Compiling Network Programs

Nate Foster
Cornell University

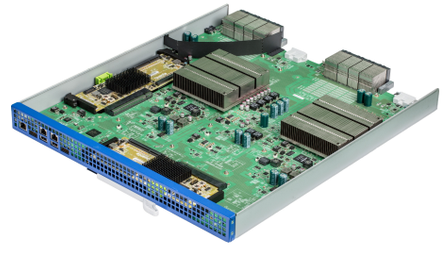
frenetic

Computing

Networking

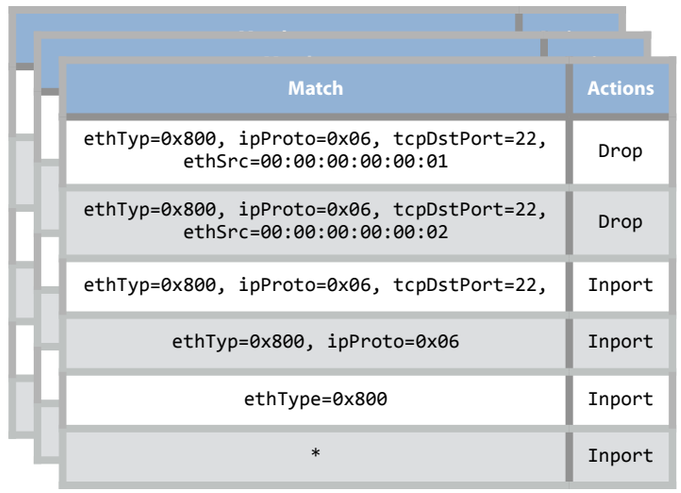
Computing

Networking



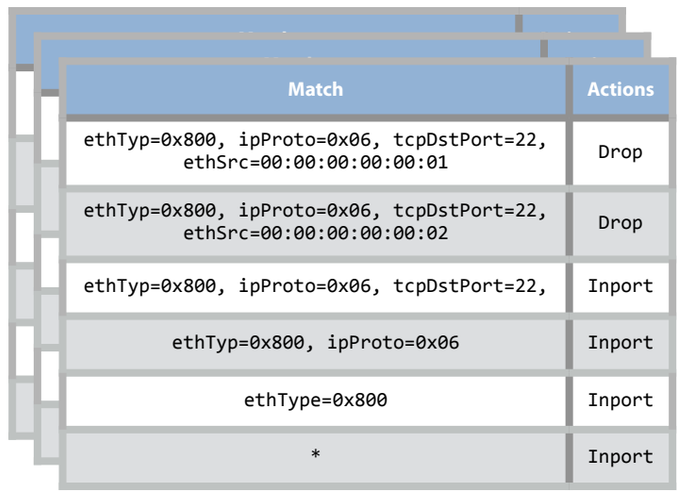
P4

A *machine model* based on flexible parsers and match-action pipelines



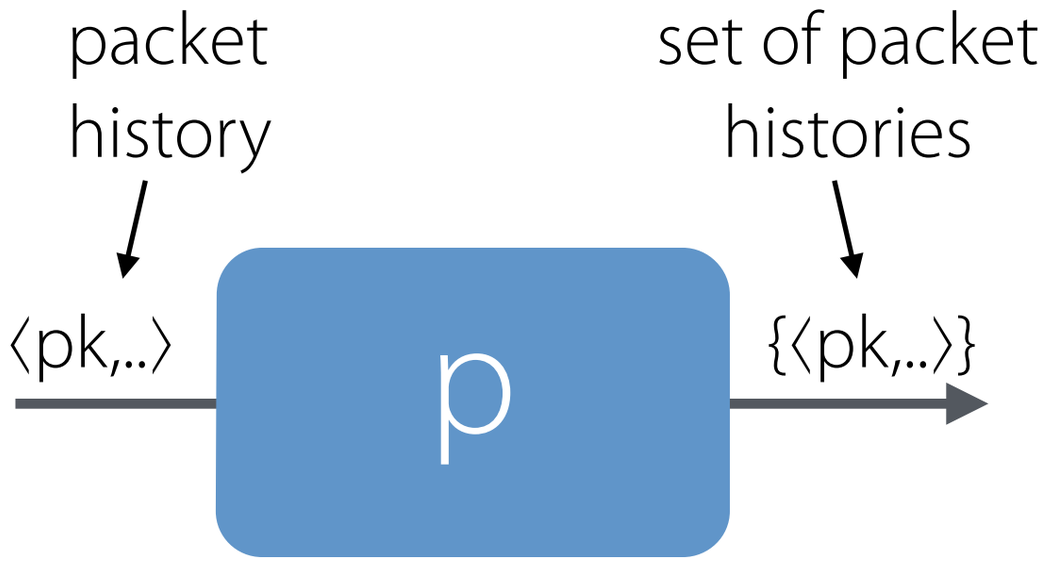
P4

A *machine model* based on flexible parsers and match-action pipelines



NetKAT

A *programming model* based on mathematical functions on packets

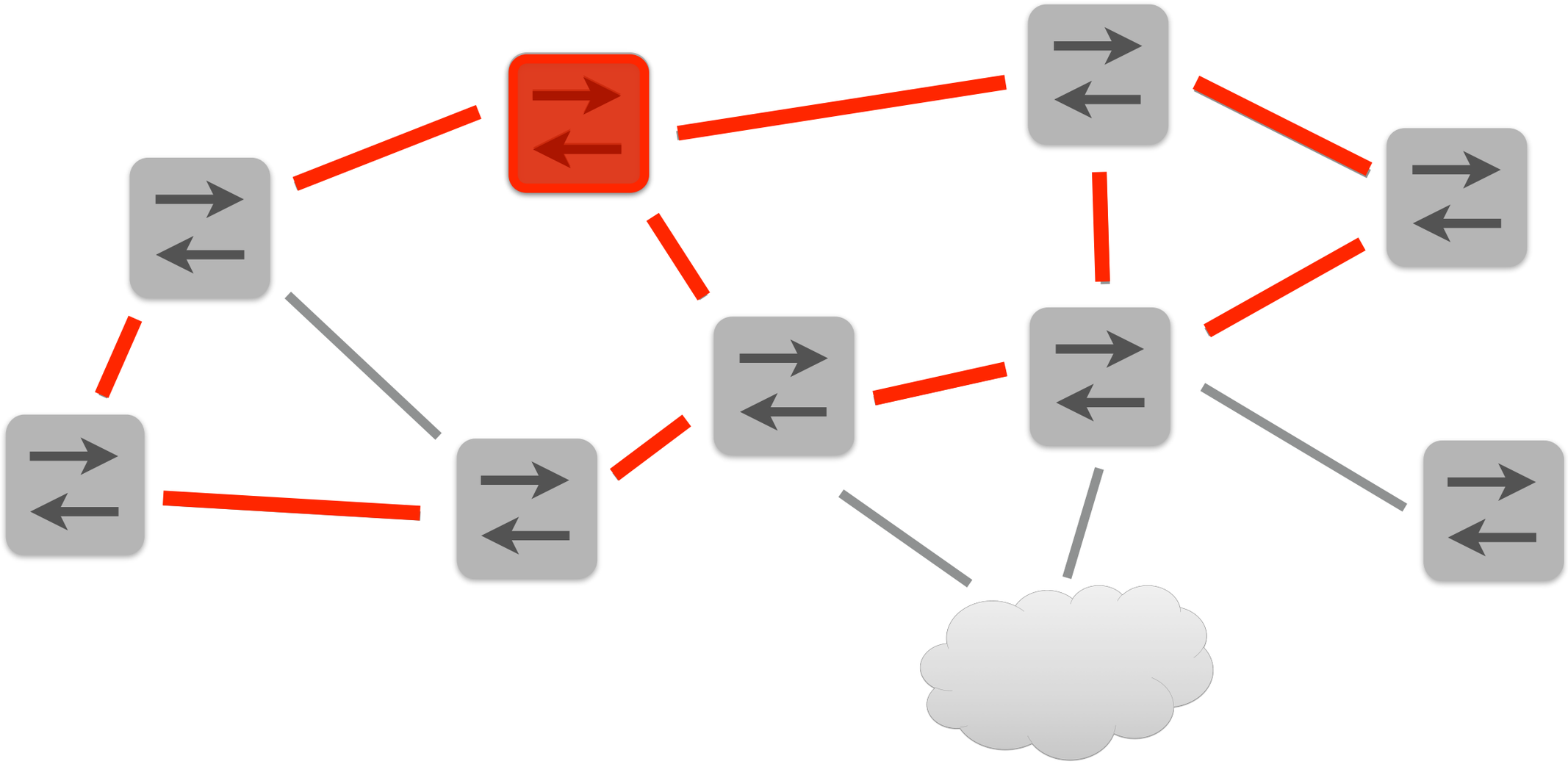


This Talk

Design and implementation of a high-level network programming language called NetKAT

