A Compiler and Run-time System for Network Programming Languages

Christopher Monsanto, Princeton
Nate Foster, Cornell
Rob Harrison, West Point
David Walker, Princeton
Software-Defined Networks

Switch

Controller

Switch
Software-Defined Networks

- Video routing
- Seamless mobility
- Energy efficiency
- DDoS protection
- Etc...

??
Software Defined Networks: Switches

• Processes packets according to classifier
  – Limited pattern matching capabilities
  – Actions: forwarding, dropping, sending to controller
• As fast as it gets

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>191.<em>.</em>.*.</td>
<td>80</td>
<td>Forward A</td>
</tr>
<tr>
<td>191.<em>.</em>.*</td>
<td>*</td>
<td>Drop</td>
</tr>
<tr>
<td>191.1.1.0</td>
<td>22</td>
<td>Forward B</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
SDN: Controllers

• Capable of arbitrary computation
• Orders of magnitude slower
• Installs and uninstalls rules from switch classifiers
Goals

1. A simple, declarative language...
   – That is, we specify the functionality, not the concrete rules that go on the switches, or the explicit install and uninstall commands the controller must issue

2. ... that is mathematically guaranteed to be correct and efficient.
Goals

1. A simple, declarative language...
   – That is, we specify the functionality, not the concrete rules that go on the switches, or the explicit install and uninstall commands the controller must issue

2. ... that is mathematically guaranteed to be correct and efficient.
Goals

• ... guaranteed to be efficient
Goals

• ... guaranteed to be *efficient*

On switches:
Goals

• ... guaranteed to be *efficient*

On switches:

On controllers:
Obstacles Faced By Programmers

Low level interface

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>191.<em>.</em>.*</td>
<td>80</td>
<td>Forward(A)</td>
</tr>
<tr>
<td>191.<em>.</em>.*</td>
<td>*</td>
<td>Drop</td>
</tr>
<tr>
<td>*</td>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

Differing switch capabilities

Synthesize communication protocol
Our Real Enemy: Complexity

• Managing these rules is complicated
• Tendency to fall back to using the simplest kind of rule: microflows (exact match rules)
• Very inefficient: many packets go to the controller!
New Frenetic Run-time Architecture

1. NetCore policy

2. Compilation

3. Refinement

(reactive specialization loop)
NETCORE

Compilation

NetCore policy

Refinement

Classifier
NetCore Grammar

Predicates

- $e ::= h : w$
  - switch $s$
  - inspect $f$
  - $e_1 \cup e_2$
  - $e_1 \setminus e_2$
  - $e_1 \cap e_2$
  - $\neg e$

Policies

- $\tau ::= e \rightarrow \{ s_1, \ldots, s_n \}$ action
  - $\tau_1 \cup \tau_2$
  - $\tau_1 \setminus \tau_2$
  - $\tau_1 \cap \tau_2$

Headers $h$, wildcards $w$

- $e$:
  - primitive match
  - match packets at switch
  - arbitrary function
  - set union
  - set difference
  - set intersection
  - set negation

- $\tau$:
  - action
  - policy union
NetCore Grammar

Predicates

- \( e ::= h : w \)
- \( \text{switch } s \)
- \( \text{inspect } f \)
- \( e \downarrow 1 \cup e \downarrow 2 \)
- \( e \downarrow 1 \setminus e \downarrow 2 \)
- \( e \downarrow 1 \cap e \downarrow 2 \)
- \( \neg e \)

Policies

- \( \tau ::= e \rightarrow \{ s \downarrow 1 ,..., s \downarrow n \} \)
- \( \tau \downarrow 1 \cup \tau \downarrow 2 \)

Headers \( h, \) wildcards \( w \)

- primitive match
- match packets at switch
- arbitrary function
- set union
- set difference
- set intersection
- set negation
- action
- policy union
NetCore Grammar

Predicates

• \( e ::= h:w \)
  | switch \( s \)
  | inspect \( f \)
  | \( e \downarrow 1 \cup e \downarrow 2 \)
  | \( e \downarrow 1 \setminus e \downarrow 2 \)
  | \( e \downarrow 1 \cap e \downarrow 2 \)
  | \( \neg e \)

