Maple: Simplifying SDN Programming Using Algorithmic Policies

Andreas Voellmy (andreas.voellmy@yale.edu)

Junchang Wang, Y. Richard Yang, Bryan Ford, Paul Hudak
Yale University
Department of Computer Science
Maple: Simplifying SDN Programming Using Algorithmic Policies

Andreas Voellmy (andreas.voellmy@yale.edu)

Junchang Wang, Y. Richard Yang, Bryan Ford, Paul Hudak
Yale University
Department of Computer Science
A Key Source of Complexity in Openflow Controllers

onPacketIn(p):
A Key Source of Complexity in Openflow Controllers

onPacketIn(p):

Step 1 examine p and decide what to do with p.
A Key Source of Complexity in Openflow Controllers

onPacketIn(p):

Step 1  examine p and decide what to do with p.

Step 2  construct and install OF rules so that similar packets are processed at switches with same action.
Simple, generic solution using exact matches

```
onPacketIn(p):

Step 1  examine p and decide what to do with p.
Step 2  insert rule with "exact match" for p, i.e. match on ALL attributes, with action determined above.
```
Simple, generic solution using exact matches

onPacketIn(p):

**Step 1** examine p and decide what to do with p.

**Step 2** insert rule with “exact match” for p, i.e. match on ALL attributes, with action determined above.

**Flow table correctness:** packet matching in flow table is indistinguishable from the packet that led to the rule being installed.
Simple, generic solution using exact matches

onPacketIn(p):

Step 1 examine p and decide what to do with p.

Step 2 insert rule with “exact match” for p, i.e. match on ALL attributes, with action determined above.

Every flow incurs flow setup delay.

Flow table correctness: packet matching in flow table is indistinguishable from the packet that led to the rule being installed.
Step 1

- p.TcpDst=22?
  - yes: drop
  - no: send to next hop for p.EthDst

Step 2
Step 1

p.TcpDst=22?

- yes
  - drop

- no
  - send to next hop for p.EthDst

Step 2

match:{TcpDst=22} action:drop
Step 1

p.TcpDst=22?

- yes: drop
- no: send to next hop for p.EthDst

Step 2

match:{TcpDst=22} action:drop

match:{EthDst=p.EthDst} action:nextHop(p)
Step 1

- p.TcpDst=22?
  - yes: drop
  - no: send to next hop for p.EthDst

Step 2

- match:{TcpDst=22} action:drop
- tcpDst!=22
  - match:{EthDst=p.EthDst} action:nextHop(p)
Step 1

- **p.TcpDst=22?**
  - yes: drop
  - no: send to next hop for p.EthDst

Step 2

- match:{TcpDst=22} action:drop
- match:{EthDst=p.EthDst} action:nextHop(p)
Step 1

- **p.TcpDst=22?**
  - yes: drop
  - no: send to next hop for p.EthDst

Step 2

- match:{TcpDst=22} action:drop
- match:{EthDst=p.EthDst} action:nextHop(p)

priority:HIGH
Step 1

- p.TcpDst=22?
  - yes: drop
  - no: send to next hop for p.EthDst

Step 2

- match:{TcpDst=22} action:drop
  - priority: HIGH
- tcpDst!=22
  - match:{EthDst=p.EthDst} action:nextHop(p)
  - priority: LOW
If p.TcpDst=22:
    insert rule
    {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
    insert rule
If p.TcpDst=22:
    insert rule
    {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
    insert rule
If p.TcpDst=22:
   insert rule
   {prio:HIGH, match:{ TcpDst=22 }, action:drop }

Else:
   insert rule
   {prio:LOW, match:{ EthDst=p.EthDst }, action:nextHop(p.EthDst) }
Switch

Controller

If p.TcpDst=22:
    insert rule
    {prio:HIGH, match:{TcpDst=22}, action:drop }

Else:
    insert rule
If p.TcpDst=22:
   insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }
Else:
   insert rule
If \( p.\text{TcpDst}=22 \):
 insert rule
 {\text{prio:HIGH}, \text{match:{TcpDst}=22}, \text{action:drop} }
Else:
 insert rule
 {\text{prio:LOW}, \text{match:{EthDst}=p.EthDst}, \text{action:nextHop(p.EthDst)}}
If $p.TcpDst=22$:  
insert rule  
{prio:HIGH, match:{TcpDst=22}, action:drop }  
Else:  
insert rule  
If p.TcpDst=22:
   insert rule
   {prio:HIGH, match:{TcpDst=22}, action:drop }