Outline

- Language Design
- Compilation
- Experience



Two essential features:

- Packet classification
- Network-wide forwarding

NetKAT Syntax

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| !pol
| pol*
| field **:=** val
| S**⇒**T

NetKAT Syntax

```
pol ::=  false
       | true
       | field = val
       | pol1 + pol2
       | pol1 ; pol2
       | !pol
       | pol*
       | field := val
       | S⇒T
```

Boolean Algebra

NetKAT Syntax

```
pol ::= false
      | true
      | field = val
      | pol1 + pol2
      | pol1 ; pol2
      | !pol
      | pol*
      | field := val
      | S⇒T
```

Boolean Algebra

+

Kleene Algebra

NetKAT Syntax

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| !pol
| pol*
| field **:=** val
| S**⇒**T

Boolean Algebra

+

Kleene Algebra

+

Packet Primitives

NetKAT Syntax

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| !pol
| pol*
| field **:=** val
| S**⇒**T

Boolean Algebra + Kleene Algebra + Packet Primitives

} KAT

[Kozen '96]

NetKAT Syntax

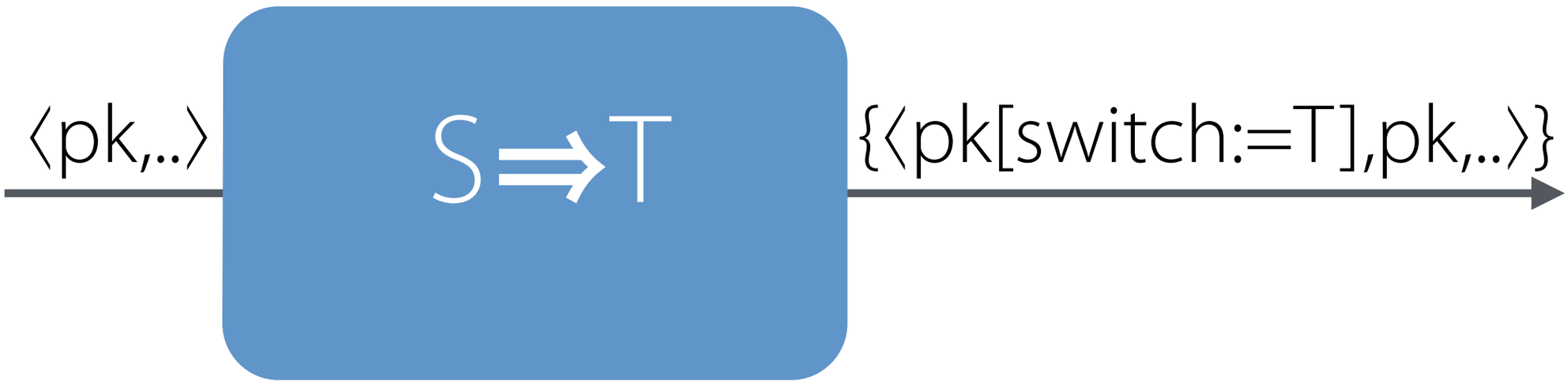
pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| !pol
| pol*
| field **:=** val
| S⇒T

Boolean Algebra + Kleene Algebra + Packet Primitives

KAT
[Kozen '96]

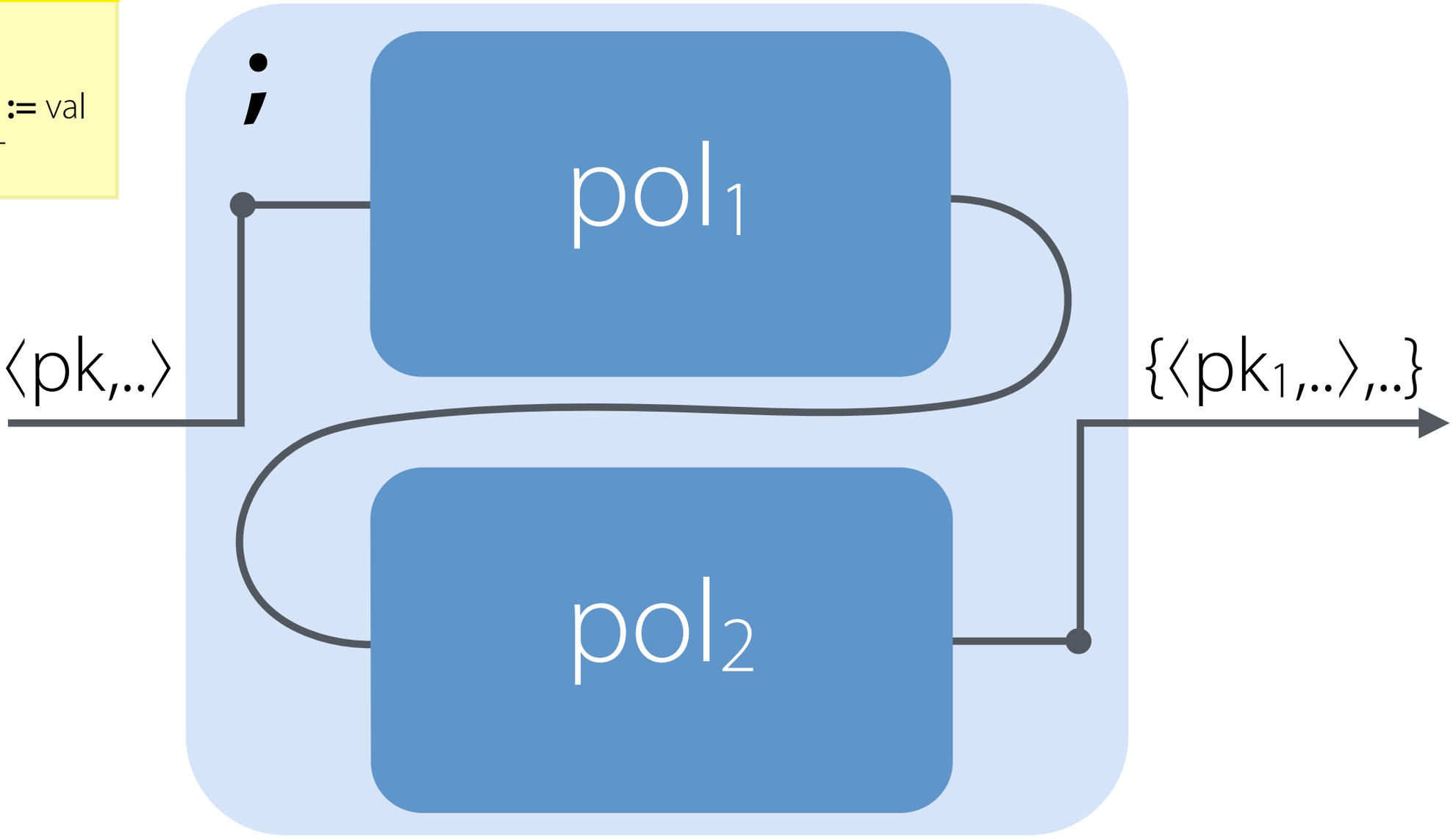
NetKAT
[Anderson et al. '14]