Policies

• \( \tau ::= e \to \{ s \downarrow 1, \ldots, s \downarrow n \} \) action
  | \( \tau \downarrow 1 \cup \tau \downarrow 2 \)
  | \( \tau \downarrow 1 \setminus \tau \downarrow 2 \)

Headers \( h \), wildcards \( w \)

primitive match

DestIP: 10.*.1.*

arbitrary function

set union

set difference

set intersection

set negation

policy union
NetCore Grammar

**Predicates**

- \( e ::= h : w \) primitive match
- switch \( s \) match packets at switch
- inspect \( f \) arbitrary function
- \( e_1 \cup e_2 \) set union
- \( e_1 \setminus e_2 \) set difference
- \( e_1 \cap e_2 \) set intersection
- \( \neg e \) set negation

**Policies**

- \( \tau ::= e \rightarrow \{ s_1, \ldots, s_n \} \) action
- \( \tau_1 \cup \tau_2 \) policy union

Headers \( h \), wildcards \( w \)
NetCore Grammar

**Predicates**

- \( e ::= h : w \)  
  - switch \( s \)  
  - inspect \( f \)  
  - \( e \cup e' \)  
  - \( e \setminus e' \)  
  - \( e \cap e' \)  
  - \( \neg e \)

**Headsrs**  
\( h, \) wildcards \( w \)

**Policies**

- \( \tau ::= e \rightarrow \{ s_1, \ldots, s_n \} \) action
- \( \tau \cup \tau' \) policy union
NetCore Grammar

Predicates

• $e ::= h:w$
  | $\text{switch } s$
  | $\text{inspect } f$

badWebsite $p = \text{addr `elem` blacklist}$
where
  $\text{headers} = \text{parseHTTP } p$
  $\text{addr} = \text{headers} ! \text{“address”}$

Headers $h$, wildcards $w$
primitive match
match packets at switch
arbitrary function

Policies

• $\tau ::= e \rightarrow \{s\downarrow 1, \ldots, s\downarrow n\}$
  | $\tau\downarrow 1 \cup \tau\downarrow 2$

Set negation

action
policy union
NetCore Grammar

Predicates

- \( e ::= h:w \)  primitive match
- \( \text{switch } s \)  match packets at switch
- \( \text{inspect } f \)  arbitrary function

Headers \( h \), wildcards \( w \)

\text{badWebsite } p = \text{addr `elem` blacklist }
where
\begin{align*}
\text{headers} &= \text{parseHTTP } p \\
\text{addr} &= \text{headers }! \text{“address”}
\end{align*}

Policies

- \( \tau ::= e \rightarrow \{ s \downarrow 1, \ldots, s \downarrow n \} \)  action
- \( \tau \downarrow 1 \cup \tau \downarrow 2 \)  policy union
- \( \neg \tau \)  set negation

inspect (not \( \circ \text{badWebsite} \))
NetCore Grammar

Predicates

\[ e ::= \begin{cases} h : \text{wildcard} \\ \text{switch} \ s \\ \text{inspect} \ f \\ e \downarrow 1 \cup e \downarrow 2 \\ e \downarrow 1 \setminus e \downarrow 2 \\ e \downarrow 1 \cap e \downarrow 2 \\ \neg e \end{cases} \]

Headers $h$, wildcards $w$

primitive match
match packets at switch
arbitrary function
set union
set difference
set intersection
set negation

Policies

\[ \tau ::= \begin{cases} e \rightarrow \{ s \downarrow 1, \ldots, s \downarrow n \} \ \text{action} \\ \tau \downarrow 1 \cup \tau \downarrow 2 \ \text{policy union} \end{cases} \]
NetCore Grammar

Predicates

- \( e \ ::= \ h : w \)  
  - switch \( s \)  
  - inspect \( f \)  
  - \( e \downarrow 1 \cup e \downarrow 2 \)  
  - \( e \downarrow 1 \setminus e \downarrow 2 \)  
  - \( e \downarrow 1 \cap e \downarrow 2 \)  
  - \( \neg e \)  

Headers \( h \), wildcards \( w \)

- primitive match
- match packets at switch
- arbitrary function
- set union
- set difference
- set intersection

Policies

- inspect \((\text{not } \circ \text{badWebSite}) \cup \text{DestIP: 10.10.50.20}\)

