  (WildcardExact (pktD1Src pk)) (WildcardExact (pktD1Dest pk))
  (WildcardExact (pktD1Type pk))
  (WildcardExact (pktD1Vlan pk)) (WildcardExact (pktD1VlanPcp pk))
  (WildcardExact (pktNwSrc pk)) (WildcardExact (pktNwDst pk))
  (WildcardExact (pktNwProto pk)) (WildcardExact (pktNwTos pk))
  (Wildcard_of_option (pktTpSrc pk)) (Wildcard_of_option (pktTpDst pk))
  (WildcardExact pt).

  negb (Pattern_is_empty (Pattern_inter (exact_pattern pk pt) pat)).
```
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

**Definition**

\[
\text{InBuf} := \text{Bag Packet.}
\]

**Definition**

\[
\text{OutBuf} := \text{Bag Packet.}
\]

**Definition**

\[
\text{OFInBuf} := \text{Bag SwitchMsg.}
\]

**Definition**

\[
\text{OFOutBuf} := \text{Bag Ctrl1Msg.}
\]
Asynchrony

“In the absence of barrier messages, switches may arbitrarily reorder messages to maximize performance.”

“There is no packet output ordering guaranteed within a port.”

Essential asynchrony: packet buffers, message reordering, and barriers

**Definition**

\[
\text{InBuf} := \text{Bag Packet.}
\]

\[
\text{OutBuf} := \text{Bag Packet.}
\]

\[
\text{OFInBuf} := \text{Bag SwitchMsg.}
\]

\[
\text{OFOutBuf} := \text{Bag CtrlMsg.}
\]
Controllers

Ultimately we want to prove theorems about controllers that implement the NetCore run-time system...

...but we didn’t want to bake specific controllers into Featherweight OpenFlow!
Controllers

Ultimately we want to prove theorems about controllers that implement the NetCore run-time system...

...but we didn’t want to bake specific controllers into Featherweight OpenFlow!

**Controller Parameters**

\[ \sum : \text{abstract type of controller state} \]

\[ f_{\text{in}} : (sw, CM, \sigma) \leadsto \sigma' \]

\[ f_{\text{out}} : \sigma \leadsto (sw, SM, \sigma') \]
Controllers

Ultimately we want to prove theorems about controllers that implement the NetCore run-time system...

...but we didn’t want to bake specific controller state into Featherweight OpenFlow.

Controller model: fully abstract

**Controller Parameters**

\[\Sigma: \text{abstract type of controller state}\]

\[f_{\text{in}}: (sw, CM, \sigma) \leadsto \sigma'\]

\[f_{\text{out}}: \sigma \leadsto (sw, SM, \sigma')\]
NetCore

Compiler

Flow tables

Run-time system

OpenFlow messages

Featherweight OpenFlow

Optimizer
There are many possible strategies for managing the rules installed on switches:

- **Naive**: don’t install any rules and process all packets on the controller instead
- **Reactive**: install fine-grained rules in response to the actual traffic seen in the network
- **Proactive**: compile and install coarse-grained rules that handle all traffic

Can also install rules with idle or hard timeouts...
Run-Time Systems

There are *many* possible strategies for managing the rules installed on switches:

- **Naive**: don’t install any rules and process all packets on the controller instead
- **Reactive**: install fine-grained rules in response to the actual traffic seen in the network
- **Proactive**: compile and install coarse-grained rules that handle all traffic

Can also install rules with idle or hard timeouts...

**Our goal**: a general framework for proving controller correctness
Run-Time Correctness

**Invariants**

- **Safety**: at all times, the rules installed on switches are an approximation of the controller function
- **Liveness**: the controller eventually processes all packets diverted to it by switches

**Correctness Theorem**

```
Module RelationDefinitions :=
    FwOF.FwOFRelationDefinitions.Make (AtomsAndController).

... 