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇛T



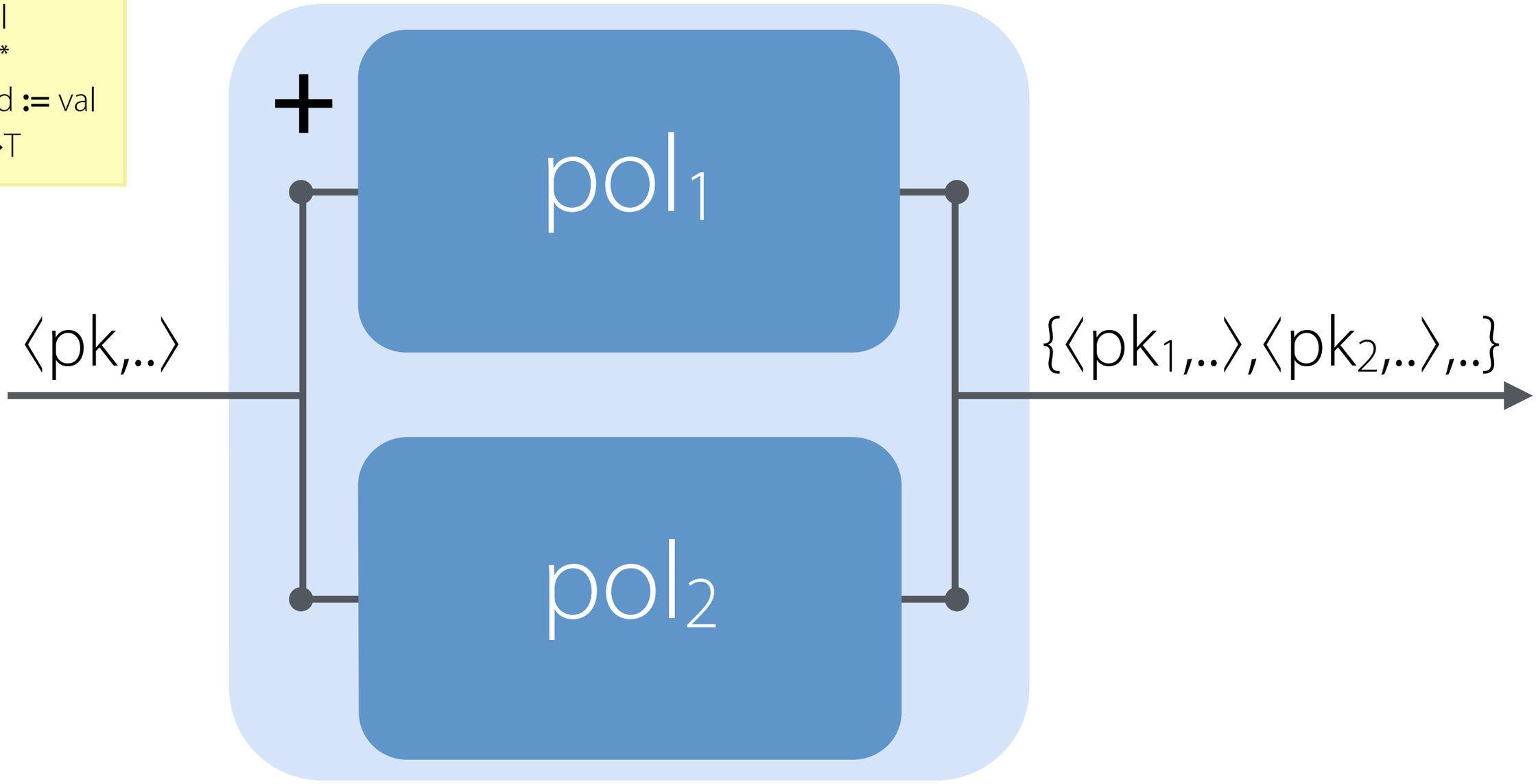
S⇛T forwards packets across the link between S and T

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S**⇒**T



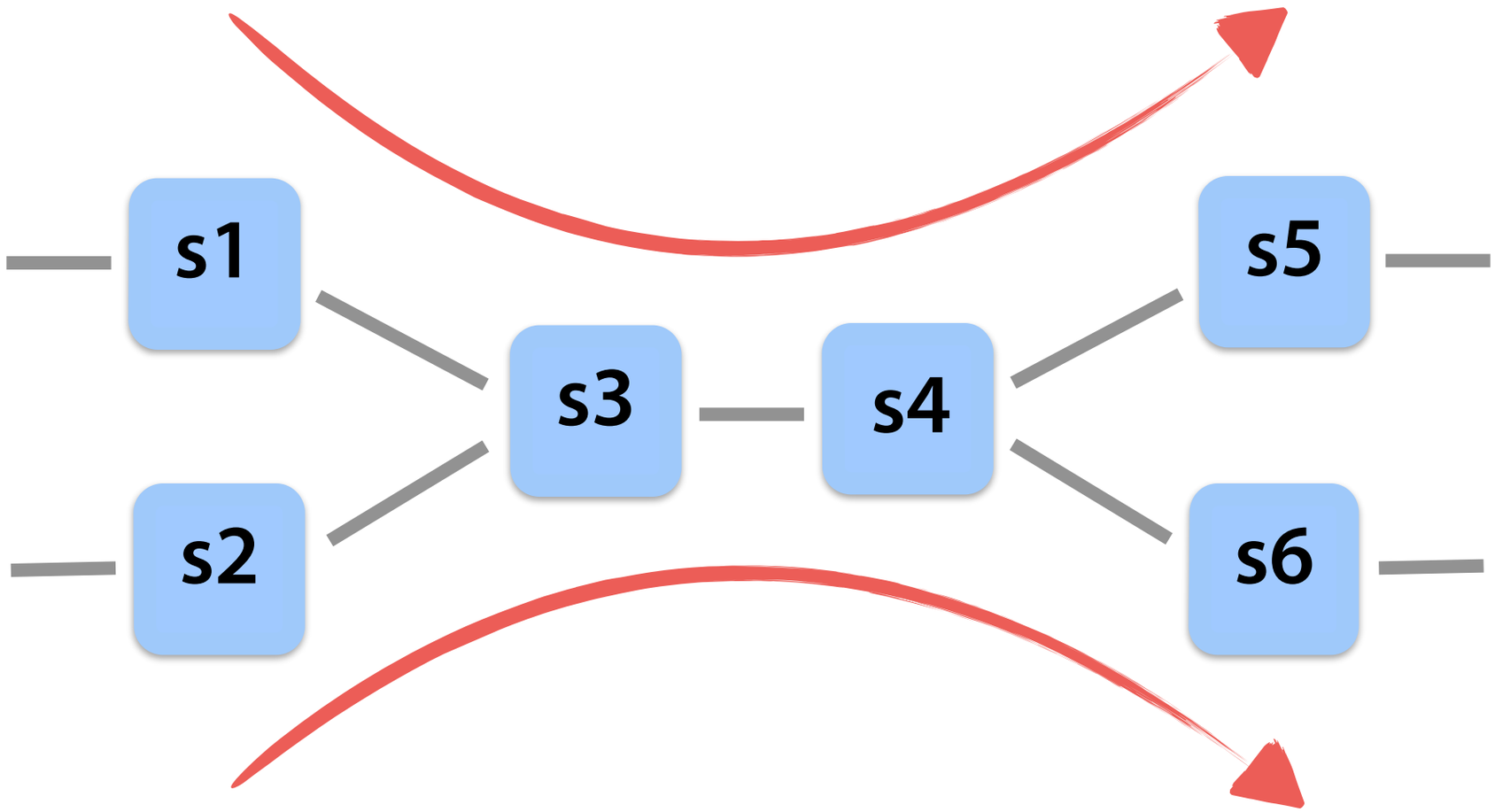
pol_1 **;** pol_2 runs the input through pol_1 and then runs every output produced by pol_1 through pol_2

pol ::= **false**
| **true**
| field **=** val
| pol_1 **+** pol_2
| pol_1 **;** pol_2
| **!**pol
| pol*
| field **:=** val
| S⇒T



pol_1 **+** pol_2 duplicates the input, sends one copy to each sub-policy, and takes the union of their outputs

