Machine-Verified Network Controllers

Nate Foster
Cornell University
Proof Assistants

Coq
Proof Assistants

Coq

Arjun Guha
Postdoc → UMass

Mark Reitblatt
PhD student
“The trigger for this event was a network configuration change”

—Amazon
"The service outage was due to a series of internal network events that corrupted router data tables"

—GoDaddy
“The airline experienced a network connectivity issue...”
—United Airlines
“The airline experienced a network connectivity issue...”
—United Airlines
Networks in Practice

There are hosts...
Networks in Practice

Connected by switches...
Networks in Practice

There are also servers...
Networks in Practice

Connected by routers...
Networks in Practice

And a load balancer...
Networks in Practice

And a gateway router...
There are other ISPs...
So we need to run BGP...
And we need a firewall to filter incoming traffic...
Networks in Practice

There are also wireless hosts...
Networks in Practice

So we need wireless gateways...
Networks in Practice

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

A clean-slate architecture that standardizes features and decouples forwarding from...
**Software-Defined Networking**

**Essential ingredients**
- Decouple control and data planes
- Logically-centralized control

**Enables**
- Novel functionality
- Formal reasoning
Software-Defined Networking

**Essential ingredients**
- Decouple control and data planes
- Logically-centralized control

**Enables**
- Novel functionality
- Formal reasoning
Existing Tools

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

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

- **FlowChecker** [SafeConfig ’10]
- **AntEater** [SIGCOMM ’11]
There is a cottage industry in SDN configuration-checking tools...

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

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

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

- FlowChecker [SafeConfig ’10]
- AntEater [SIGCOMM ’11]
- NICE [NSDI ’12]
- Header Space Analysis [NSDI ’12]
- VeriFlow [HotSDN ’12]
- and many others...
There is a cottage industry in SDN 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) foundation.
Machine-Verified Controllers

**Vision**
- Develop programs in a high-level language
- Reason at a high level of abstraction
- Use a compiler and run-time system to generate low-level control messages
- Machine-verified proofs of correctness

**Contributions**
- NetCore compiler + optimizer
- Featherweight OpenFlow model
- General framework for establishing run-time system correctness
OVERVIEW
OpenFlow Switches

Forwarding Table: prioritized list of rules
Rule: pattern, actions, and counters
Pattern: prefix match on headers
Action: forward or modify
Counters: total bytes and packets processed

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
<th>Bytes</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>1010</td>
<td>Drop</td>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>010*</td>
<td>Forward(2)</td>
<td>100</td>
<td>4</td>
</tr>
<tr>
<td>011*</td>
<td>Controller</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
Network Events
- Topology changes
- Diverted packets
- Traffic statistics

Control Messages
- Modify rules
- Query counters
Issue #1: Switch-Level Errors

What happens if...

• The controller misses a keep-alive message?
• The controller sends a malformed message?
  - Bad output port
  - Too many actions
  - Inconsistent actions
  - Unsupported actions
• The switches runs out of space for rules?

Any of these can lead to essentially arbitrary behavior
Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```plaintext
FlowMod AddFlow {
  match = { srcIPAddress = 10.0.1.*", ... },
  actions = [ flood ], ... }
```
What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow {
  match = {
    srcIPAddress = 10.0.1.*", ...
  },
  actions = [ flood ], ...
}
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...
Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```python
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*"},
    actions = [ flood ], ... }
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...

...but it actually installs a rule that floods all traffic
Issue #2: Malformed Patterns

What happens if the controller sends the following message to a switch?

```
FlowMod AddFlow { match = { srcIPAddress = 10.0.1.*, },
actions = [ flood ], ... }
```

We’d expect the switch to install a rule that broadcasts all traffic from a host the given subnet...

...but it actually installs a rule that floods *all* traffic

Why? Switches *silently* ignore IP fields unless the Ethernet frame type is IP!
Issue #3: Message Reordering

What happens if the controller sends the following pair of OpenFlow messages to a switch in sequence?

```
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
srcIPAddress = "10.0.1.99", ... },
priority = 1,
actions = [ ] }

FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
srcIPAddress = "10.0.1.*", ... },
priority = 2,
actions = [ flood ] }
```

The intention is to encode a negation...
What happens if the controller sends the following pair of OpenFlow messages to a switch in sequence?

```
FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                           srcIPAddress = "10.0.1.99", ...
                           priority = 1,
                           actions = [ ] }

