Magellan: Automatic SDN Pipelining from Algorithmic Policies

Presenter: Qiao Xiang

Work by S. Chen, A. Voellmy, T. Wang, R. Yang*

Systems Networking Lab (SNLab)

June 3, 2016

Authors are ordered alphabetically.

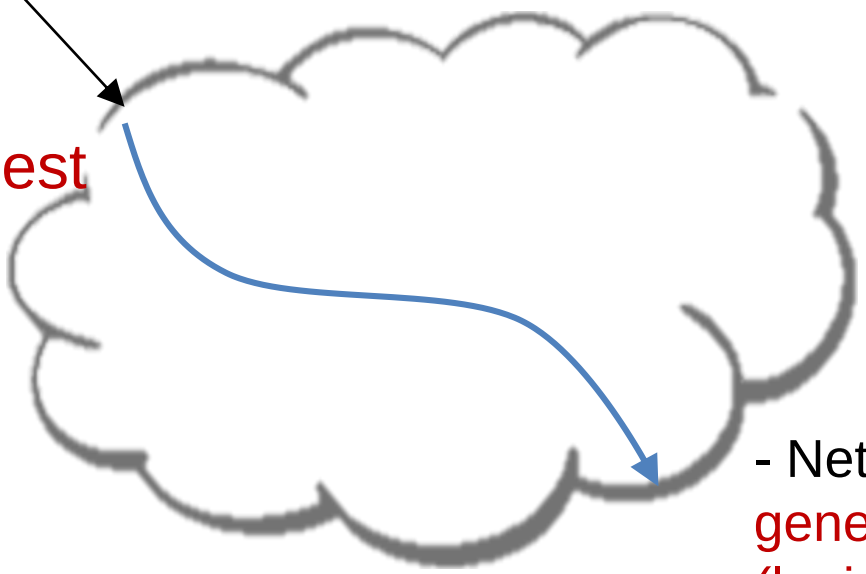
Outline

- Background: algorithmic SDN programming
- Maple
- Magellan
- Summary

Background: High-Level Algorithmic SDN Programming

Goal: Can we let programmers write the most obvious SDN code?

consider each pkt as a request



- Network control expressed in general purpose language, (logically) invoked on each pkt

- A network control function returns how a pkt traverses network, not how datapath (flow tables) are configured.

Example Algorithmic Policy in Java

```
Route f(Packet p) {
  if (p.tcpDstIs(22)) return null();
  else {
    Location sloc = hostTable(p.ethSrc());
    Location dloc = hostTable(p.ethDst());
    Route path    = myRoutingAlg(topology(),
                                 sloc,dloc);
    return path;
  }
}
Route myRoutingAlg(Topology topo,
                   Location sLoc, Location dloc) {
   if ( isSensitive(sLoc) || isSensitive(dLoc) )
      return secureRoutingAlg(topo, sloc, dloc);
   else
      return standardRoutingAlg(topo, sloc, dloc);
}
```

Does not specify anything on flow tables!

Challenge

- Naïve solution of processing each packet at controller is not possible

- Key challenge: How to use data-path (flow tables) from data-path oblivious algorithmic policies?

Outline

- Background: algorithmic SDN programming
- Maple: dynamic tracing

Maple: Basic Idea

- There are two representations of computation
  - A sequence of instructions
  - Memorization tables

- Although the decision function f does **not** specify how flow tables are configured, if for a given decision (e.g., drop), we know the dependency of the decision, we can construct the flow tables (aka, memorization tables).

Maple: Realizing the Basic Idea

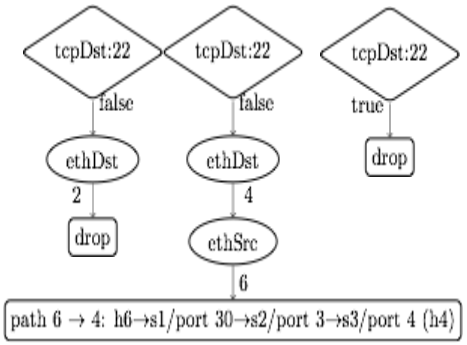
- Only requirement: Program f uses a simple library to access pkt attributes:

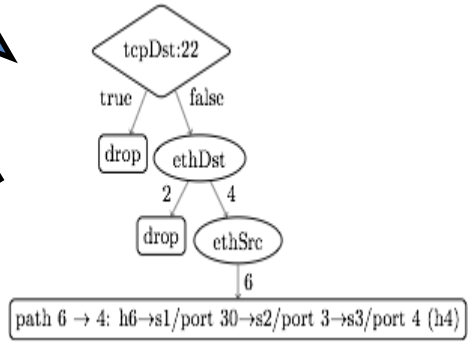
```
readPacketField :: Field -> Value
testEqual       :: (Field,Value) -> Bool
ipSrcInPrefix   :: IPPrefix -> Bool
ipDstInPrefix   :: IPPrefix -> Bool
```

- Library provides both convenience and more importantly, decision dependency!

Dynamic Tracing: Abstraction to Flow Tables

1. **Observes decision dependency** of f on pkt attributes.