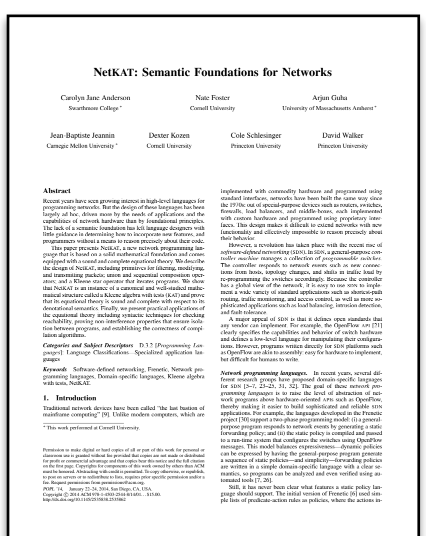
Example



(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)

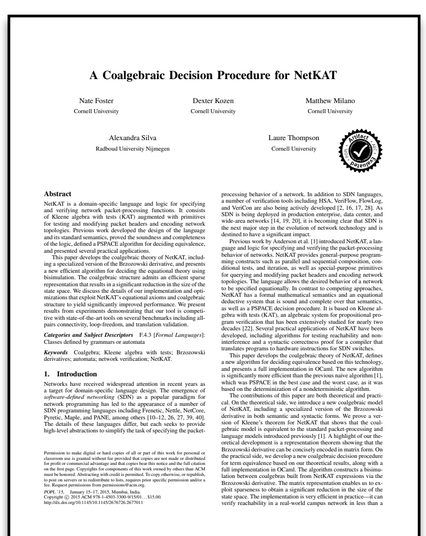
Other Applications

- Automatic Verification
- Network Virtualization
- Traffic Engineering
- Fault Tolerance
- Application Intent



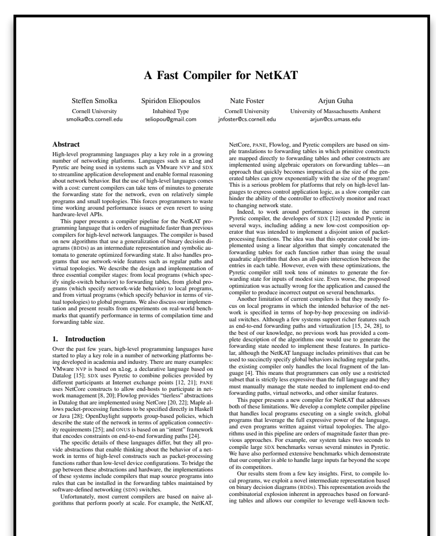
NetKAT: Semantic Foundations for Networks

[POPL '14]



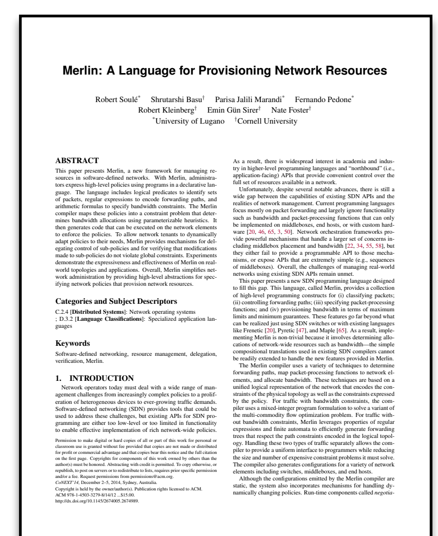
A Coalgebraic Decision Procedure for NetKAT

[POPL '15]



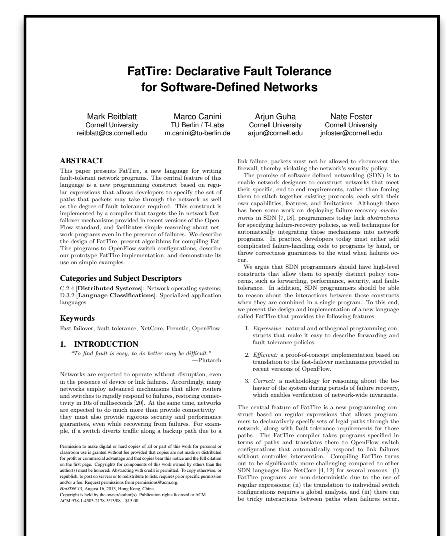
A Fast Compiler for NetKAT

[ICFP '15]



Merlin: A Language for Provisioning Network Resources

[CoNext '14]

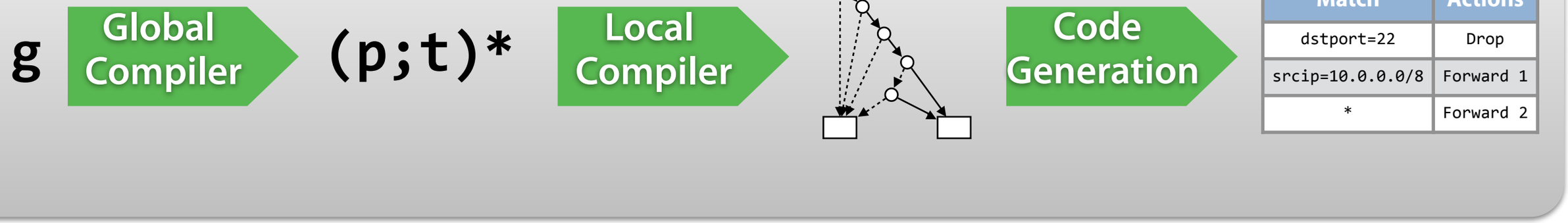


FatTire: Declarative Fault Tolerance for Software-Defined Networks

[HotSDN '13]