Else:
   insert rule
User Level

Step 1. Make Decisions
Step 2. Generate Rules

Under the hood

OF Controller Library

OF Switches
User Level

Step 1. Make Decisions

Step 2. Generate Rules

Under the hood

OF Controller Library

OF Switches
Algorithmic Policy

Step 2. Generate Rules

OF Controller Library

OF Switches
Algorithmic Policies
Algorithmic Policies

• Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.
Algorithmic Policies

• Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

• **Conceptually invoked on every packet entering the network**; may also access network environment state; hence it has the form:
Algorithmic Policies

• Function in a **general purpose language** that describes how a packet should be routed, **not** how flow tables are configured.

• **Conceptually invoked on every packet entering the network**; may also access network environment state; hence it has the form:

\[
f : (\text{packet} \times \text{env}) \rightarrow \text{route}
\]
Algorithmic Policies

• Function in a general purpose language that describes how a packet should be routed, not how flow tables are configured.

• Conceptually invoked on every packet entering the network; may also access network environment state; hence it has the form:

\[ f : (\text{packet } \times \text{ env}) \rightarrow \text{route} \]

• Written in a familiar language such as Java, Python, or Haskell.
Example Algorithmic Policy in Java
Example Algorithmic Policy in Java

```java
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc());
        Location dloc = e.location(p.ethDst());
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
```
Example Algorithmic Policy in Java

```java
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc);
        Location dloc = e.location(p.ethDst);
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
```

Does not specify flow table configuration.
How to implement algorithmic policies?
How to implement algorithmic policies?

- Naive solutions -- process every packet at controller or use only exact match rules -- perform poorly.
How to implement algorithmic policies?

• Naive solutions -- process every packet at controller or use only exact match rules -- perform poorly.

• Static analysis to determine layout of flow tables is possible, but has drawbacks:
  
  • Static analysis of program in general-purpose language is hard and is typically conservative.

  • System becomes source-language dependent.
Maple’s approach: runtime tracing
Maple’s approach: runtime tracing

1. Maple **observes the dependency** of $f$ on packet data.
Maple’s approach: runtime tracing

1. Maple **observes the dependency** of f on packet data.

2. Build a **trace tree (TT)**, a partial decision tree for f.
Maple’s approach: runtime tracing

1. Maple **observes the dependency** of \( f \) on packet data.

2. Build a **trace tree (TT)**, a partial decision tree for \( f \).

3. Compile flow tables (FTs) from a trace tree.
Route \( f(Packet \ p, Env \ e) \) {

if (\( p.tcpDstIs(22) \))

\[ return \] null();

else {

Location sloc =
\[ e.location(p.ethSrc()); \]

Location dloc =
\[ e.location(p.ethDst()); \]

Path path =
\[ shortestPath(\]
\[ e.links(),sloc,dloc); \]

return
\[ unicast(sloc,dloc,path); \]
}
}
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc());
        Location dloc = e.location(p.ethDst());
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
Route f(Packet p, Env e) {
  if (p.tcpDstIs(22))
    return null();
  else {
    Location sloc = e.location(p.ethSrc());
    Location dloc = e.location(p.ethDst());
    Path path = shortestPath(e.links(), sloc, dloc);
    return unicast(sloc, dloc, path);
  }
}
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc);
        Location dloc = e.location(p.ethDst);
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc());
        Location dloc = e.location(p.ethDst());
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
Route \( f(Packet \ p, \ Env \ e) \) {

\[
\text{if } (p.\text{tcpDstIs}(22))
\]

\[
\quad \text{return null();}
\]

\[
\text{else }
\]

\[
\quad \text{Location } sloc = e.\text{location}(p.\text{ethSrc}());
\]

\[
\text{Location } dloc = e.\text{location}(p.\text{ethDest}());
\]

\[
\text{Path } path = \text{shortestPath(}
\quad e.\text{links}(), sloc, dloc);
\]

\[
\quad \text{return unicast(sloc, dloc, path);}
\]
}
Route f(Packet p, Env e) {

    if (p.tcpDstIs(22))

        return null();

    else {

        Location sloc =
            e.location(p.ethSrc());

        Location dloc =
            e.location(p.ethDst());

        Path path =
            shortestPath( 
                e.links(),sloc,dloc);

        return unicast(sloc,dloc,path);

