Programming Network Policies by Examples: Platform, Abstraction and User Studies

Boon Thau Loo
University of Pennsylvania
NetPL workshop @ SIGCOMM 2017

Joint work with Yifei Yuan, Dong Lin, Siri Anil, Harsh Verma, Anirudh Chelluri, Rajeev Alur, Boon Thau Loo
Programming Abstractions in SDN

General-purpose Programming Languages
• C/C++ (NOX)
• Python (POX)
• Java (Floodlight)
• ...

Domain Specific Languages (DSL)
• Declarative Networking [CACM’09]
• Frenetic [ICFP’11]
• NetCore [POPL’12]
• Pyretic [NSDI’13]
• NetKAT [POPL’14]
• ...

Missing: SDN programming for non-programmers
NSF I-Corps Program

- Interviewed 101 network operators/architects, software engineers and router vendors
  - Need for programming skills among network operators
  - Frequent network configuration changes that should be automated
  - (Lack of) programming expertise an impediment towards SDN usage.
Need for Programming Capabilities

Needs for programming skills among network operators

- **Strong need**: 60.81%
- **Good to have**: 21.62%
- **Needs**: 17.57%
Outline

- NetEgg overview
- Controlled user study
NetEgg Overview

Synthesizer

Policy

Controller

Interpreter

Network events

Action Instructions / Rule installation

Example behaviors

Conflicts

Bad behaviors
NetEgg Overview

Synthesizer

Example behaviors

Policy

Interpreter

Controller

Network events

Action Instructions / Rule installation

Bad behaviors

Conflicts
A First Example

Learning switch:
1. Learn the mapping between hosts and ports
2. Forward packets according to the learnt mapping
A Stateful Policy Model

- Intuition: states + cases (policies)
- Learning switch:
  - States: Mapping between hosts and ports
  - Case 1. Flood packets if the destination is unknown; store the port for the source
  - Case 2. Forward packets otherwise; store port for source in packets
**Execution Model**

### State Table

<table>
<thead>
<tr>
<th>MAC</th>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

### Policy Table

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>ST(p.dst).state=0</td>
<td>flood</td>
<td>ST(p.src):=(1,port)</td>
</tr>
<tr>
<td>*</td>
<td>ST(p.dst).state=1</td>
<td>fwd(ST(p.dst).value)</td>
<td>ST(p.src):=(1,port)</td>
</tr>
</tbody>
</table>

---

**Controller**

**A**

**B**

**C**
Why This Model?

- **Simple:**
  - Fast execution
  - Deployment on dataplane (FAST [HotSDN’14], openState SIGCOMM CCR’14)

- **Expressive:**
  - Dynamic policies (Kinetic [NSDI’2015])
**Synthesizer**

\[
\begin{align*}
\langle p_1, h_1, h_2 \rangle &\quad \text{flood} \\
\langle p_2, h_3, h_1 \rangle &\quad \text{fwd}(p_1) \\
\langle p_3, h_2, h_3 \rangle &\quad \text{fwd}(p_2)
\end{align*}
\]

---

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>$\text{ST}(\text{dstmac}).\text{state}=0$</td>
<td>flood</td>
<td>$\text{ST}(\text{srcmac}) := (1,\text{port})$</td>
</tr>
<tr>
<td>*</td>
<td>$\text{ST}(\text{dstmac}).\text{state}=1$</td>
<td>$\text{fwd}(\text{ST(\text{dstmac}.value})$</td>
<td>$\text{ST}(\text{srcmac}) := (1,\text{port})$</td>
</tr>
</tbody>
</table>
Synthesis Algorithm

- Goal: consistent, minimal
Synthesis Algorithm

- **Goal**: consistent, minimal
- **Greedy search**

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>ST(dst).state=0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Synthesis Algorithm

- **Goal:** consistent, minimal
- **Greedy search**

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>ST(dst).state=0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>ST(dst).state=0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>*</td>
<td>ST(dst).state=1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Backtrack with pruning**
### Idea: Keep rules not updating state tables on the switch