Prio	Match	Action
1	tcpDst:22	ToControl
0	ethDst:2	discard
0	ethDst:4,	port 30

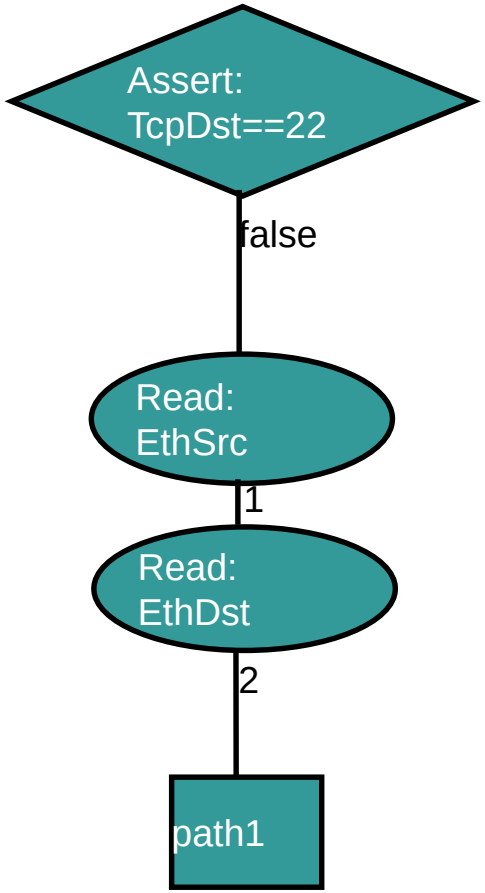
2. Builds a **trace tree (TT)**, a universal (general), partial decision tree representation of any f.

3. **Compile** trace tree to generate flow tables (FTs).

Policy

EthSrc:1,
EthDst:2,
TcpDst:80

```
Route f(Packet p) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      hostTable(p.ethSrc());

    Location dloc =
      hostTable(p.ethDst());


    Route path    =
      myRoutingAlg(
        topology(),sloc,dloc);
    return path;
  }
}
```

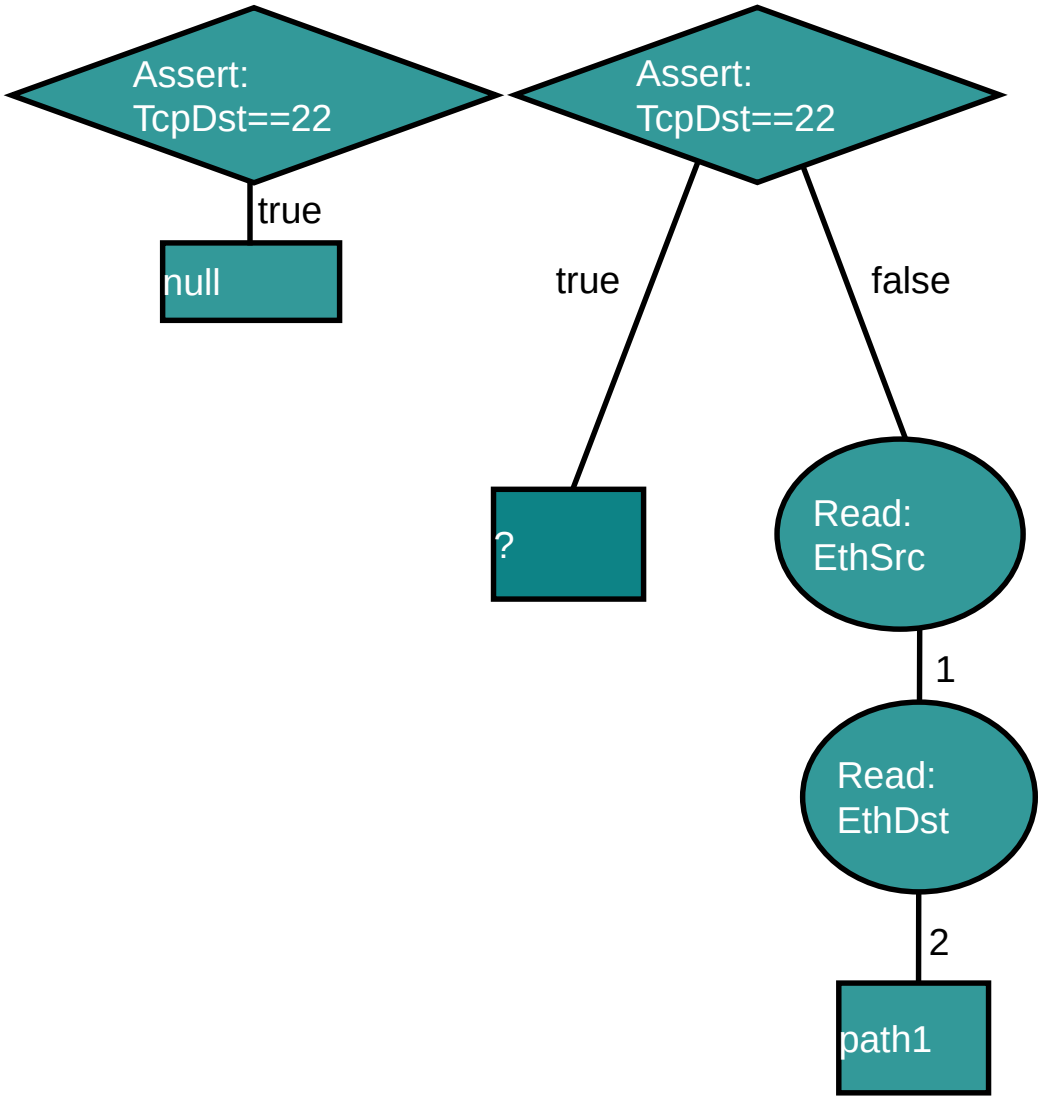


Policy

Trace Tree

EthDst:1,
TcpDst:22

```
Route f(Packet p) {

  if (p.tcpDstIs(22))

    return null();

  else {

    Location sloc =
      hostTable(p.ethSrc());

    Location dloc =
      hostTable(p.ethDst());

    Route path    =
      myRoutingAlg(
        topology(),sloc,dloc);
    return path;
  }
}
```