- \( \tau \ ::= \ e \rightarrow \{ s \downarrow 1 ,..., s \downarrow n \} \)  
  - \( \tau \downarrow 1 \cup \tau \downarrow 2 \)  

action

(policy union)
NetCore Grammar

Predicates

• $e ::= h:w$
  | switch $s$
  | inspect $f$
  | $e_1 \cup e_2$
  | $e_1 \setminus e_2$
  | $e_1 \cap e_2$
  | $\neg e$

Policies

• $\tau ::= e \rightarrow \{s_1, \ldots, s_n\}$ action
  | $\tau_1 \cup \tau_2$

Headers $h$, wildcards $w$

primitive match
match packets at switch
arbitrary function
set union
set difference
set intersection
set negation

Predicates

Policies

action
policy union
NetCore Grammar

Predicates

• \( e ::= h:w \)
  |  switch \( s \)
  |  inspect \( f \)
  |  \( e \downarrow_1 \cup e \downarrow_2 \)
  |  \( e \downarrow_1 \setminus e \downarrow_2 \)
  |  \( e \downarrow_1 \cap e \downarrow_2 \)
  |  \( \neg e \)

Headers \( h \), wildcards \( w \)

primitive match
match packets at switch
arbitrary function
set union
set difference
set intersection
set negation

Policies

• \( \tau ::= \) inspect (not . badWebSite) \( \rightarrow \) \{A\}
  |  \( \tau \downarrow_1 \cup \tau \downarrow_2 \)
  |  \( \tau \downarrow_1 \setminus \tau \downarrow_2 \)
  |  \( \tau \downarrow_1 \cap \tau \downarrow_2 \)
  |  \( \neg \tau \)

policy union
NetCore Grammar

Predicates

• $e ::= h:w$
  | switch $s$
  | inspect $f$
  | $e\downarrow 1 \cup e\downarrow 2$
  | $e\downarrow 1 \setminus e\downarrow 2$
  | $e\downarrow 1 \cap e\downarrow 2$
  | $\neg e$

Policies

• $\tau ::= e \rightarrow \{ s\downarrow 1 ,..., s\downarrow n \}$ action
  | $\tau\downarrow 1 \cup \tau\downarrow 2$

Headers $h$, wildcards $w$

- primitive match
- match packets at switch
- arbitrary function
- set union
- set difference
- set intersection
- set negation

Predicates

- $e$ is an expression
- $h$ is a header
- $w$ is a wildcard

Policies

- $\tau$ is a policy
- $s\downarrow$ is a switch
- $f$ is an arbitrary function

Action

- $e \rightarrow \{ s\downarrow 1 ,..., s\downarrow n \}$
- $\tau\downarrow 1 \cup \tau\downarrow 2$

Policy Union

- $\tau\downarrow 1 \cup \tau\downarrow 2$
NetCore Example

- Load balanced, fast path
- Slow path

DestIP: *0 → {A} ∪ DestIP: *1 → {B}
NetCore Example

- Load balanced, fast path
  - Internal traffic (90.80.*.*)
  - SSH traffic
- Slow path

\[(\text{DestIP:90.80.*.* } \cup \text{ DestPort: 22}) \cap (\text{DestIP: *0 } \rightarrow \{A\} \cup \text{ DestIP: *1 } \rightarrow \{B\})\]
NetCore Example

• Load balanced, fast path
  • Internal traffic (90.80.*.*)
  • SSH traffic
• Slow path
  • External traffic

(((DestIP:90.80.*.* ∪ DestPort: 22) ∩ (DestIP: *0 → \{A\} ∪ DestIP: *1 → \{B\})) ∪ (¬DestIP:90.80.*.*) → \{C\})

• Load balanced, fast path
  • Internal traffic (90.80.*.*)
  • SSH traffic
• Slow path
  • External traffic

(((DestIP:90.80.*.* ∪ DestPort: 22) ∩ (DestIP: *0 → \{A\} ∪ DestIP: *1 → \{B\})) ∪ (¬DestIP:90.80.*.*) → \{C\})

• Load balanced, fast path
  • Internal traffic (90.80.*.*)
  • SSH traffic
• Slow path
  • External traffic

(((DestIP:90.80.*.* ∪ DestPort: 22) ∩ (DestIP: *0 → \{A\} ∪ DestIP: *1 → \{B\})) ∪ (¬DestIP:90.80.*.*) → \{C\})

