Maple: Simplifying SDN Programming Using Algorithmic Policies

Andreas Voellmy • Junchang Wang • Y. Richard Yang • Bryan Ford • Paul Hudak
Presented by Eldad Rubinstein • November 21, 2013
Introduction

• Looking for an abstraction for SDN
  – (specifically for OpenFlow)
• Trying to infer forwarding rules from packets
• Setup description
  – Single controller
  – Many switches
    • (also deal with TCP/IP headers)
First Try: Exact Matching

• The controller
  – Handles a packet $p$
  – Outputs a forwarding path
  – Installs rules in the switches to handle exact matching packets the same way

• Disadvantages
  – Too many packets will pass through the controller
  – Big forwarding tables (in the switches)
Maple Overview

1. Algorithmic Policy
   – Given a packet, it outputs a forwarding path
   – Arbitrary
   – Written by the user

2. Maple
   – Optimizer – infers “smart” forwarding rules
   – Scheduler – distributes work between controller cores

3. OpenFlow
   – Controller Library
   – Switches
Algorithmic Policy \( f \)

- \( f : (\text{packet, network topology}) \rightarrow \text{forwarding path} \)
- Can be written in any language (theoretically)
- Should use Maple API
  - `readPacketField : Field \rightarrow Value`
  - `testEqual : (Field, Value) \rightarrow Boolean`
  - `ipSrcInPrefix : IpPrefix \rightarrow Boolean`
  - `ipDstInPrefix : IpPrefix \rightarrow Boolean`
  - `invalidateIf : SelectionClause \rightarrow Boolean`
Algorithmic Policy $f$ (example)

```python
def f(pkt, topology):
    srcSw = pkt.switch()
    srcInp = pkt.inport()
    if locTable[pkt.eth_src()] != (srcSw, srcInp):
        invalidateHost(pkt.eth_src())
        locTable[pkt.eth_src()] = (srcSw, srcInp)
    dstSw = lookupSwitch(pkt.eth_dst())
    if pkt.tcp_dst_port() == 22:
        outcome.path = securePath(srcSw, dstSw)
    else:
        outcome.path = shortestPath(srcSw, dstSw)
    return outcome
```
Maple Optimizer

- Follows the policy execution using trace trees
  - Keeps a separate trace tree for each switch
- Compiles each trace tree into a forwarding table
- Actually it is an incremental process:

  - For each packet, a trace is augmented to the trace tree
Creating a Trace Tree

trace for packet $p$:
- $\text{test}(p, \text{tcpDst}, 22) = \text{True}$
- drop
Creating Flow Tables

• Scan the trace tree using an in-order traversal

• Emit a rule
  – For each leaf
  – For each test node ("barrier rules")

• Ordering constraint: \( r_- \rightarrow r_b \rightarrow r_+ \)

• Increase the priority after each rule
Creating Flow Tables (example)

<table>
<thead>
<tr>
<th>priority</th>
<th>match</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>tcp_dest_port = 22</td>
<td>drop</td>
</tr>
<tr>
<td>2</td>
<td>tcp_dest_port = 22</td>
<td>toController</td>
</tr>
<tr>
<td>1</td>
<td>eth_dst = 4 &amp;&amp; eth_src = 6</td>
<td>port 30</td>
</tr>
<tr>
<td>0</td>
<td>eth_dst = 2</td>
<td>drop</td>
</tr>
</tbody>
</table>

trace tree

flow table

path 6->4: h6->s1/port 30->s2/port 3->s3/port 4 (h4)
Correctness Theorems

• Trace Tree Correctness
  – Start with $t = \text{empty tree}$.
  – Augment $t$ with the traces formed by applying the policy $f$ to packets $pkt_1, \ldots, pkt_n$.
  – Then $t$ safely represents $f$. That is, if $\text{SEARCHTT}(t, pkt)$ is successful, then it has the same answer as $f(pkt)$.