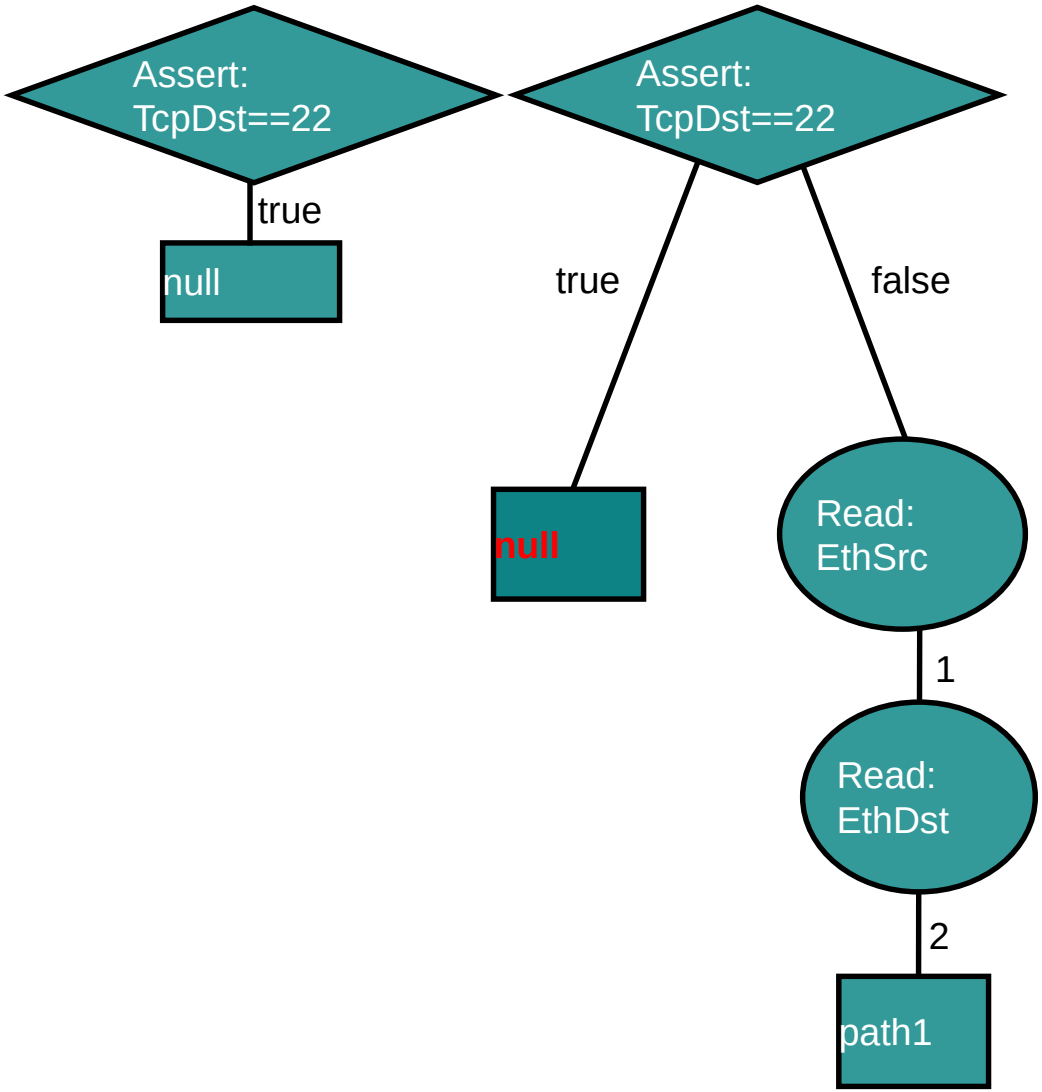


Policy

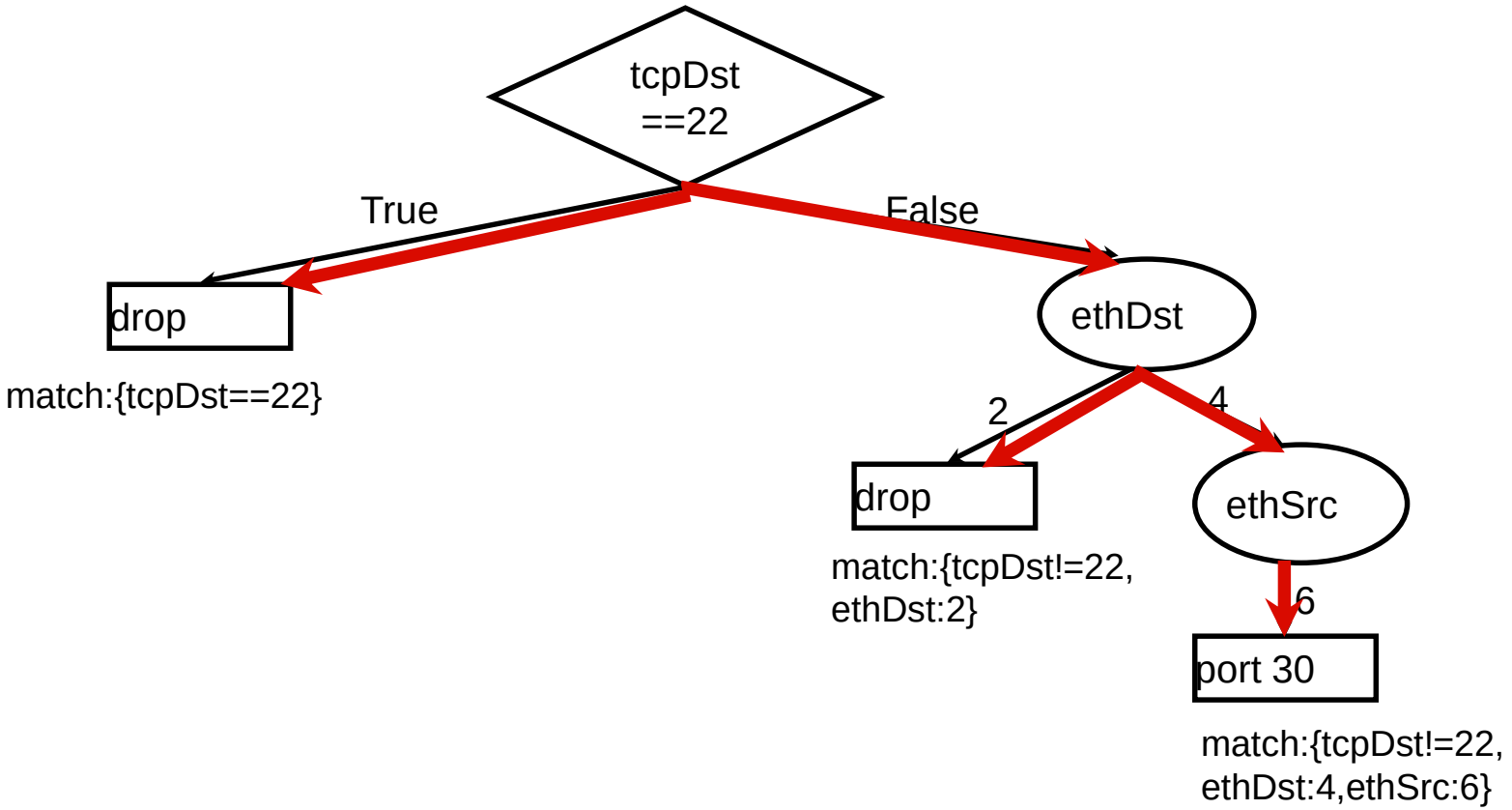
Trace Tree

EthDst:1,
TcpDst:22

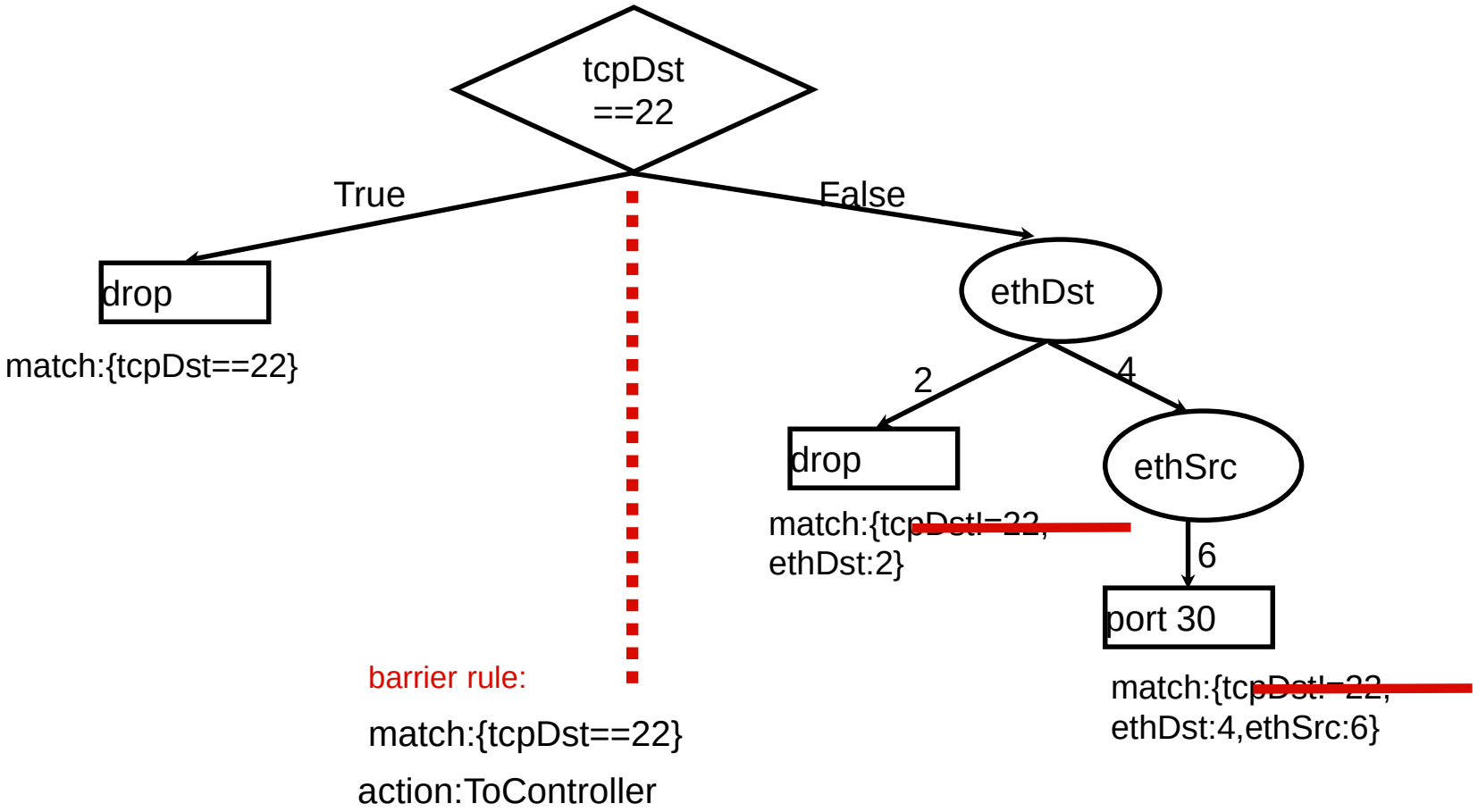
```
Route f(Packet p) {

  if (p.tcpDstIs(22))


    return null();

  else {

    Location sloc =
      hostTable(p.ethSrc());


    Location dloc =
      hostTable(p.ethDst());


    Route path    =
      myRoutingAlg(
        topology(),sloc,dloc);
    return path;
  }
}
```