NetKAT Compilation

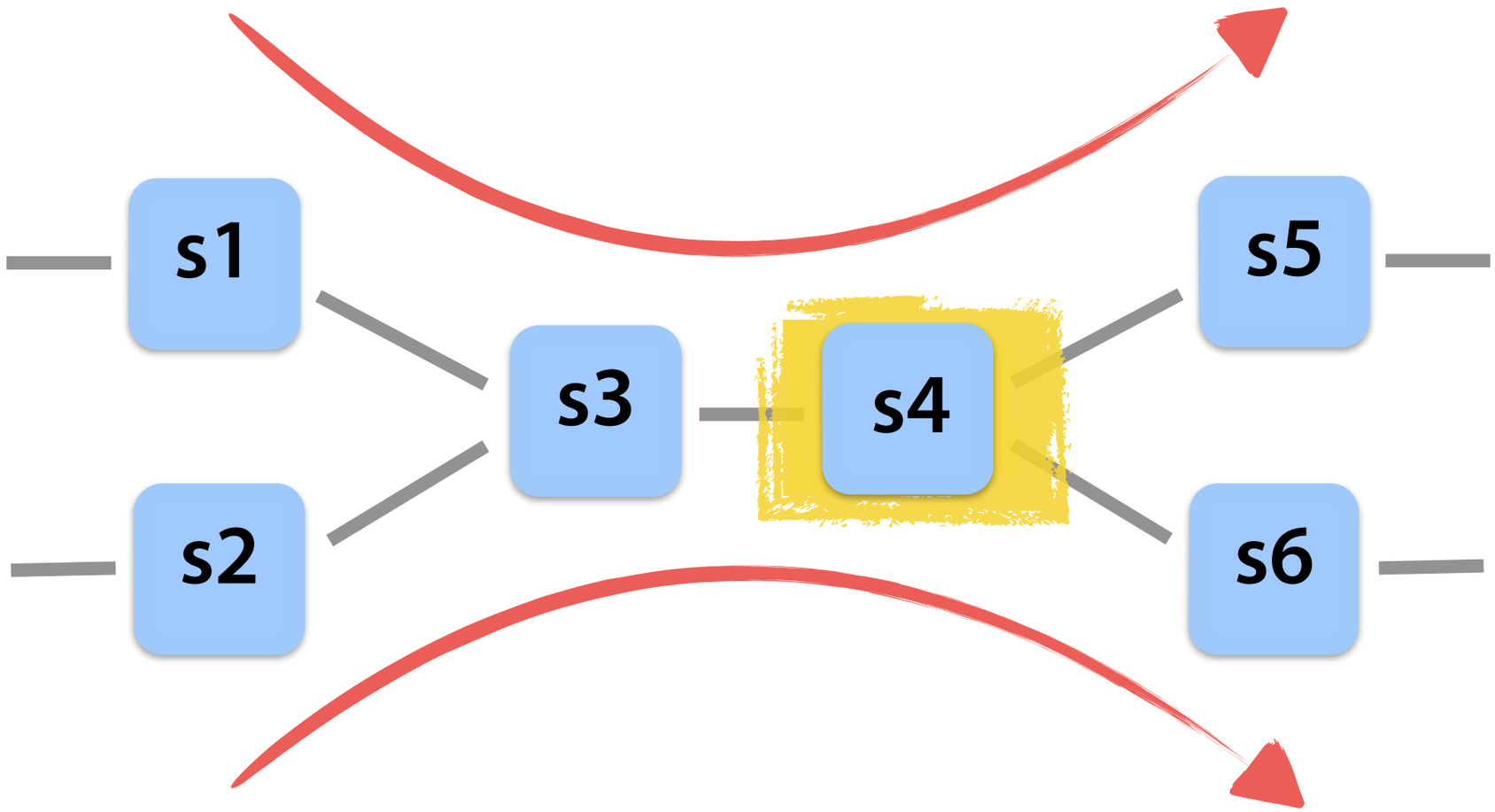
Compiler Pipeline



Key Challenges

- Implementing network-wide functions using local packet-processing elements
- Developing efficient data structures and algorithms to allow compilation to scale

Example



(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)

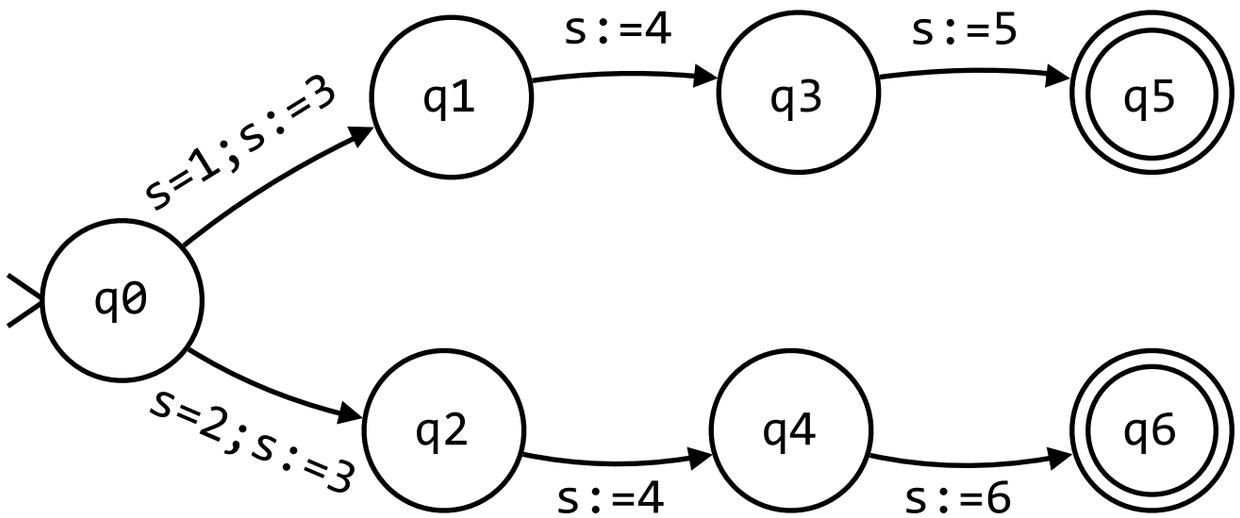
Global program

```
(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)
```

Global program

```
(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)
```

Automaton

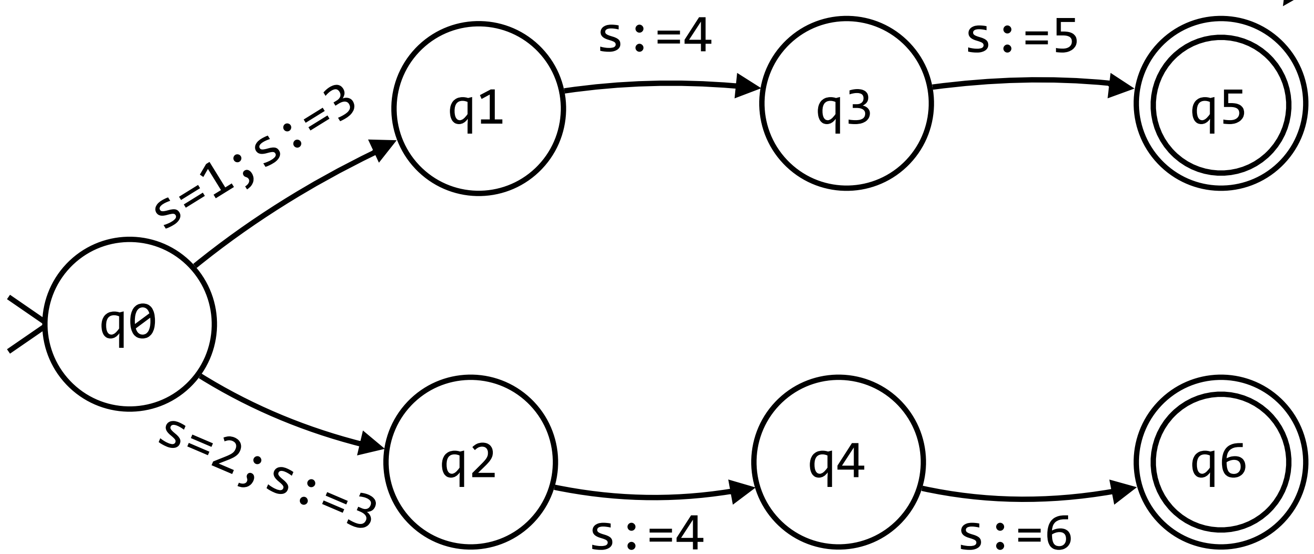


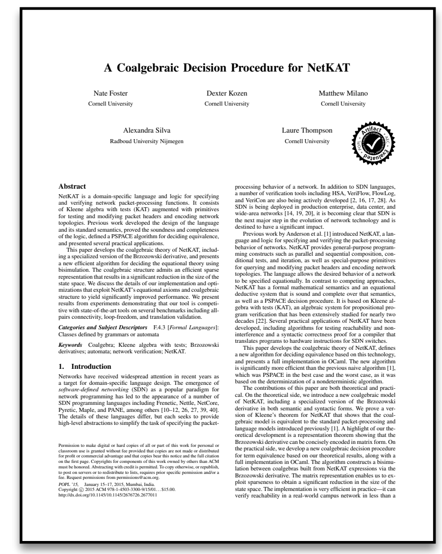
Global program

```
(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)
```

calculated using regular expression derivatives

Automaton




A Coalgebraic Decision Procedure for NetKAT

[POPL '15]

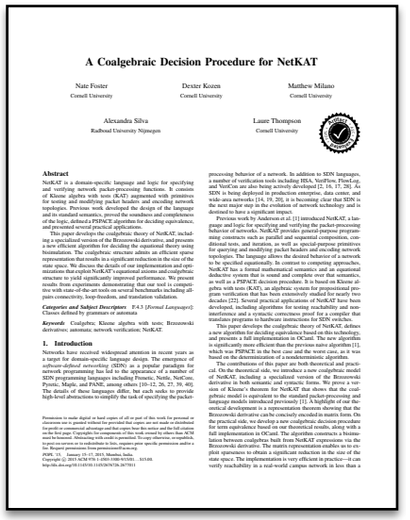

A Fast Compiler for NetKAT

[ICFP '15]