    }
}
Route f(Packet p, Env e) {

if (p.tcpDstIs(22))

    return null();

else {

    Location sloc =
        e.location(p.ethSrc());

    Location dloc =
        e.location(p.ethDst());

    Path path  =
        shortestPath(
            e.links(), sloc, dloc);

    return unicast(sloc, dloc, path);
}
}
Policy

Route \( f(Packet \ p, \ Env \ e) \) {

if (\( p.tcpDstIs(22) \))

    return null();

else {

    Location \( sloc = e.location(p.ethSrc()) \);

    Location \( dloc = e.location(p.ethDst()) \);

    Path \( path = \) shortestPath(
        e.links(), sloc, dloc);

    return unicast(sloc, dloc, path);
}

}
Policy

Route f(Packet p, Env e) {
   if (p.tcpDstIs(22))
      return null();
   else {
      Location sloc = e.location(p.ethSrc());
      Location dloc = e.location(p.ethDst());
      Path path = shortestPath(e.links(), sloc, dloc);
      return unicast(sloc, dloc, path);
   }
}
Route f(Packet p, Env e) {
    if (p.tcpDstIs(22))
        return null();
    else {
        Location sloc = e.location(p.ethSrc());
        Location dloc = e.location(p.ethDst());
        Path path = shortestPath(e.links(), sloc, dloc);
        return unicast(sloc, dloc, path);
    }
}
Route \( f(\text{Packet } p, \text{Env } e) \) {

if \( (p.\text{tcpDstIs}(22)) \)

return null();

else {

Location \( sloc = \) e.location\( (p.\text{ethSrc}()) \);

Location \( dloc = \) e.location\( (p.\text{ethDst}()) \);

Path \( path = \) shortestPath\( (e.\text{links}(), sloc, dloc) \);

return unicast\( (sloc, dloc, path) \);
}
}
Policy

Route f(Packet p, Env e) {

    if (p.tcpDstIs(22))
        return null();
    else {

        Location sloc =
            e.location(p.ethSrc());

        Location dloc =
            e.location(p.ethDst());

        Path path =
            shortestPath(
                e.links(), sloc, dloc);

        return unicast(sloc, dloc, path);
    }
}
Compile recorded executions into flow table

- tcpDst==22
  - True: drop
  - False: ethDst
    - ethDst: 2
    - ethSrc: 4
      - drop
      - port 30
Compile recorded executions into flow table

- Compile recorded executions into flow table

- tcpDst==22
  - True: drop
  - False: ethDst
    - 2: drop
    - 4: ethSrc
      - 6: port 30

Compile recorded executions into flow table

match:{tcpDst!=22,ethDst:4,ethSrc:6}
Compile recorded executions into flow table

- barrier rule:
  match:{tcpDst==22}

- match:{ethDst:4,ethSrc:6}

Priority
Compile recorded executions into flow table

- **Match:** tcpDst==22
  - **True:** drop
  - **False:** ethDst=4, ethSrc=6
    - **Port 30**

**Barrier Rule:**
match:{tcpDst==22}
match:{ethDst:4, ethSrc:6}

**Priority:**
Compile recorded executions into flow table

```
barrier rule:
match:{tcpDst==22}

match:{ethDst:4,ethSrc:6}
```

Priority
Compile recorded executions into flow table

```
match:{tcpDst==22}
drop

match:{ethDst:4, ethSrc:6}
drop

priority: 1
```

**Barrier rule:**
match:{tcpDst==22}
Compile recorded executions into flow table

Priority

barrier rule:
match:{tcpDst==22}

match:{ethDst:4,ethSrc:6}

port 30
Compile recorded executions into flow table

Priority

**barrier rule:**
match:{tcpDst==22}

match:{ethDst:4,ethSrc:6}

```
match:{tcpDst==22}
```

```
match:{ethDst:4,ethSrc:6}
```

```python
Priority

3

drop

2

tcpDst==22

True

2

False

1

drop

ethSrc

4

ethDst

2

port 30
```
Compile recorded executions into flow table

```
match:{tcpDst==22}
action:ToController
```