Trace Tree => Flow Table

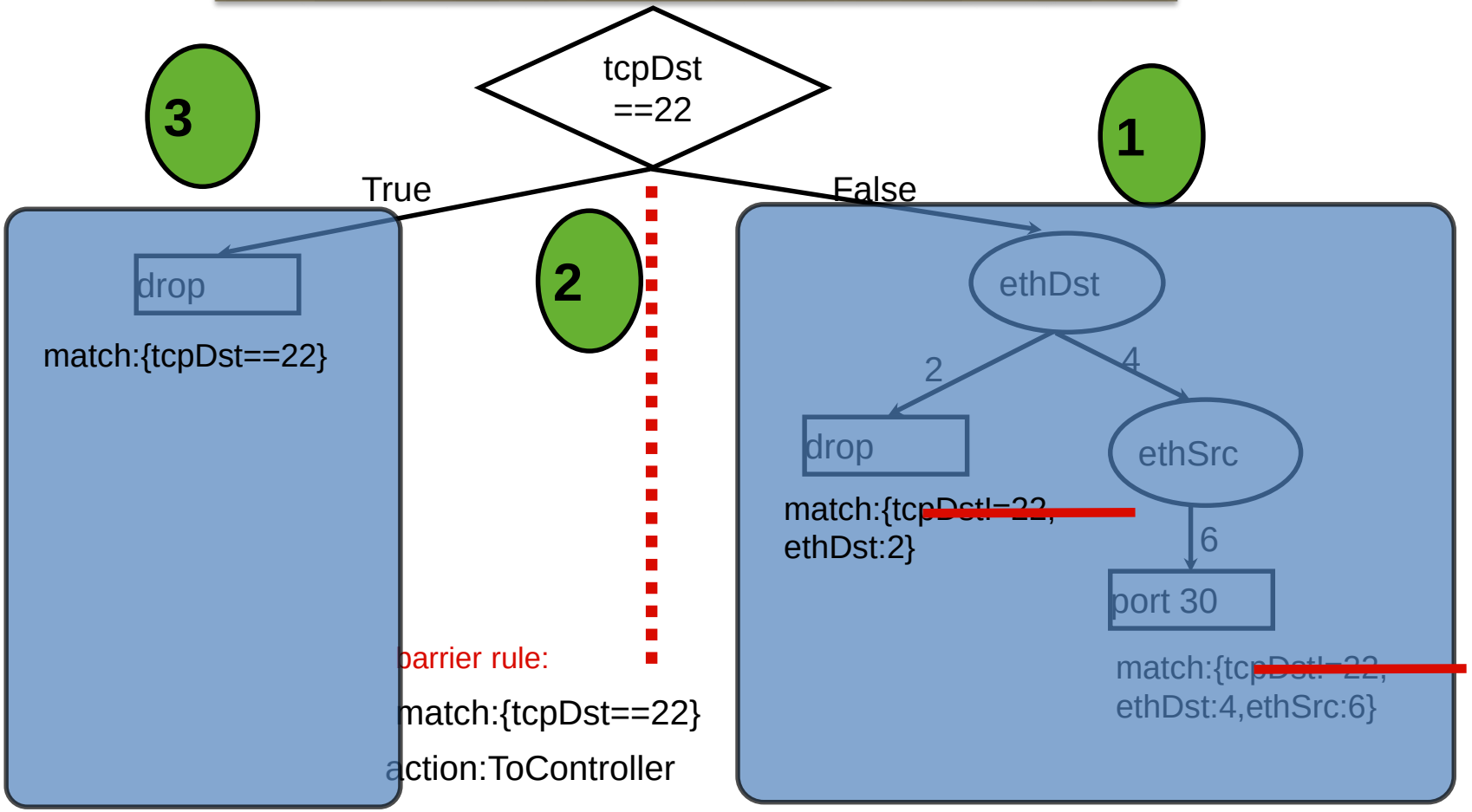


Trace Tree => Flow Table



Trace Tree => Flow Table

Simple, classical in-order tree traversal generates flow table rules!



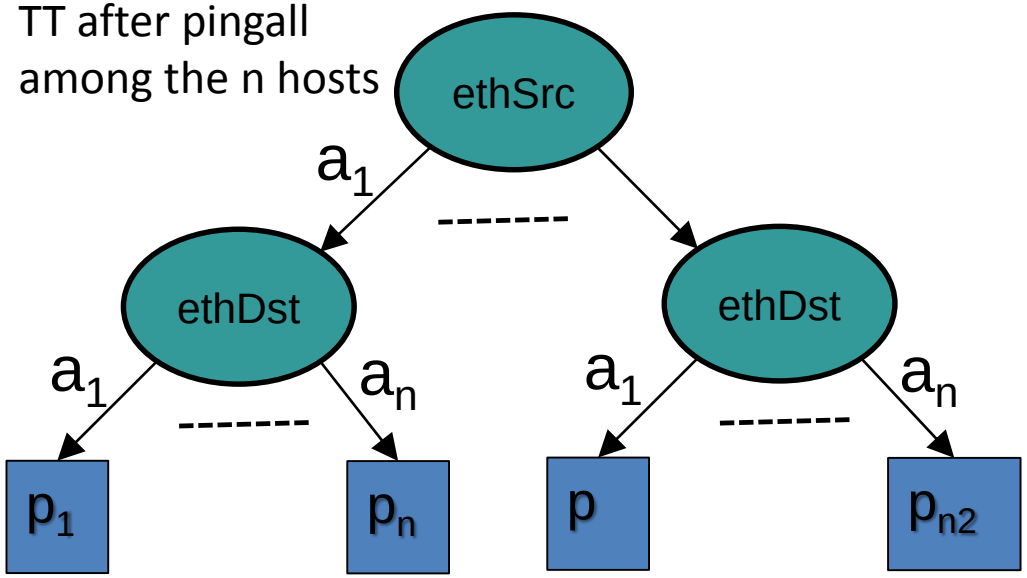
Problems of Maple Trace Tree

- Quality: Compiles to only a single flow table
- Latency: A reactive approach that waits for punted packets to begin unfolding the trace tree and generating rules