• Load balanced, fast path
  • Internal traffic (90.80.*.*)
  • SSH traffic
• Slow path
  • External traffic

(((DestIP:90.80.*.* ∪ DestPort: 22) ∩ (DestIP: *0 → \{A\} ∪ DestIP: *1 → \{B\})) ∪ (¬DestIP:90.80.*.*) → \{C\})

• Load balanced, fast path
  • Internal traffic (90.80.*.*)
  • SSH traffic
• Slow path
  • External traffic

(((DestIP:90.80.*.* ∪ DestPort: 22) ∩ (DestIP: *0 → \{A\} ∪ DestIP: *1 → \{B\})) ∪ (¬DestIP:90.80.*.*) → \{C\})
COMPILATION
Compilation

P1 = DestIP:90.80.*.* → {A}

Compiles to:

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.80.<em>.</em></td>
<td>*</td>
<td>Forward A</td>
</tr>
</tbody>
</table>
Compilation: Approximations

P2 = DestIP:90.*.70.60 → {B}

Compiles to:

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.<em>.</em>.*</td>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

Non prefix match!

Overapproximation
Compilation

Run programs in parallel:

$$P_3 = P_1 \cup P_2$$

Compiles to:

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.80.70.60</td>
<td>*</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>*</td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
### Compilation

Run programs in parallel:

\[ P_3 = P_1 \cup P_2 \]

Subtle interaction: where did B come from?

Compiles to:

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.80.70.60</td>
<td>*</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>*</td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
Compilation: Exponential Growth

DestIP: 1.*.*.* → {A}

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.<em>.</em>.*</td>
<td>*</td>
<td>Forward A</td>
</tr>
</tbody>
</table>
Compilation: Exponential Growth

DestIP: 1.*.*.* → \{A\} ∪
DestIP: *.2.*.* → \{B\}

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.<em>.</em></td>
<td>*</td>
<td>Forward A, B</td>
</tr>
<tr>
<td><em>.2.</em>.*</td>
<td>*</td>
<td>Forward B</td>
</tr>
<tr>
<td>1.<em>.</em>.*</td>
<td>*</td>
<td>Forward A</td>
</tr>
</tbody>
</table>
## Compilation: Exponential Growth

DestIP: 1.*.*.* → {A} ∪
DestIP: *.2.*.* → {B} ∪
DestIP: *.3.*.* → {C}

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Dest. Port</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2.3.*</td>
<td>*</td>
<td>Forward A, B, C</td>
</tr>
<tr>
<td><em>.2.3.</em></td>
<td>*</td>
<td>Forward B, C</td>
</tr>
<tr>
<td>1.<em>.3.</em></td>
<td>*</td>
<td>Forward A, C</td>
</tr>
<tr>
<td><em>.3.</em></td>
<td>*</td>
<td>Forward C</td>
</tr>
<tr>
<td>1.2.<em>.</em></td>
<td>*</td>
<td>Forward A, B</td>
</tr>
<tr>
<td><em>.2.</em>.*</td>
<td>*</td>
<td>Forward B</td>
</tr>
<tr>
<td>1.<em>.</em>.*</td>
<td>*</td>
<td>Forward A</td>
</tr>
<tr>
<td>Dest. IP</td>
<td>Dest. Port</td>
<td>Action</td>
</tr>
<tr>
<td>----------</td>
<td>------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>1.2.3.4</td>
<td>*</td>
<td>Forward A, B, C, D</td>
</tr>
<tr>
<td>*.2.3.4</td>
<td>*</td>
<td>Forward B, C, D</td>
</tr>
<tr>
<td>1.*.3.4</td>
<td>*</td>
<td>Forward A, C, D</td>
</tr>
<tr>
<td><em>.</em>.3.4</td>
<td>*</td>
<td>Forward C, D</td>
</tr>
<tr>
<td>1.2.*.4</td>
<td>*</td>
<td>Forward A, B, D</td>
</tr>
<tr>
<td><em>.2.</em>.4</td>
<td>*</td>
<td>Forward B, D</td>
</tr>
<tr>
<td>1.<em>.</em>.4</td>
<td>*</td>
<td>Forward A, D</td>
</tr>
<tr>
<td><em>.</em>.*.4</td>
<td>*</td>
<td>Forward D</td>
</tr>
<tr>
<td>1.2.3.*</td>
<td>*</td>
<td>Forward A, B, C</td>
</tr>
<tr>
<td><em>.2.3.</em></td>
<td>*</td>
<td>Forward B, C</td>
</tr>
<tr>
<td>1.<em>.3.</em></td>
<td>*</td>
<td>Forward A, C</td>
</tr>
<tr>
<td><em>.</em>.3.*</td>
<td>*</td>
<td>Forward C</td>
</tr>
<tr>
<td>1.2.<em>.</em></td>
<td>*</td>
<td>Forward A, B</td>
</tr>
<tr>
<td><em>.2.</em>.*</td>
<td>*</td>
<td>Forward B</td>
</tr>
<tr>
<td>1.<em>.</em>.*</td>
<td>*</td>
<td>Forward A</td>
</tr>
</tbody>
</table>
Compilation: Takeaways