Global program

```
(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)
```

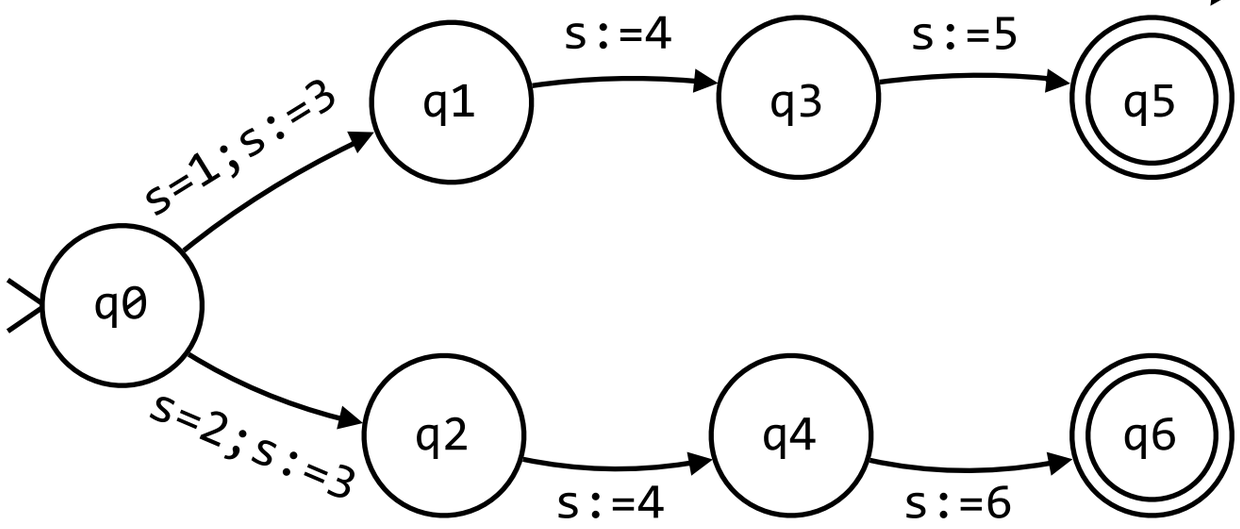
calculated using regular expression derivatives



[POPL '15]

[ICFP '15]

Automaton



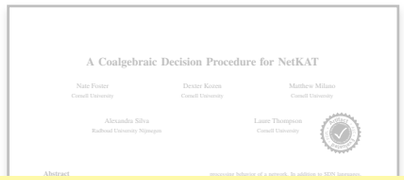
Local program

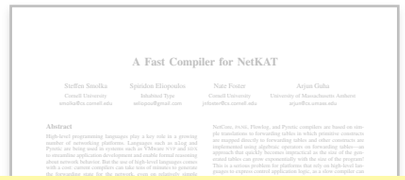
```
s=1;pc=q0;pc:=q1 +
s=2;pc=q0;pc:=q2 +
s=3;(pc=q1;pc:=q3 + pc=q2;pc:=q4) +
s=4;(pc=q3;pc:=q5 + pc=q4;pc:=q6)
```

Global program

```
(s1⇒s3⇒s4⇒s5) + (s2⇒s3⇒s4⇒s6)
```

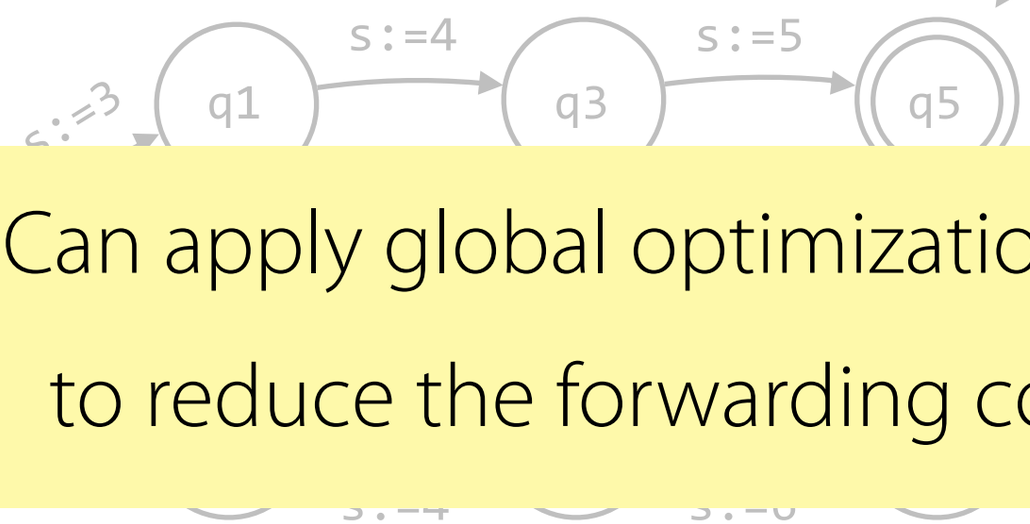
calculated using regular expression derivatives





[POPL '15] [ICFP '15]

Automaton



Can apply global optimizations to NetKAT automata

to reduce the forwarding complexity at each hop

Local program

```
s=1;pc=q0;pc:=q1 +
s=2;pc=q0;pc:=q2 +
s=3;(pc=q1;pc:=q3 + pc=q2;pc:=q4) +
s=4;(pc=q3;pc:=q5 + pc=q4;pc:=q6)
```

Table-Based Compilation

```
let route =
  if ethDst = 00:00:00:00:00:01 then
    port := 1
  else if ethDst = 00:00:00:00:00:02 then
    port := 2
  else if ethDst = 00:00:00:00:00:03 then
    port := 3
  else if ethDst = 00:00:00:00:00:04 then
    port := 4
  else if ethDst = ff:ff:ff:ff:ff:ff then
    flood
  else
    port := learn
```

Match	Actions
ethDst=00:00:00:00:00:01	Forward 1
ethDst=00:00:00:00:00:02	Forward 2
ethDst=00:00:00:00:00:03	Forward 3
ethDst=00:00:00:00:00:04	Forward 4
ethDst=ff:ff:ff:ff:ff:ff	Flood
*	Controller

;

```
let firewall =
  if (ethSrc = 00:00:00:00:00:01 +
      ethSrc = 00:00:00:00:00:02);
     ethTyp = 0x800 ;
     ipProto = 0x06 ;
     tcpDstPort = 22
   then
     false
   else
     true
```

Match	Actions
ethSrc=00:00:00:00:00:01 ethTyp=0x800, ipProto=0x06, tcpDstPort=22	Drop
ethSrc=00:00:00:00:00:01 ethTyp=0x800, ipProto=0x06, tcpDstPort=22	Drop
*	Inport

Table-Based Compilation

Pattern	Actions
ethSrc=00:00:00:00:00:01,ethDst=00:00:00:00:00:01,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:01,ethDst=00:00:00:00:00:02,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:01,ethDst=00:00:00:00:00:03,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:01,ethDst=00:00:00:00:00:04,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:01,ethDst=ff:ff:ff:ff:ff:ff,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:01,ethTyp=0x800,ipProto=0x06,tcpPort=22	Controller
ethSrc=00:00:00:00:00:02,ethDst=00:00:00:00:00:01,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:02,ethDst=00:00:00:00:00:02,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:02,ethDst=00:00:00:00:00:03,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:02,ethDst=00:00:00:00:00:04,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:02,ethDst=ff:ff:ff:ff:ff:ff,ethTyp=0x800,ipProto=0x06,tcpPort=22	Drop
ethSrc=00:00:00:00:00:02,ethTyp=0x800,ipProto=0x06,tcpPort=22	Controller
ethDst=00:00:00:00:00:01	Forward 1
ethDst=00:00:00:00:00:02	Forward 2
ethDst=00:00:00:00:00:03	Forward 3
ethDst=00:00:00:00:00:04	Forward 4
ethDst=ff:ff:ff:ff:ff:ff	Flood
*	Controller