Why is Multi-Table Important for Quality (A Simple GBP Example)?

```
Map<MAC, ConditionSet>    hostTable;
0. Route onPacketIn(Packet p) {
1.   ConditionSet srcCond = hostTable.get( p.ethSrc() );
2.   ConditionSet dstCond = hostTable.get( p.ethDst() );
3.   if (srcCond != null && dstCond != null
        && pass(srcCond, dstCond) )
4.     return port1;
5.   else
6.       return drop; }
```

- Assume n hosts in hostTable
- TT after pingall among the n hosts



Flow table from trace tree

ethSrc	ethDst	Action
a_1	a_1	p_1
a_1	a_2	p_2
..	…	…
a_n	a_n	p_{n2}

n^2 entries; more if under attacks

More Efficient Multi-Table (2 Tables) Design

Table 1

ethSrc	Action
a_1	$reg_{srcCond}=y_1$ jump 2
a_2	$reg_{srcCond}=y_2$ jump 2
..	…
a_n	$reg_{srcCond}=y_n$ jump 2
otherwise	drop

Table 2

$regs_{rcSw}$	ethDst	Action
y_1	a_1	$p_{1,1}$
y_1	a_2	$p_{1,2}$
..	…	…
y_k	a_n	$p_{k,n}$
otherwise		drop

Assume k condition possibilities.

n + kn entries

More Efficient Multi-Table (3 Tables) Design

Table 1

ethSrc	Action
a_1	$reg_{srcCond}=y_1$ jump 2
a_2	$reg_{srcCond}=y_2$ jump 2
..	…
a_n	$reg_{srcCond}=y_n$ jump 2
otherwise	drop

Table 2

ethDst	Action
a_1	$reg_{dstCond}=y_1$ jump 3
a_2	$reg_{dstCond}=y_2$ jump 3
..	…
a_n	$reg_{dstCond}=y_n$ jump 3
otherwise	drop

Assume k condition possibilities.

Table 3

$reg_{srcCond}$	$regs_{dstCond}$	Action
y_1	y_1	$p_{1,1}$
y_1	y_2	$p_{1,2}$
..	…	…
y_k	y_n	$p_{k,k}$
otherwise		drop

$2n + k^2$ entries

Comparison of 3 Designs

Assume n = 4000, k = 100

Design	#flow rules
1 table	16,000,000 = 16M
2 tables	4000+400,000 = 404K
3 tables	8000+10,000 = 18K

Outline

- Background: algorithmic SDN programming
- Maple
- Magellan: automatic SDN pipelining

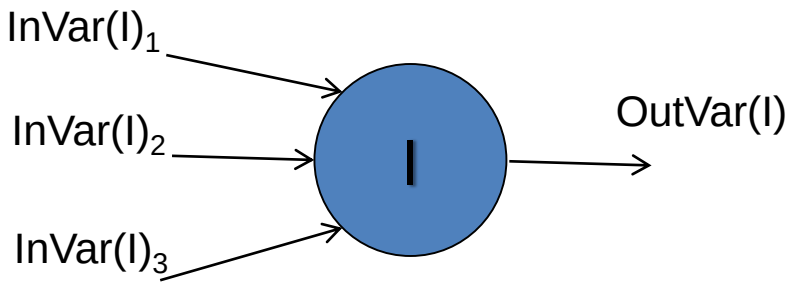
Magellan: Basic Idea

- Basic idea:
  - Trace tree is a mostly blackbox approach, while Magellan starts with the other extreme---a whitebox approach.
  - Proactively explore the program and generate flow tables

Basic Insight: Per-Instruction Table (PIT)

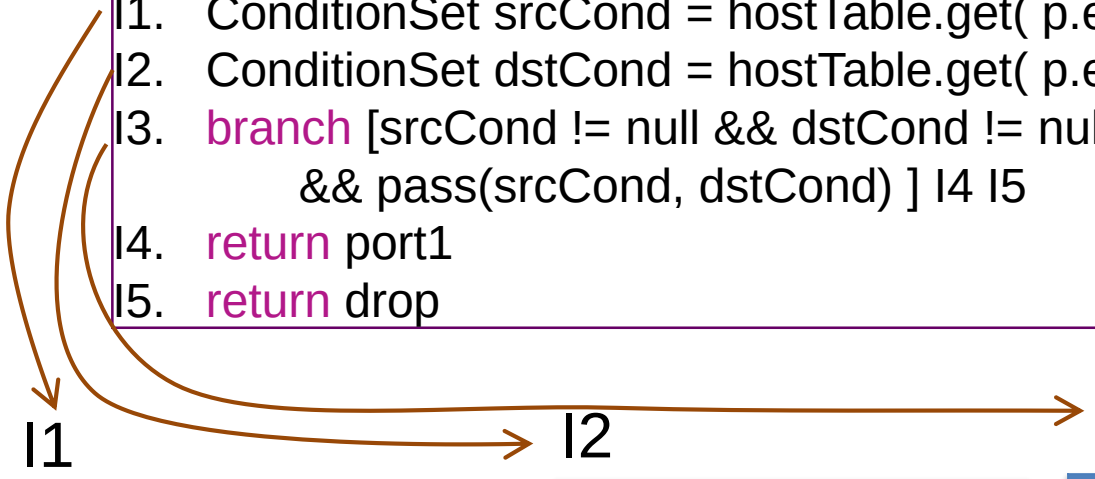
- Function f consists of a sequence of instructions $I_1, I_2, \ldots, I_N$
- One can consider each instruction I a table: a mapping from input variable states to output variable states, represented as a table