• Flow Table Correctness
  – A tract tree $t$ and the flow table built from it encode the same function on packets.
Optimization I – Barrier Elimination

- Goal – emitting less rules and less priorities

\[
\text{needsBarrier}(\text{testNode } t) = \begin{cases} 
\text{true} & \text{if } t_+ \text{ is not complete and } t_- \text{ is not empty} \\
\text{false} & \text{otherwise}
\end{cases}
\]

<table>
<thead>
<tr>
<th>priority</th>
<th>match</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>tcp_dest_port = 22</td>
<td>drop</td>
</tr>
<tr>
<td>2</td>
<td>tcp_dest_port = 22</td>
<td>toController</td>
</tr>
<tr>
<td>1</td>
<td>eth_dst = 4 &amp;&amp; eth_src = 6</td>
<td>port 30</td>
</tr>
<tr>
<td>0</td>
<td>eth_dst = 2</td>
<td>drop</td>
</tr>
</tbody>
</table>

Path 6->4: h6->s1/port 30->s2/port 3->s3/port 4 (h4)
Optimization II – Priority Minimization

• Motivation – minimizing switches update algorithms running time

• Disjoint match conditions → Any ordering is possible

• First try
  – Create a DAG \( G_r = (V_r, E_r) \)
  – \( V_r \) = set of rules
  – \( E_r \) = set of ordering constraints
  – Start with setting \( priority = 0 \) for the first nodes
  – Increase the priority and continue to the next nodes
  – Works but requires two steps, not incremental
Optimization II – Priority Minimization

- Keep in mind the ordering constraint: \( r_- \rightarrow r_b \rightarrow r_+ \)
- Define a weighted DAG \( G_O = (V_O, E_O, W_O) \)
- \( V_O = \{ \text{trace tree nodes} \} \)
- \( E_O = \{ \text{all trace tree edges except } t \rightarrow t_- \}
  \quad \text{up edges – from some rule generating nodes} \)
- \( W_O = \begin{cases} 
0 & \text{for most edges} \\
1 & \text{for edges } t \rightarrow t_+ \text{ if needs a barrier} \\
1 & \text{for up edges} \end{cases} \)
Optimization II – Priority Minimization

• Work with $G_o$ while emitting rules
• Incremental build of flow tables given a new trace
  – Emit rules only where priorities have increased
Optimization III – Network-wide

• Core switches
  – are not connected to any hosts
  – they do not see “new packets”, therefore no ToController rules should be installed on them

• Route aggregation
  – Merge routes from many sources to the same destination
Multicore Scheduler

- Even after all optimizations, the controller still has a lot of work to do
- As the network grows (i.e. more switches) the controller grows as well (i.e. has more cores)
- Still more switches than cores
- *Switch level parallelism* – Each core is responsible for some switches
Results – Quality of Flow Tables

- Does Maple create efficient switch flow tables?
- Filter-based policies
  - TCP port ranges issue
  - Barrier rules issue
- \( \frac{\text{(# rules created)}}{\text{(# policy filters)}} = 0.70 \text{ to } 1.31 \)
- \( \frac{\text{(# modifications)}}{\text{(# rules created)}} = 1.00 \text{ to } 18.31 \)
Results – Flow Table Miss Rates

![Graph showing flow table miss rates vs number of packets per flow. The graph compares exact and Maple methods.](image)
Results – HTTP on real switches

![Graph showing HTTP Connection Time (ms) vs. Number Concurrent Requests (req/s)]

- **Exact**
- **Maple**
- **Native**
What is missing?

• Installing proactive rules
  – using historical packets
  – using static analysis
• Collecting statistics?
• Update consistency issues?
Summary

• SDN abstraction
• Forwarding rules are based on arriving packets
• Trying to minimize
  – Number of rules
  – Number of priorities
  – Forwarding tables miss rates
• Dealing with “real world” issues (e.g. scalability)
• Still slower then native switches
• Visit www.maplecontroller.com
Questions?