<table>
<thead>
<tr>
<th>MAC</th>
<th>State</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Match</th>
<th>Test</th>
<th>Action</th>
<th>Update</th>
</tr>
</thead>
<tbody>
<tr>
<td>*</td>
<td>ST(dstmac).state=0</td>
<td>flood</td>
<td>ST(srcmac):(1,port)</td>
</tr>
<tr>
<td>*</td>
<td>ST(dstmac).state=1</td>
<td>fwd(ST(dstmac).value)</td>
<td>ST(srcmac):(1,port)</td>
</tr>
</tbody>
</table>

```plaintext
<inport=3, srcmac=B, dstmac=A>
```

<table>
<thead>
<tr>
<th>Match</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>inport=3, srcmac=B, dstmac=A</td>
<td>fwd(2)</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Evaluation

- Is scenario-based programming feasible?
  - Expressiveness
  - Intuitiveness
  - Conciseness
  - Efficiency

- Is the performance of synthesized implementations comparable to hand-crafted implementation?
  - Controller response latency
  - End-to-end performance
## Breadth and Synthesis Effort/Time

<table>
<thead>
<tr>
<th>Network Policy</th>
<th>#EV</th>
<th>#SC</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>maclearner1</td>
<td>3</td>
<td>1</td>
<td>11 ms</td>
</tr>
<tr>
<td>maclearner2</td>
<td>3</td>
<td>1</td>
<td>15 ms</td>
</tr>
<tr>
<td>auth</td>
<td>3</td>
<td>2</td>
<td>13 ms</td>
</tr>
<tr>
<td>gardenwall</td>
<td>5</td>
<td>3</td>
<td>52 ms</td>
</tr>
<tr>
<td>ids</td>
<td>3</td>
<td>2</td>
<td>15 ms</td>
</tr>
<tr>
<td>monitor</td>
<td>3</td>
<td>2</td>
<td>13 ms</td>
</tr>
<tr>
<td>ratelimiter</td>
<td>10</td>
<td>5</td>
<td>147 ms</td>
</tr>
<tr>
<td>serverlb</td>
<td>7</td>
<td>3</td>
<td>143 ms</td>
</tr>
<tr>
<td>stateful firewall1</td>
<td>3</td>
<td>2</td>
<td>12 ms</td>
</tr>
<tr>
<td>stateful firewall2</td>
<td>3</td>
<td>2</td>
<td>16 ms</td>
</tr>
<tr>
<td>stateful firewall3</td>
<td>6</td>
<td>3</td>
<td>107 ms</td>
</tr>
<tr>
<td>trafficlb</td>
<td>7</td>
<td>3</td>
<td>402 ms</td>
</tr>
<tr>
<td>ucap</td>
<td>3</td>
<td>2</td>
<td>13 ms</td>
</tr>
<tr>
<td>vmprov</td>
<td>3</td>
<td>2</td>
<td>24 ms</td>
</tr>
<tr>
<td>TCP firewall</td>
<td>9</td>
<td>5</td>
<td>64 ms</td>
</tr>
<tr>
<td>ARP proxy</td>
<td>5</td>
<td>2</td>
<td>49 ms</td>
</tr>
</tbody>
</table>

Table 11: **Network policies generated from scenarios.**

#SC is the number of scenarios used to synthesize the policy, #EV is the total number of events in scenarios, Time is the running time of the synthesizer.
## Comparison in Code Size

<table>
<thead>
<tr>
<th></th>
<th>NetEgg</th>
<th>Pyretic</th>
<th>POX</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac learner</td>
<td>3</td>
<td>17</td>
<td>29</td>
</tr>
<tr>
<td>Stateful firewall</td>
<td>3</td>
<td>21</td>
<td>58</td>
</tr>
<tr>
<td>TCP firewall</td>
<td>9</td>
<td>24</td>
<td>68</td>
</tr>
</tbody>
</table>
Response Time

- mac learner
- stateful firewall
- tcp firewall

Response time (ms)

- netegg
- pox
End-to-end Performance

- Topology: fattree, 20 switches, 16 hosts
- Policy: learning switch
- Setup:
  - 1 host as HTTP server
  - other hosts send HTTP requests to the server
  - benchmark connection time (i.e. time between a request is issued and it is finished.)
End-to-end Performance

![Graph showing end-to-end performance with HTTP connection time on the y-axis and connection rate on the x-axis. The graph compares two systems, netegg and pox.](image-url)
NetEgg Toolkit
Outline

- NetEgg overview
- Controlled user study
Controlled User Studies (2016/17)