FlowMod AddFlow { match = { ethFrameType = ethTypeIP,
                           srcIPAddress = "10.0.1.*", ...
                           priority = 2,
                           actions = [ flood ] }
```

The intention is to encode a negation...

...but the switch may process these in either order!
MACHINE-VERIFIED CONTROLLERS
NetCore

Compiler

Flow tables

Run-time system

OpenFlow messages

Featherweight OpenFlow

Optimizer
NetCore

Compiler

Flow tables

Run-time system

OpenFlow messages

Featherweight OpenFlow

Optimizer
Syntax

**Inductive** pred : Type :=
- OnSwitch : Switch -> pred
- IngressPort : Port -> pred
- DlSrc : EthernetAddress -> pred
- DlDst : EthernetAddress -> pred
- DlVlan : option VLAN -> pred
- ...
- And : pred -> pred -> pred
- Or : pred -> pred -> pred
- Not : pred -> pred
- All : pred
- None : pred.

**Inductive** PseudoPort : Type :=
- PhysicalPort : Port -> PseudoPort
- AllPorts : PseudoPort.

**Inductive** act : Type :=
- FwdMod : Mod -> PseudoPort -> act

**Inductive** pol : Type :=
- Policy : pred -> list act -> pol
- Union : pol -> pol -> pol
- Restrict : pol -> pred -> pol.
Semantics

\[ lp = (sw, pt, pk) \]
\[ lps_{out} = pol(sw, pt, pk) \]
\[ S = \{ (T(sw, pt_{out}), pk) \mid (pt_{out}, pk) \in lps_{out} \} \]

\[ \{lp\} \cup \{lp_1 \cdots lp_n\} \xrightarrow{lp} S \cup \{lp_1 \cdots lp_n\} \]

- Models hop-by-hop forwarding behavior of the network
- Abstracts away from the underlying distributed system
- Makes it easy to reason about network-wide properties
NetCore

Compiler

Flow tables

Run-time system

Optimizer

OpenFlow messages

Featherweight OpenFlow
NetCore to Flow Tables

Example

<table>
<thead>
<tr>
<th>Priority</th>
<th>Pattern</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>65534</td>
<td><code>inPort = 2, dlSrc = d:ca:66:43:21</code></td>
<td>Fwd 2</td>
</tr>
<tr>
<td>65533</td>
<td><code>inPort = 2</code></td>
<td>Fwd 3</td>
</tr>
</tbody>
</table>

NetCore compiler
- Key operation: flow table intersection
- Must restrict to “valid” patterns

Optimizer
- Optimizer prunes (many) redundant rules
- Based on simple algebra of operations

Correctness Theorem
NetCore ~ FlowTable
Valid Patterns

**Inductive ValidPattern : Pattern -> Prop :=**

| SupportedIPPValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos
tpSrc tpDst inPort nwProto,
| | In nwProto SupportedL4Protos ->
| | ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x800)
dlVlan dlVlanPcp
nwSrc nwDst (WildcardExact nwProto)
nwTos tpSrc tpDst inPort)
| UnsupportedIPPValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst nwTos
| | inPort nwProto,
| | ~ In nwProto SupportedL4Protos ->
| | ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x800)
dlVlan dlVlanPcp
nwSrc nwDst (WildcardExact nwProto)
nwTos WildcardAll WildcardAll inPort)
| ARPPacketValid : forall dlSrc dlDst dlVlan dlVlanPcp nwSrc nwDst inPort,
| | ValidPattern (MkPattern dlSrc dlDst (WildcardExact Const_0x806)
dlVlan dlVlanPcp
nwSrc nwDst WildcardAll
WildcardAll WildcardAll WildcardAll inPort)
| UnknownDlTypPatternValid : forall dlSrc dlDst dlTyp dlVlan dlVlanPcp inPort,
| | ValidPattern (MkPattern dlSrc dlDst dlTyp
| | dlVlan dlVlanPcp
| | WildcardAll WildcardAll WildcardAll WildcardAll inPort)
| EmptyPatternValid : ValidPattern Pattern_empty.
NetCore

Flow tables

Run-time system

OpenFlow messages

Featherweight OpenFlow

Compiler

Optimizer
OpenFlow Specification

42 pages...
...of informal English text
...and C struct definitions
Key judgments:

- Controller in: \((sw, CM, \sigma) \rightsquigarrow \sigma'\)
- Controller out: \(\sigma \rightsquigarrow (sw, SM, \sigma')\)
- Network step: \(M \rightarrow M'\)

Models all essential asynchrony
NetCore

Compiler

Flow tables

Run-time system

Optimizer

OpenFlow messages

Featherweight OpenFlow
Invariants
- Maintain a sound approximation of overall flow table each switch
- Eventually process all diverted packets

Theorem
FlowTable $\approx$ Featherweight OpenFlow

Run-time instances
- Trivial: processes all packets on controller
- Proactive: installs rules, falls back to Trivial when out of space
- Full: like Proactive, but also installs exact-match rules
Inductive SafeWire : SF -> SF -> SF -> list CM -> Prop :=
| SafeWire_nil : forall lb ub,
  extends ub lb ->
  SafeWire lb ub lb nil
| SafeWire_cons_FlowMod : forall lb ub sf sft lst,
  SafeWire lb ub sf lst ->
  extends ub (apply_SFT sft sf) ->
  SafeWire lb ub (apply_SFT sft sf) (FlowMod sft :: lst)
| SafeWire_cons_PktOut : forall lb ub sf pt pk lst,
  SafeWire lb ub sf lst ->
  SafeWire lb ub sf (PktOut pt pk :: lst)
| SafeWire_cons_BARRIERRequest : forall lb ub sf n lst,
  SafeWire lb ub sf lst ->
  SafeWire lb ub sf (BARRIERRequest n :: lst).
Implementation

**Source**
- ~8,000 lines of Coq
- ~1,500 lines of Haskell

**Components**
- NetCore compiler and optimizer
- Flow tables
- Featherweight OpenFlow
- Run-time system instances
- Proofs of correctness

**Status**
- Extracts to Haskell source code
- Compiles against Nettle libraries
- Running on “production” traffic in the lab
Performance

![Performance Chart](image)

FlowMods / second

Unverified

Verified

Implementation

FlowMods / second:

- 0
- 1,250
- 2,500
- 3,750
- 5,000
Conclusion

Networks are critical infrastructure...

...developed using 1970s-era techniques

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

Machine-verified controllers based on NetCore a first step in this direction
A Grand Collaboration: Languages + Networking

**Frenetic Cornell**
- Shrutarshi Basu (PhD)
- Nate Foster (Faculty)
- Arjun Guha (Postdoc)
- Stephen Gutz (Undergrad)
- Mark Reitblatt (PhD)
- Robert Soulé (Postdoc)
- Alec Story (Undergrad)

**Frenetic Princeton**
- Chris Monsanto (PhD)
- Joshua Reich (Postdoc)
- Jen Rexford (Faculty)
- Cole Schlesinger (PhD)
- Dave Walker (Faculty)
- Naga Praveen Katta (PhD)

http://frenetic-lang.org