Theorem fwof_abst_weak_bisim :
    weak_bisimulation
    concreteStep
    abstractStep
    bisim_relation.
```
Implementation

Source
• ~12,000 lines of Coq
• ~1,600 lines of OCaml

Components
• NetCore compiler and optimizer
• Flow tables
• Featherweight OpenFlow
• Run-time system instances
• Machine-checked proofs of correctness

Status
• Extracts to OCaml source + unverified “glue” code
• Running on “production” traffic at home and in the lab
Experience

Deployments

• *Lab*: Pronto 3290 Gigabit switch
• *Home*: TP-Link Wifi Router

Applications

• Host discovery
• Shortest-path routing
• Spanning tree
• Traffic monitoring
• Access control
Throughput

Experiment

• Controllers each run a simple repeater

• Use `cbench` to flood controllers with inputs and measure the time needed to generate a response

• Setup: dual-core 3.3 GHz Intel i3, 8GB RAM, Ubuntu 12.04

Results

Analysis

• Poor multicore support in OCaml hurts throughput

• Extra latency from copying OpenFlow messages to/from extracted code
Throughput

Experiment

• Controllers each run a simple repeater

• Use `cbench` to flood controllers with inputs and measure the time needed to generate a response

• Setup: dual-core 3.3 GHz Intel i3, 8GB RAM, Ubuntu 12.04

Results

<table>
<thead>
<tr>
<th>Controller</th>
<th>Messages / Second</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unverified NetCore (Haskell)</td>
<td>26,022</td>
</tr>
<tr>
<td>NOX (Python and C++)</td>
<td>16,997</td>
</tr>
<tr>
<td>Verified NetCore (OCaml)</td>
<td>9,437</td>
</tr>
<tr>
<td>POX (Python)</td>
<td>6,150</td>
</tr>
</tbody>
</table>

Analysis

• Poor multicore support in OCaml hurts throughput

• Extra latency from copying OpenFlow messages to/from extracted code
Control Traffic

**Topology**
- Six-node Waxman graph
- Two hosts per switch

**Application**
- Broadcast along spanning tree
- Point-to-point forwarding along shortest paths

**Traffic**
- Each host periodically broadcasts ICMP request
- All other hosts send ICMP reply
Control Traffic

_verifiedNetCore__

_verifiedNetCore_

_unverifiedNetCore__

_unverifiedNetCore_

_MicroFlow_

_MicroFlow_

_PacketOut_

_PacketOut_
NetCore Verification Tool

- Encode NetCore programs and network topologies using Z3 (Xie et al. [INFOCOM ’05], Kazemian et al. [NSDI ’12])
- Express network transfer relation as the transitive closure of the NetCore and topology transfer relations
- Automatically check network reachability properties

```
(declare-datatypes ()
  ((Packet (packet (PSwitch Int) (PInPort Int) (PDlSrc Int) (PDlDst Int) ...))))
(declare-relation Program (Packet Packet))
(declare-relation Topology (Packet Packet))
(declare-relation Query ())
...
(query Query
  :default-relation smt_relation2
  :engine pdr
  :print-answer true)
```

We’ve also used Z3 to implement translation validation for NetCore
Networks are critical infrastructure...
...developed using 1970s-era techniques

Software-defined networks are a new architecture that could be used to put networking on a solid foundation

Machine-verified network controllers and NetCore verification tool are solid first steps in this direction
Other Abstractions

**Verified Controllers** [PLDI 2013]
- Featherweight model of OpenFlow, formalized in Coq
- Machine-verified compiler and run-time system

**Network Queries** [ICFP 2011]
- Declarative language for reading network state
- Decouples monitoring from forwarding

**Network Compilation** [POPL 2012]
- Expressive intermediate language
- Efficient proactive compiler

**Network Updates** [SIGCOMM 2012]
- Policy updates with strong consistency guarantees
- Runtime system automatically applies optimizations

**Network Isolation** [HotSDN 2012]
- Language abstraction for “slicing” the network
- Guarantees non-interference

**Network Composition** [NSDI 2013]
- Virtual networks via topology views
- Implementation via sequential composition
<table>
<thead>
<tr>
<th>Concern</th>
<th>Assembly Languages</th>
<th>Programming Languages</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Assembly Languages</td>
<td>Programming Languages</td>
</tr>
<tr>
<td></td>
<td>x86</td>
<td>OpenFlow</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resource Allocation</td>
<td>Move values to/from register</td>
<td>Manipulate forwarding rules</td>
</tr>
<tr>
<td>Resource Tracking</td>
<td>Have I spilled that register?</td>
<td>Will a packet arrive at the controller?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Coordination</td>
<td>Unregulated calling conventions</td>
<td>Unregulated rule management</td>
</tr>
<tr>
<td>Portability</td>
<td>Hardware dependent</td>
<td>Hardware dependent</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
A Grand Collaboration: PL + Networking

Cornell
Shrutarshi Basu (PhD)
Nate Foster (Faculty)
Arjun Guha (Postdoc)
Rebecca Coombes (Undergrad)
Mark Reitblatt (PhD)
Robert Soulé (Postdoc)
Alec Story (Undergrad)

Princeton
Nanxi Kang (PhD)
Christopher Monsanto (PhD)
Joshua Reich (Postdoc)
Jennifer Rexford (Faculty)
Cole Schlesinger (PhD)
David Walker (Faculty)
Naga Praveen Katta (PhD)

http://frenetic-lang.org
Come join us!
Summer School on Formal Methods and Networks
June 10-14, 2013
Cornell University
Ithaca, NY, USA

Speakers

Nikolaj Bjorner
Microsoft Research
Satisfiability Modulo Theories Solving for Network Verification

Brighten Godfrey
University of Illinois at Urbana-Champaign
Verifying Networks in Real Time

Timothy Griffin
University of Cambridge
Partial Automation in the Design and Implementation of Path-finding Algorithms

Arjun Guha
University of Massachusetts Amherst
Network Programming With Frenetic

Peyman Kazemian and Nick McKeown
Stanford University
Network Verification Using Header Space Analysis

Shriram Krishnamurthi
Brown University
Modeling and Reasoning about Network Components

Ratul Mahajan
Microsoft Research
Systematically Exploring the Behavior of Control Programs

Pamela Zave
AT&T Research
Compositional Abstractions of Network Architectures