 Barrier rule:

```
match:{tcpDst==22}
action:ToController
```

Priority
Basic compilation: in-order traversal & barrier rules

tcpDst==22

null

ethDst

port 30

null

ethSrc

Tuesday, August 13, 13
Basic compilation: in-order traversal & barrier rules

Priority := 0

tcpDst==22

null

ethDst

null

2

4

ethSrc

port 30

2

4

6

true

False
Basic compilation: in-order traversal & barrier rules

Priority := 0

tcpDst==22

null

ethDst

2

4

null

ethSrc

port 30
Basic compilation: in-order traversal & barrier rules

Priority := 0

tcpDst==22

accumulated match: {}

ethDst

null

null

ethSrc

port 30
Basic compilation: in-order traversal & barrier rules

Priority := 0

![Diagram showing basic compilation with in-order traversal and barrier rules.](image)
Basic compilation: in-order traversal & barrier rules

Priority := 0

$$\text{tcpDst} = 22$$

Priority := 0

$$\text{ethDst}$$

Priority := 0

$$\text{ethSrc} = 6$$

Priority := 0

Port 30

Priority := 0

$$\text{null}$$

Priority := 0

Match: $$\{\}$$
Basic compilation: in-order traversal & barrier rules

Priority := 0

 tcpDst==22

  True
     null

  False
    ethDst
        {}
    2
     null
    4
     ethSrc
        {}
        {ethDst:4}
    6
     port 30
        {ethDst:4, ethSrc:6}

(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
Basic compilation: in-order traversal & barrier rules

Priority := 1

tcpDst==22

null

accumulated match: {}

ethDst

null

ethSrc

port 30

(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
Basic compilation: in-order traversal & barrier rules

\[
\text{tcpDst} == 22 \quad \text{match: \{\}} \\
\text{Priority} := 1
\]

\[
\text{null} \quad \text{True} \\
\text{False} \quad \text{ethDst} \quad \{\}
\]

\[
\{\text{ethDst:2}\} \quad \text{null} \\
\text{ethSrc} \quad \{\text{ethDst:4}\} \\
\text{port 30} \quad \{\text{ethDst:4, ethSrc:6}\}
\]

\[
(prio:0,\{\text{ethDst:4, ethSrc:6}\},\text{action: [port 30]})
\]
Basic compilation: in-order traversal & barrier rules

\[ \text{tcpDst} = 22 \]

Priority := 1

(\text{prio}:=0, \{\text{ethDst}:4, \text{ethSrc}:6\}, \text{action}: [\text{port } 30])

(\text{prio}:=1, \{\text{ethDst}:2\}, \text{action}: \text{drop})

null
Basic compilation: in-order traversal & barrier rules

Priority := 2

tcpDst==22

null

ethDst

2

null

{ethDst:2}

(ethDst:2), action: drop

4

port 30

(ethDst:4, ethSrc:6), action: [port 30]

(ethDst:4)

{ethDst:4}

(ethSrc, ethDst:6)

{ethDst:0, ethSrc:6}

(ethDst:6)

{ethSrc:4, ethDst:6}

{ethDst:4}

null

{ethDst:0}

match: {}

accumulated
Basic compilation: in-order traversal & barrier rules

Priority := 2

barrier rule: (prio:2, {tcpDst:22}, action: ToController)
(prio:1, {ethDst:2}, action: drop)
(prio:0, {ethDst:4, ethSrc:6}, action: [port 30])
Basic compilation: in-order traversal & barrier rules

Priority := 3

barrier rule: (prio:2, {tcpDst:22}, action:ToController)
(prio:1, {ethDst:2}, action:drop)
(prio:0, {ethDst:4, ethSrc:6}, action:[port 30])
Basic compilation: in-order traversal & barrier rules

Priority := 3

barrier rule: (prio:2,{tcpDst:22},action:ToController)  
(prio:1,{ethDst:2},action:drop)  
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
Basic compilation: in-order traversal & barrier rules

Priority := 3

**barrier rule:** (prio:2,{tcpDst:22},action:ToController)

(prio:1,{ethDst:2},action:drop)

(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
Basic compilation example result

(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
Basic compilation example result

(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])

• Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.
Basic compilation example result

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.

- Deficiencies:

  ```
  (prio:3, {tcpDst:22}, action: drop)
  (prio:2, {tcpDst:22}, action: ToController)
  (prio:1, {ethDst:2}, action: drop)
  (prio:0, {ethDst:4, ethSrc:6}, action: [port 30])
  ```
Basic compilation example result

(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.

- Deficiencies:
  - More rules than necessary.
Basic compilation example result

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.

- Deficiencies:
  - More rules than necessary.
  - More priorities levels than necessary.