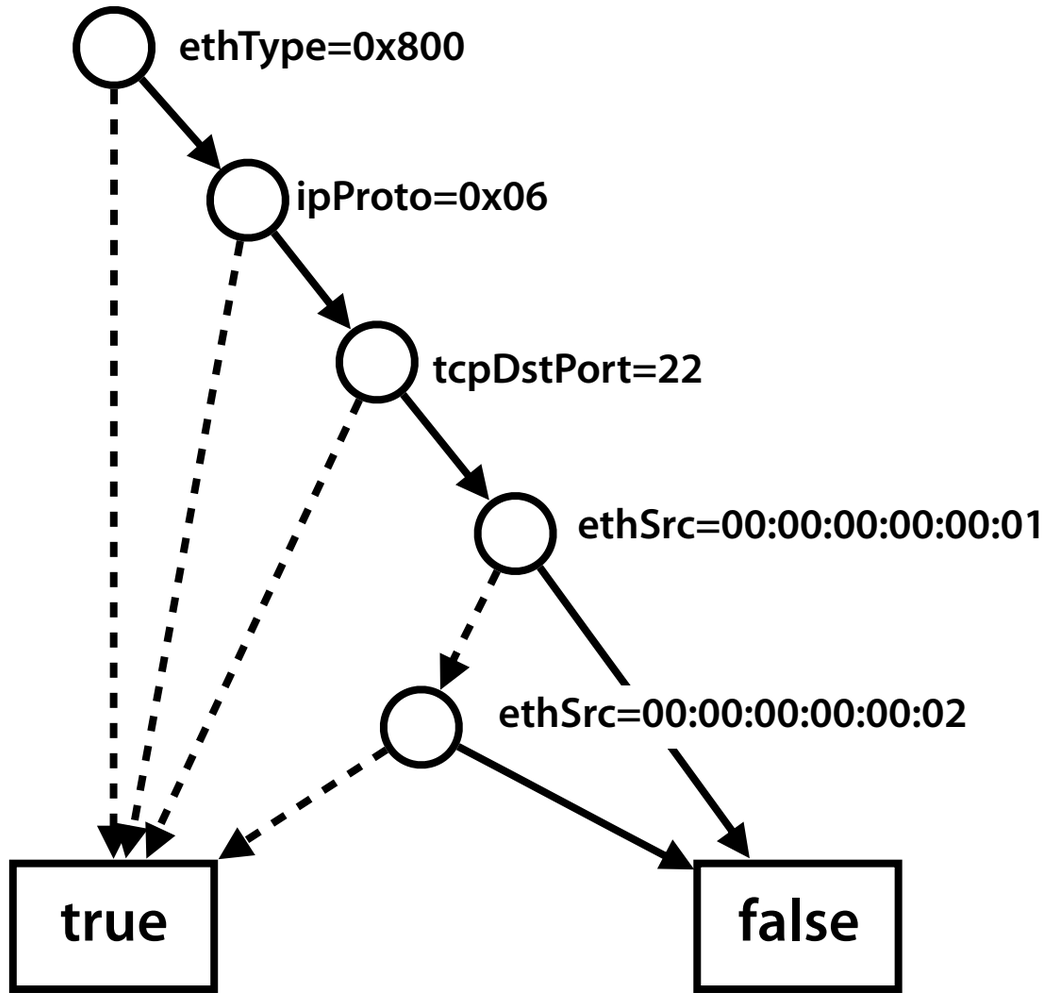
Forwarding Decision Diagrams

Observation: fundamentally, NetKAT compilation involves constructing a boolean classifier on packets

Idea: use representation based on decision diagrams

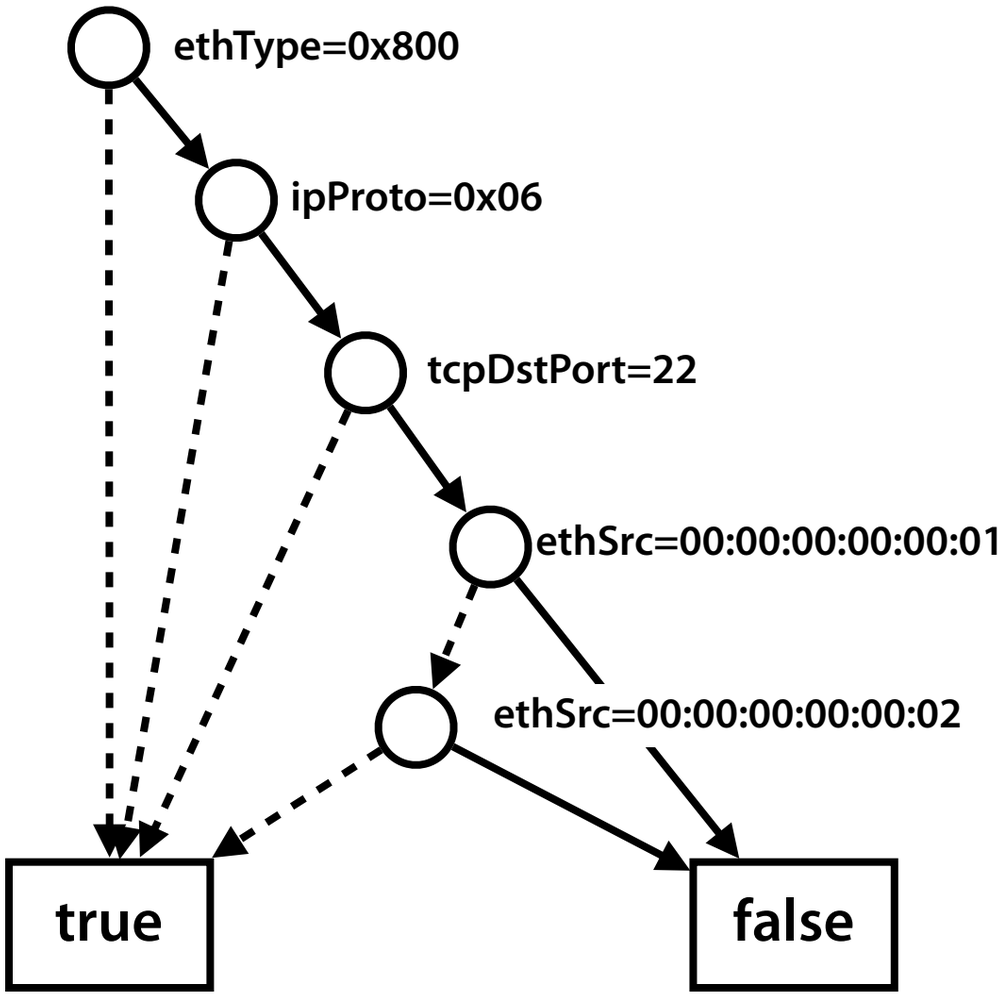
Example:

```
let firewall =
  if (ethSrc = 00:00:00:00:00:01 +
      ethSrc = 00:00:00:00:00:02);
     ethTyp = 0x800 ;
     ipProto = 0x06 ;
     tcpDstPort = 22
  then
    false
  else
    true
```



OpenFlow Table Generation

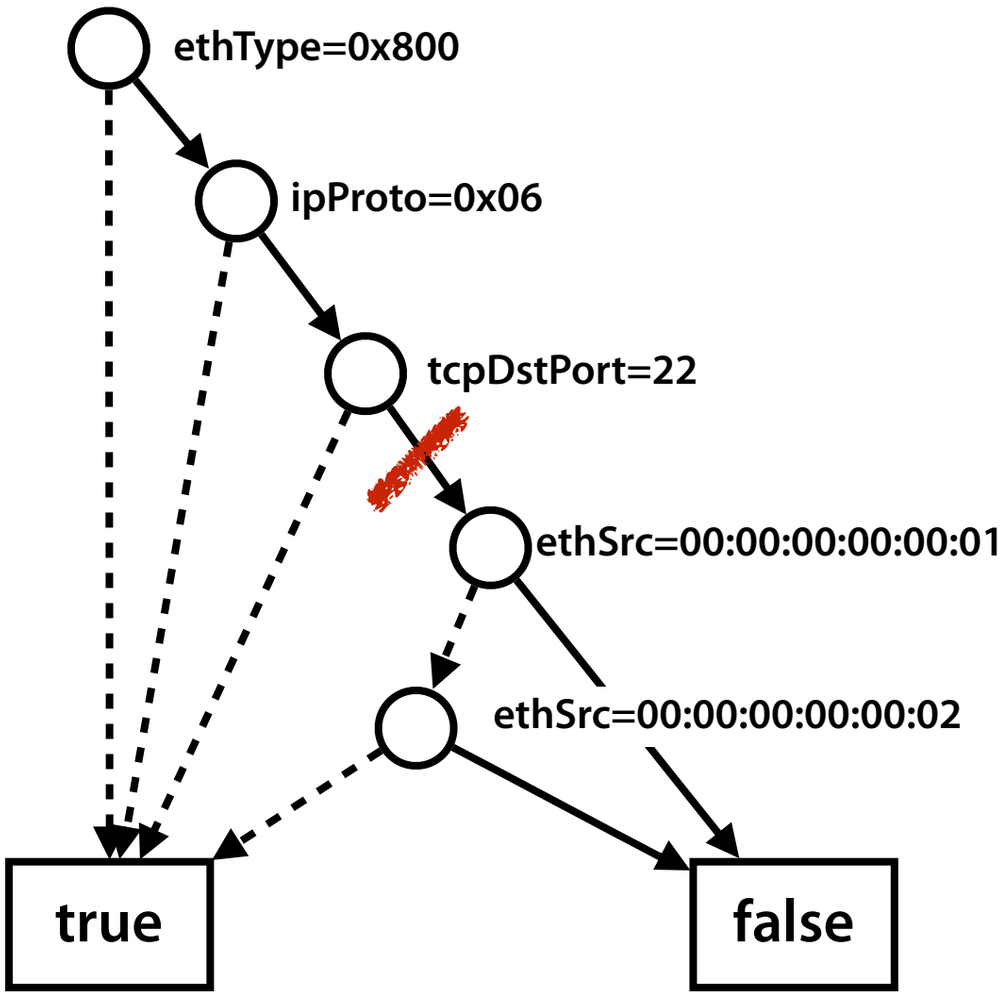
Can "read off" OpenFlow tables by enumerating paths