• Switch classifiers are not compositional – they do not contain enough information
  – Compiler uses an intermediate form which makes use of high-level information

• Classifiers that implement high-level specifications grow in size quickly
  – Compiler uses optimized algorithms and classifier minimization techniques
REACTIVE SPECIALIZATION
Reactive Specialization

- Can’t compile this classifier exactly without using many rules
  - Most switches only support prefix matches
  - Worst case can take billions and billions of rules

DestIP: 90.80.*** → {A} ∪
DestIP: 90.*.70.60 → {B}
R. Specialization: Base Compilation

• Step 1: install a “base” classifier
  – Proactively handle packets on switch if possible
  – Send other packets to the controller

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

Policy

Proactive Compilation
R. Specialization: Refinement

- Packet \{\text{DestIP}=90.2.60.70, \text{DestPort}=80\} comes in
- Use packet to generate semantically equivalent, but structurally different policy

\[
\begin{align*}
\text{DestIP: } 90.80.\ast.\ast &\rightarrow \{A\} \cup \\
\text{DestIP: } 90.\ast.70.60 &\rightarrow \{B\}
\end{align*}
\]
R. Specialization: Recompilation

• Compile refined policy
  – Extra structural information handles similar packets to the one we handled

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.2.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
R. Specialization: Pruning

- Remove controller rules and rules that don’t match packet we handled.
- Take result and place on switches.

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.2.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
R. Specialization: Pruning

- Remove controller rules and rules that don’t match packet we handled
- Take result and place on switches

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.2.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.2.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>
R. Specialization: Second Packet

- Consider handling a second packet, \(\{\text{DestIP}=90.100.70.60, \text{DestPort}=21\}\)

<table>
<thead>
<tr>
<th>Dest. IP</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>90.100.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.2.70.60</td>
<td>Forward B</td>
</tr>
<tr>
<td>90.80.70.60</td>
<td>Forward A, B</td>
</tr>
<tr>
<td>90.80.<em>.</em></td>
<td>Forward A</td>
</tr>
<tr>
<td>90.<em>.</em>.*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

DestIP: \(90.80.*.* \rightarrow \{A\} \cup (\text{DestIP: } 90.*.70.60 \cup \text{DestIP: } 90.100.70.60) \rightarrow \{B\}\)
THEORETICAL & EMPIRICAL RESULTS
Functional Correctness

• Define idealized machine that handles all traffic at the controller

Theorem (Functional Correctness):

*NetCore machines bisimulate idealized machines.*

• Statement of correctness more subtle when considering queries
Quiescence

• Recall: Programs are efficient if they handle traffic on the switches, not the controller.

Theorem (Quiescence):

*Under reasonable circumstances, NetCore programs eventually handle all traffic on the (fast) switches, instead of on the (slow) controller*

• Special case of cost theorem: identify packets sent to controller
Benchmarks: Controller Traffic

Controller Traffic (kB)

5k | 50k | 95k
---|---|---

Packet counts:
- 5k
- 50k
- 95k

Graphs:
- Static Policy
- Static Policy + Monitoring
- Network Auth.

Legend:
- uFlow
- Full
Conclusion

• This talk was short; we didn’t talk about:
  – Switch heterogeneity & lattice theory
  – Arbitrary Haskell functions inside policies
  – Queries
  – How compilation and specialization actually work

• So:
  – Check out the paper!
  – Talk to Chris, Dave, or Nate!
  – Check out the compiler at frenetic-lang.org!