$InVar(I)_1$	$InVar(I)_2$	$InVar(I)_3$	OutVar(I)
1	1	1	OutVar(I)=I(1,1,1)
…			
…			



Example

```
Map<MAC, ConditionSet>    hostTable;
   Route onPacketIn(Packet p) {
I1.   ConditionSet srcCond = hostTable.get( p.ethSrc() );
I2.   ConditionSet dstCond = hostTable.get( p.ethDst() );
I3.   branch [srcCond != null && dstCond != null
            && pass(srcCond, dstCond) ] I4 I5
I4.   return port1
I5.   return drop
```



p.ethSrc	Action
1	$Reg_{srcCond}$ =$srcCond_1$ jump I2
2	
…	
2^{48}	$Reg_{srcCond}$ =$srcCond_{2^{\wedge}48}$ jump I2

p.ethDst	Action
1	$Reg_{dstCond}$ =$dstCond_1$ jump I3
2	
…	
2^{48}	$Reg_{dstCond}$ =$dstCond_{2^{\wedge}48}$ jump I3

$reg_{srcCond}$	$reg_{dstCond}$	Action
$srcCond_1$	$dstCond_1$	jump I4
…		jump I5
…		

Problems of PIT

- Too large table size: Naïve construction of each instruction table is still not practical
  - Ins(var_1, $var_{2, \ldots,}$ var_N) has $|var_1|$ x $|var_2|$…x $|var_N|$ rows, where $|var_i|$ is the potential values of var_i
- Too many tables: a switching element allows only a small number of flow tables, and a program may have many more instructions

Outline

- Background: algorithmic SDN programming
- Maple
- Magellan
  - Basic idea
  - Reduce table size: Compact-mappable instructions

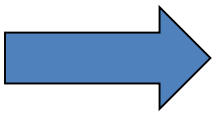
Reduce Table Size: Compact-Mappable (CM) Instructions

Table construction does not consider available state info: only the n values in current hostTable state are needed. Hence table size should be n+1, not 2^{48}

We say I1 is a compact-mappable (CM) statement.

I1

p.ethSrc	Action
1	$Reg_{srcCond}$ =$srcCond_1$ jump I2
2	
…	
2^{48}	$Reg_{srcCond}$ =$srcCond_{2^{\wedge}48}$ jump I2



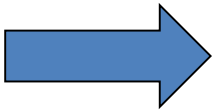
p.ethSrc	Action
a_1	$reg_{srcCond}$=$srcCond_{a1}$ jump I2
a_2	
..	…
a_n	$reg_{srcCond}$=$srcCond_{an}$ jump I2
otherwise	$reg_{srcCond}$=null

I1. ConditionSet srcCond = hostTable.get(p.ethSrc());

More Examples of CM Instructions

y = p.ethSrc == p.ethDst

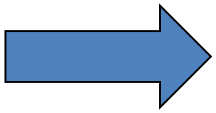
p.ethSrc	p.ethDst	Action
1	1	y=false
1	2	y=true
…	…	…
2^{48}	2^{48}	y=true



p.ethSrc	p.ethDst	Action
***0	***1	false
***1	***0	false
**0*	**1*	false
**1*	**0*	False
…		
*	*	true

y = p.ethSrc() > m

p.ethSrc	Action
1	y=false
…	
m	y=false
m+1	y=true
…	…
2^{48}	y=true

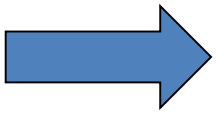


p.ethSrc	Action
…	…

More Examples of CM Instructions

y = p.ethSrc() > m

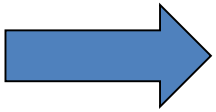
p.ethSrc	Action
1	y=false
…	
m	y=false
m+1	y=true
…	…
2^{48}	y=true