Match	Actions
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:01	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22, ethSrc=00:00:00:00:00:02	Drop
ethTyp=0x800, ipProto=0x06, tcpDstPort=22,	Inport
ethTyp=0x800, ipProto=0x06	Inport
ethType=0x800	Inport
*	Inport

P4 Table Generation

Can generate more compact tables by splitting FDDs on fields and using metadata to track control flow



SSH

Match	Actions
ethTyp=0x800, ipProto=0x06, tcpDstPort=22	ssh=1
*	ssh=0

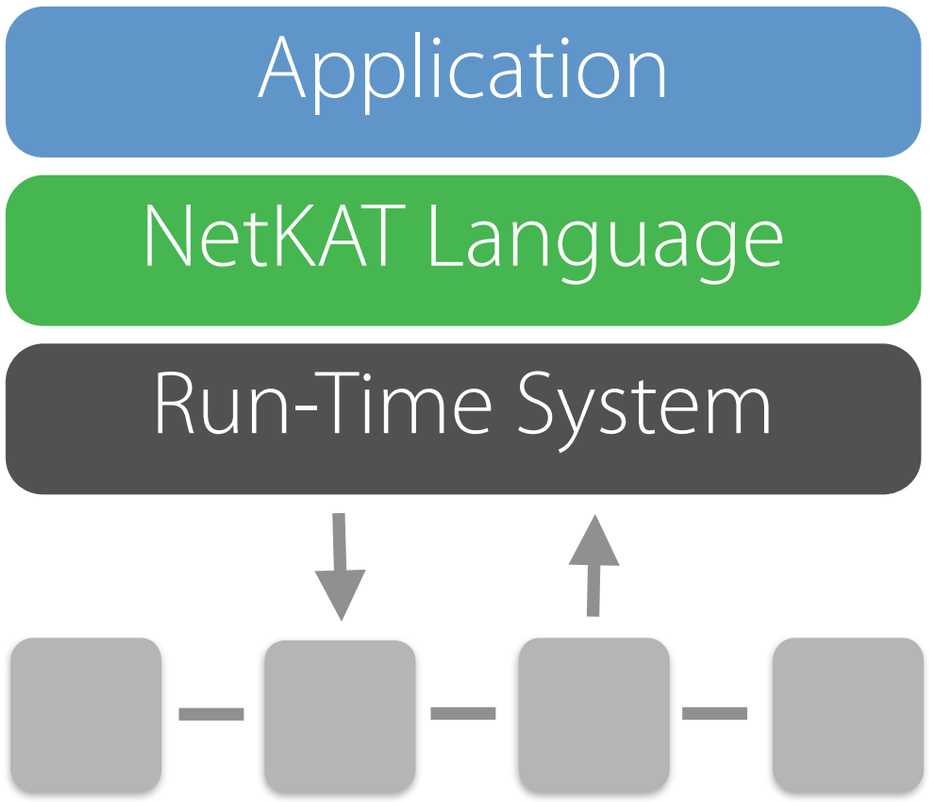
ACL

Match	Actions
ssh=1, ethSrc=00:00:00:00:00:01	Drop
ssh=1, ethSrc=00:00:00:00:00:02	Drop
*	Inport

Experience

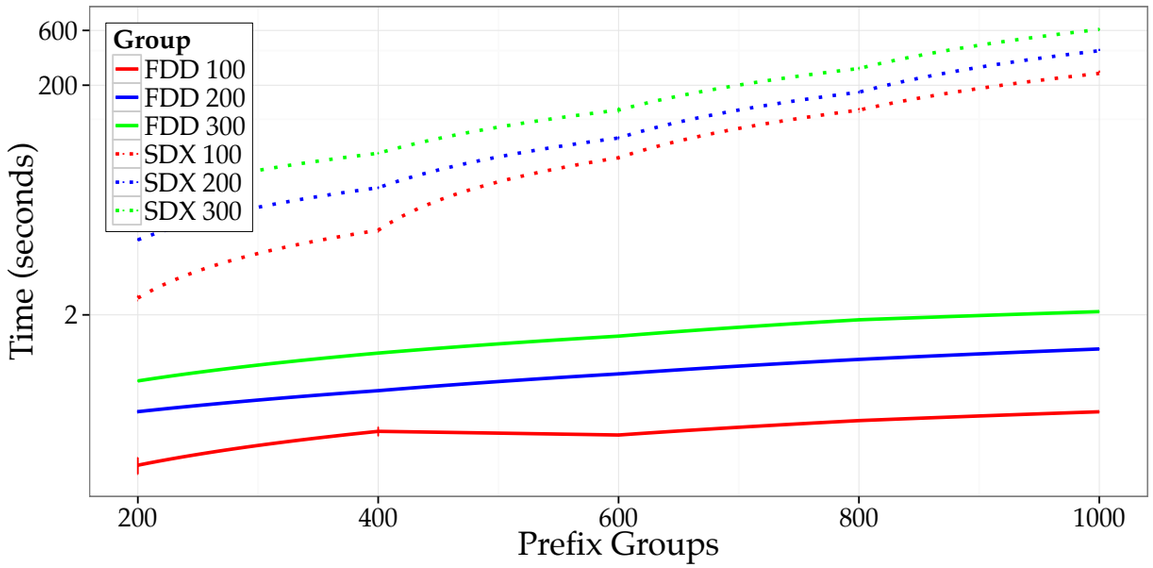
Implementation

Frenetic controller now based on NetKAT



Performance

Compiler is ~100X faster than its competitors



SDX [SIGCOMM '14]

Conclusion

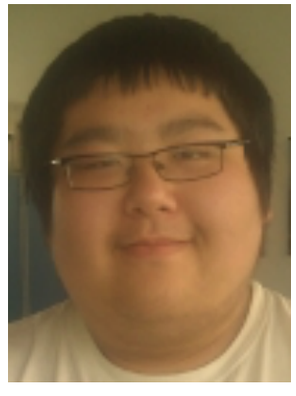
- High-level languages have an essential role to play in modern networking platforms
- NetKAT combines natural constructs for building packet classifiers and forwarding paths
- Can compile NetKAT efficiently using BDDs, symbolic automata, and (soon) P4 :-)

Thank you!

Steffen Smolka
Cornell PhD

Arjun Guha
UMass Faculty

Spiros Eliopoulos
Inhabited Type



Xiang Long
Cornell PhD

Other collaborators

- Carolyn Anderson (McGill)
- Jean-Baptiste Jeannin (CMU)
- Dexter Kozen (Cornell)
- Matthew Milano (Cornell)
- Cole Schlesinger (Princeton)
- Alexandra Silva (Nijmegen)
- Laure Thompson (Cornell)
- Dave Walker (Princeton)

`http://frenetic-lang.org/`