Can use priority 0

\[
\begin{align*}
\text{(prio:3,} & \text{tcpDst:22,} \text{action:drop)} \\
\text{(prio:2,} & \text{tcpDst:22,} \text{action:ToController)} & \text{No effect} \\
\text{(prio:1,} & \text{ethDst:2,} \text{action:drop)} \\
\text{(prio:0,} & \text{ethDst:4, ethSrc:6,} \text{action:[port 30])}
\end{align*}
\]
Basic compilation example result

- Trace tree method converts arbitrary algorithmic policies into correct forwarding tables that effectively use wildcard rules.

- Deficiencies:
  - More rules than necessary.
  - More priorities levels than necessary.

- We **annotate TT nodes** with extra information to improve compilation.

```
(prio:3,{tcpDst:22},action:drop)
(prio:2,{tcpDst:22},action:ToController)
(prio:1,{ethDst:2},action:drop)
(prio:0,{ethDst:4, ethSrc:6},action:[port 30])
```

Can use priority 0 → No effect
Optimization 1: Annotate TT nodes with completeness

- `tcpDst==22`
- `ethDst 2`
- `drop`
- `port 30`
- `ethSrc 6`
Optimization 1: Annotate TT nodes with completeness

\[
\begin{align*}
\text{tcpDst} &= 22 \\
\text{ethDst} &= 2 \\
\text{drop} &= \text{True}
\end{align*}
\]

\[
\begin{align*}
\text{complete} = \{(\text{tcpDst:22}), \{\text{ethDst:2}, \text{ethSrc:6}\}\}
\end{align*}
\]
Optimization 1: Annotate TT nodes with completeness

**tcpDst==22**

- **True**
  - {tcpDst:22} drop
  - (prio:2, {tcpDst:22}, action: drop)

- **False**
  - ethDst
    - {ethDst:2}
      - drop
      - (prio:1, {ethDst:2}, action: drop)
  - ethSrc
    - {ethDst:4}
      - port 30
      - (prio:0, {ethDst:4, ethSrc:6}, action: [port 30])
Optimization 1: Annotate TT nodes with completeness

\[
\begin{align*}
&\text{tcpDst}==22 \\
&\text{True} \\
&\text{drop} \\
&\{\text{tcpDst}:22\} \\
&\text{complete} \\
&(prio:2,\{\text{tcpDst}:22\},\text{action:drop})
\end{align*}
\]

\[
\begin{align*}
&\text{False} \\
&\text{ethDst} \\
&\{\text{ethDst}:2\} \\
&\text{complete} \\
&(prio:1,\{\text{ethDst}:2\},\text{action:drop})
\end{align*}
\]

\[
\begin{align*}
&\text{False} \\
&\text{ethDst} \\
&\{\text{ethDst}:4\} \\
&\{\text{ethDst}:4,\text{ethSrc}:6\} \\
&\text{complete} \\
&(prio:0,\{\text{ethDst}:4,\text{ethSrc}:6\},\text{action:port 30})
\end{align*}
\]
Optimization 1: Annotate TT nodes with completeness

 tcpDst==22

 True

 {tcpDst:22}

 drop

 complete

 (prio:2,{tcpDst:22},action:drop)

 False

 no barrier

 {ethDst:2}

 drop

 complete

 (prio:1,{ethDst:2},action:drop)

 {ethDst:4, ethSrc:6}

 port 30

 complete

 (prio:0,{ethDst:4, ethSrc:6},action:[port 30])

 (prio:1,{ethDst:4}, action: drop)

 {ethDst:4}
Optimization 2: Annotate nodes with priority dependencies

- tcpDst==22
  - True: {tcpDst:22} → drop
  - False: ethDst
    - {ethDst:2} → drop
    - ethSrc
      - 2: {ethDst:4} → port 30
      - 4: {ethDst:4, ethSrc:6}
Optimization 2: Annotate nodes with priority dependencies

- tcpDst==22: drop (True)
- ethDst: drop (False)
- ethSrc: port 30 (False, ethSrc:6)
- ethDst:4: {} (False, ethDst:4, ethSrc:6)

{tcpDst:22}
Optimization 2: Annotate nodes with priority dependencies

- tcpDst==22
  - True
    - {tcpDst:22} drop
  - False
    - ethDst
      - 2
      - {ethDst:2} drop
    - ethSrc
      - 6
      - (prio:0, {ethDst:2}, action: drop)
    - (prio:0, {ethDst:2, ethSrc:6}, action: port 30)
    - (prio:0, {ethDst:4, ethSrc:6}, action: [port 30])