p.ethSrc	Action
…	…

y = p.ethSrc != p.ethDst

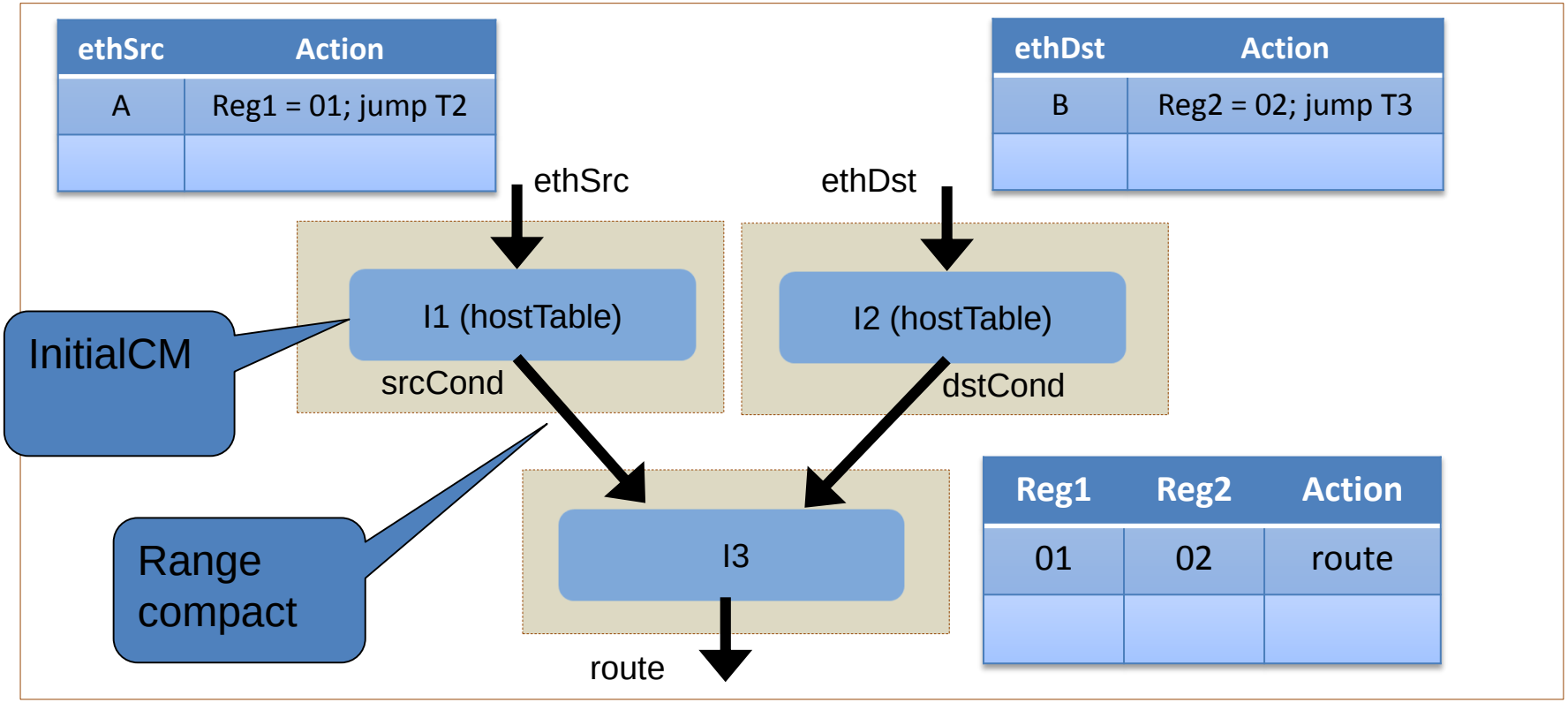
p.ethSrc	p.ethDst	Action
1	1	y=false
1	2	y=true
…	…	…
2^{48}	2^{48}	y=true



p.ethSrc	p.ethDst	Action

CM Propagation through Data-Flow

```
Map<MAC, ConditionSet>    hostTable;                                  Instructions
   Route onPacketIn(Packet p) {
I1.   ConditionSet srcCond = hostTable.get( p.ethSrc() );
I2.   ConditionSet dstCond = hostTable.get( p.ethDst() );
I3.   branch [srcCond != null && dstCond != null
            && pass(srcCond, dstCond) ] I4 I5
```



CM Propagation through Data-Flow

```
Map<MAC, ConditionSet>    hostTable;
   Route onPacketIn(Packet p) {
I1.   ConditionSet srcCond = hostTable.get( p.ethSrc() );
I2.   ConditionSet dstCond = hostTable.get( p.ethDst() );
I3.   branch [srcCond != null && dstCond != null
            && pass(srcCond, dstCond) ] I4 I5
I4.   return port1
I5.   return drop
```

I1

p.ethSrc	Action
a_1	$srcCond_{a1}$ jump I2
a_2	
...	
a_n	$srcCond_{an}$ jump I2

I2

p.ethDst	Action
a_1	$dstCond_{a1}$ jump I3
a_2	
...	
a_n	$dstCond_{an}$ jump I3

I3

srcCond	dstCond	Action
$srcCond_1$	$dstCond_1$	y=true; jump I4
...		y=false; jump I5
...		

Only output in I1/I2 tables are needed as input to I3 table input

Outline

- Background: algorithmic SDN programming
- Maple
- Magellan
  - Basic idea
  - Reduce table size: Compact table mapping
  - Bound #tables: Table design w/ bound on #tables

Problem Formulation

- Given a fine-grained flow table pipeline with M tables, compute min size new pipeline w/ #tables < bound M
- Key issue: combining two tables, one matching on $attr_1$ and another on $attr_2$ can lead to combination explosion

A Naïve Algorithm

- Consider the table-design problem as a graph partition problem, with #partitions <= bound M
- Enumerate potential partitions
  - Each table i is assigned 1 to M
- A total of M^N possibilities

An Efficient Enumeration Alg.

- Insights:
  - an AP uses a small number of pkt attributes (inputs)
  - for a given set of pkt attributes, all instructions determined by the set can merge into the same table

I2 can merge w/ I1.

Consider I2, which follows I1. If input(I2)=input(I1,I2), then merge them will not increase table size.

```
Map<MAC, int>    table;
   Route onPacketIn(Packet p) {
I1.  int val1 = table.get( p.ethSrc() );
I2.  int val2 = val1 * val1;
I3.  branch [val2 > 10] I4 I5
I4.  return pass
I5.  return drop
```

Summary

- Algorithmic policy: programmers focus on network control expressed in general purpose language, and no need to configure datapath
- Key challenge: how to get data-path (flow tables) from data-path oblivious algorithmic policies
- Maple:
  - Reactive (blackbox) approach
  - Dynamic tracing tree -> single table
- Magellan
  - A proactive (whitebox) approach
  - Automatic derivation and population of multi-table pipelines
  - Substantial performance improvement: 46-68x fewer rules

Thank You