- **Hypothesis:**
  - “Programmers are more productive in programming language X compared to programming language Y for solving problems type Z”

- **Keep all factors as constant as possible:**
  - Academic/industry backgrounds, ...
  - Solve the same problem Z once
  - In a controlled setting under our watch
  - Software environment is heavily instrumented

- **Vary only one factor: programming language X vs Y**

Controlled NetEgg user study (IRB approval was needed)
The Case for User Studies in PL Design

- Goal: identify early on the mental model of target audience and programming abstractions:

- Complementary Approaches
  - Release as open-source and measure adoption
  - Projects in large classes (>100 students): Declarative networking (SIGCOMM’11 Education workshop), Kinetic (NSDI’15)
User Study

- Q1: Can NetEgg reduce the error rate of programming SDN policies?
- Q2: Can NetEgg improve user’s productivity?
- Q3: How the users interact with NetEgg?
User Study Design

- **Object:** NetEgg VS. POX
- **Users:** masters/PhD students from the Engineering School of Penn
  - **With** Networking/Distributed System background
  - **No** SDN programming background
- **15 users in NetEgg, 9 users in POX**
  - All users in the POX group are required to be proficient in Python programming
  - No requirement for NetEgg group
- **More than 21 users in early pilot test**
Programming Assignment: Firewall++

Requirment:
1. Allow all non-IP.
2. Block all UDP traffic from the Internet.
3. Allow all UDP traffic from the CS department.
4. Allow all TCP traffic from the CS department.
5. Block all SSH traffic from the Internet.
6. TCP traffic from any host A in the Internet at TCP port P to any host B in the CS at TCP port Q is blocked, unless B sends TCP traffic using TCP port Q to A at TCP port P before.
Q1: Error Rate

POX

- 33% Correct
- 67% Incorrect

3 users

NetEgg

- 100% Correct
- 0% Incorrect

15 users
Q2: Programming Time

NetEgg achieves 50% reduction in programming time compared with POX.
Q3: User Interaction Pattern

- Monitored the number of (unique) scenarios at each call to the synthesizer

- Identified three interaction patterns:
  - Smooth interaction
  - Back-and-forth interaction
  - Stuck interaction
Smooth Interaction Pattern

Gradually adding scenarios to describe new behaviors until NetEgg converges
Back-and-forth Interaction Pattern

# of Scenarios

Time (minute)

Delete a scenario
Stuck Interaction Pattern

# of Scenarios

Time (minute)

Keep modifying the scenario
Statistics across Patterns

# Users

- Smooth Interaction: 9
- Back-and-forth: 2
- Stuck: 4

Programming Time:

- Smooth
- Back-and-forth
- Stuck
User Feedback

“the tool has a small learning curve and is quite intuitive . . . it is probably easier to use than a programming language.”

“I am very clear about what I am doing when I use the interface, and know exactly how the packets are handled under what situations.”

“(I) may get confused about how to define variables.”
Some Takeaways on User Study

- Effective way to evaluate programming abstractions from the user point of view
  - Participants found the tool intuitive to use
  - Interesting observations: variables confusion and tendency to want to tweak compiled policy tables

- Finite pool of participants → need to get the design of the study correct to maximize participants’ time

- Missing element: longitudinal analysis
Summary

- Scenario-based programming for SDN policies:
  - Expressive to program a range of policies by examples
  - Target *non-programmers* who manage networks
  - Comparable performance to hand-crafted implementations
  - User study shows evidence of improved productivity

- Other similar opportunities in networking?
  - Measurements, data-plane processing, security attack patterns …. 
Thank You!

- netdb.cis.upenn.edu/netegg
NSF Expeditions in Computer Augmented Program Engineering
excape.cis.upenn.edu

- Designer expresses “what”, using multiple input formats
- Synthesizer discovers new artifacts via integration
- Synthesizer solves computationally demanding problems using advanced analysis tools
- Interactive iterative design
- Integrated formal verification
Software-Defined Networking (SDN)

distributed protocols

<table>
<thead>
<tr>
<th>Dst</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Control plane

Data plane

e.g. POX, NOX, Floodlight, etc.

Openflow

Controller

APIs

match | action
---|---
Src=A | drop
… | …
NetEgg Scenarios: Stateful firewall

\(<\text{port}, \text{ether\_type}, \text{srcip}, \text{dstip}>\)