- ethDst
  - {ethDst:4}
    - 4
    - (prio:0, {ethDst:4}, action: drop)
    - (prio:0, {ethDst:4, ethSrc:6}, action: [port 30])
Optimization 2: Annotate nodes with priority dependencies

- **tcpDst==22**
  - True:
    - {tcpDst:22} drop
  - False:
    - ethDst:2 drop
    - ethDst:2, ethSrc:6 drop
    - ethSrc:4, ethDst:4, action:[port 30]
    - ethSrc:6, action:[port 30]
Optimization 2: Annotate nodes with priority dependencies

- tcpDst==22
  - False
  - True

1. {tcpDst:22}  drop
   - (prio:1, {tcpDst:22}, action: drop)

2. {ethDst:2}  drop
   - (prio:0, {ethDst:2}, action: drop)

3. ethDst  
   - {}  
   - {ethDst:4}  
   - 2

4. ethSrc  
   - {}  
   - {ethDst:4, ethSrc:6}  
   - 6

5. ethSrc  
   - {}  
   - {ethDst:4}  
   - 4

6. port 30  
   - {ethDst:4, ethSrc:6}  
   - (prio:0, {ethDst:4, ethSrc:6}, action: [port 30])
Improved compilation result

(\text{prio:1}, \{\text{tcpDst:22}\}, \text{action:drop})
(\text{prio:0}, \{\text{ethDst:2}\}, \text{action:drop})
(\text{prio:0}, \{\text{ethDst:4, ethSrc:6}\}, \text{action:[port 30]})
Maple Status

• Maple has been implemented in Haskell using the McNettle Openflow controller, which implements Openflow 1.0.

• The implementation includes several other features:

  • **Incremental TT compilation**, to avoid full recompilation on every update.
  
  • **Trace reduction**, to ensure traces and trace trees do not contain redundant nodes.

  • **Automatic and user-specified invalidation**, to support removing and updating TT and FT when network state changes.
Maple use case: ACL compiler
Maple use case: ACL compiler

• Many networks use filter sets, e.g. for access control, to check IP addresses and port ranges.

  • How can we implement these using OpenFlow flow tables?

  • How can we combine filter sets with other policies, e.g. routing, load balancing?
Maple use case: ACL compiler

• Many networks use filter sets, e.g. for access control, to check IP addresses and port ranges.
  
  • How can we implement these using Openflow flow tables?
  
  • How can we combine filter sets with other policies, e.g. routing, load balancing?
  
• We implement a filter set interpreter as an algorithmic policy in Maple:
  
  • Loads a filter set from a file at startup.
  
  • When given a packet, \( f \) iterates over filters in filter set, testing if packet satisfies filter; return action based on first matching filter.
Maple generates compact flow tables

- Classbench filter sets and packet traces for 3 filter set types - ACL, FW, IPC - and two sizes - 1000 and 2000 filters per filter set.

- blue = # filters activated by packets in trace; green + blue = # rules generated by Maple for packet trace.

- More generated rules expected, since TCP port ranges expanded to exact matches.
Maple uses few priority levels

- blue = # priority levels used by Maple-generated rules.

- **acl1a**: 9 priorities for nearly 1000 rules.

- **acl2b**: 120 priorities used for 2000 rules.
Maple reduces HTTP connection time

• 3 real HP 5406 switches, client on one side, server on the other.

• Client performs X HTTP reqs/sec. using httperf.

• Measure average HTTP connection time as X varies.

• Compare exact & maple
  • exact: up to 282 ms average
  • maple: 1-2 ms: 100x reduction.
Related Work

• FML, FSL: logic-based policy languages.

• Frenetic family of languages: Frenetic, NetCore, Pyretic offer more declarative approaches.

• Onix: introduces NIB abstraction to abstract distributed flow tables.

• Openflow controllers: Maestro, Beacon, NOX, NOX-MT, POX (and many more).
Summary: Contributions

• Algorithmic policies provide a simple, expressive programming model for SDN, eliminating a key source of errors and performance problems.

• Maple provides a scalable implementation of algorithmic policies through several novel techniques, including:

  • runtime tracing of algorithmic policies,
  • maintaining a trace tree and compiling TT to flow tables to distribute processing to switches;
  • using TT annotations to implement compiler optimizations such as rule and priority reductions.