A Fast Compiler for NetKAT

Steffen Smolka
Nate Foster
Arjun Guha
Spiridon Eliopoulos
Networks have become programmable (just now!)...
Networks have become programmable (just now!)…

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstport=22</td>
<td>Drop</td>
</tr>
<tr>
<td>src=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>*</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>
Networks have become programmable (just now!)…
Networks have become programmable (just now!)…

**SDX: A Software Defined Internet Exchange**

Arpit Gupta¹, Laurent Vanbever, Muhammad Shahbaz², Sean P. Donovan³, Brandon Schlinker³, Nick Feamster, Jennifer Rexford, Scott Shenker, Russ Clark, Ethan Katz-Bassett¹

¹Georgia Tech ²Princeton University ³UC Berkeley ⁴Univ. of Southern California

Abstract

BGP severely constrains how networks route traffic over the Internet. Today's networks can only route traffic based on the destination prefix. Networks cannot make more fine-grained routing decisions based on the type of application or the sender. Instead, they rely on default routes to route traffic to distant neighbors. A network also automatically routes traffic to immediate neighbors. We believe Software Defined Networks (SDN) could revolutionize how networks route traffic to neighbors closer in end-cast. We believe SDN could revolutionize how networks route traffic to neighbors closer in end-cast.

These problems motivate two complementary systems that tackle these challenges. First, we propose an SDN switch implementation that can represent policies for hundreds of participants who advertise full routing tables, achieving sub-second response to configuration changes and routing updates. SDN switches can be used in multiple network operators who advertise full routing tables, achieving sub-second response to configuration changes and routing updates.

In the second system, we propose an SDX switch that can route traffic over direct neighbors. A network can automatically route traffic to direct neighbors. We believe SDN could revolutionize how networks route traffic to neighbors closer in end-cast.

Categories and Subject Descriptors: C.2.1 [Computer Communication Networks]: Network Architecture and Design; C.2.2 [Computer Communication Networks]: Network Protocols.

1 Introduction

Internet routing is complex, intricate, and difficult to manage. Network operators must rely on arcane mechanisms to perform routing engineering, prevent attacks, and realize peering agreements. Internet routing's problems result from the nature of the Border Gateway Protocol (BGP), the Internet's internetworking protocol.
Networks have become programmable (just now!)…

**System**

- **Compiler**
  - **Firewall**
  - **Route**

**Pattern** → **Actions**

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstport=22</td>
<td>Drop</td>
</tr>
<tr>
<td>src=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>*</td>
<td>Fwd 2</td>
</tr>
</tbody>
</table>

**SIGCOMM 2014**

6.3 Compilation Time

We measure the compilation time for two scenarios: (1) initial compilation time, which measures the time to compile the initial set of policies to the resulting forwarding rules; and (2) incremental compilation time, which measures how long it takes to recompute when changes occur.

Initial compilation time. Figure 8 shows how the time to compile low-level forwarding rules from higher-level policies varies as we increase both the number of prefix groups and IXP participants. The time to compute the forwarding rules is in the order of several minutes for typical numbers of prefix groups and participants. The results also show that compilation time increases roughly quadratically with the number of prefix groups. The compilation time increases more quickly than linearly because, as the number of prefix groups increases, the interactions between policies of pairs of participants at the SDX also increases. The time for the SDX to compute VNFs increases non-linearly as the number of participants and prefix groups increases. We observed that for 1,000 prefix groups and 100 participants, VNF compilation took about five minutes.

As discussed in Section 4.3, the SDX controller achieves faster compilation by memorizing the results of partial policy compilations. Supporting caching for 300 participants at the SDX and 1,000 prefix groups could require a cache of about 4.5 GB. Although this requirement seems large, it is on the order of the amount of memory required for a route server in a large operational IXP today.

Incremental compilation time. Recall that in addition to compiling an initial set of forwarding table rules, the SDX controller must recompile them whenever the best BGP route changes or when any participant updates its policy. We now evaluate the benefits of the optimizations that we discussed in Section 4.3 in terms of the savings in computation time. When new BGP updates arrive at the controller, the controller must recompute VNF IP addresses and

7 Related Work

We briefly describe related work in SDN exchange points, interdomain route control, and policy languages for SDNs.

SDN-based exchange points. The most closely related work is Google’s Cardigan project [22], which shares our broad goal of using SDN to enable innovation at IXPs. Cardigan runs a route server based on ReciFlow [17] and uses an OpenFlow switch to enforce security and routing policies. The Cardigan project is developing a logical SDN-based exchange point that is physically distributed across multiple locations. Unlike the SDX in this paper, Cardigan does not provide a general controller for composing participant policies, offer a framework that allows IXP participants to write policies in a high-level language, or implement technologies that can be integrated into existing IXP operational environments.
Networks have become programmable (just now!)…

Compilation Time
Networks have become programmable (just now!)…

Compilation Time

…with ad hoc performance hacks
(some of which turned out to be unsound)
Networks have become programmable (just now!)…

Compilation Time

…with ad hoc performance hacks (some of which turned out to be unsound)

Top 5 IXPs

<table>
<thead>
<tr>
<th>Name</th>
<th>Participants</th>
</tr>
</thead>
<tbody>
<tr>
<td>IX.br</td>
<td>861</td>
</tr>
<tr>
<td>Equinix</td>
<td>768</td>
</tr>
<tr>
<td>AMS-IX</td>
<td>710</td>
</tr>
<tr>
<td>LINX</td>
<td>652</td>
</tr>
<tr>
<td>DE-CIX</td>
<td>610</td>
</tr>
</tbody>
</table>

Networks have become programmable (just now!)…

…but current compilers are too slow
Networks have become programmable (just now!)…

…but current compilers are too slow
Networks have become programmable (just now!)…

…but current compilers are:

→ too slow
→ limited to "local" languages
Our Contribution

First *complete* compiler pipeline for NetKAT
Our Contribution

First *complete* compiler pipeline for NetKAT

- **3**
- **2** local policy
- **Local Compiler**

Drop-in replacement
~ 100x speedup
Our Contribution

First *complete* compiler pipeline for NetKAT

3 global policy

**Global Compiler**

local policy

**Local Compiler**

- network-wide behavior
- drop-in replacement
- ~ 100x speedup

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstpt=2</td>
<td>drop</td>
</tr>
<tr>
<td>srcpt=7</td>
<td>fwd 1</td>
</tr>
<tr>
<td>*</td>
<td>fwd 2</td>
</tr>
</tbody>
</table>
Our Contribution

First *complete* compiler pipeline for NetKAT

- **Virtual Compiler**: abstract topologies
- **Global Compiler**: network-wide behavior
- **Local Compiler**: drop-in replacement ~ 100x speedup
Our Contribution

First *complete* compiler pipeline for NetKAT

- **Virtual Compiler**
  - abstract topologies

- **Global Compiler**
  - network-wide behavior

- **Local Compiler**
  - drop-in replacement
    - ~ 100x speedup

**Virtual Policy**

**Global Policy**

**Local Policy**
Our Contribution

First *complete* compiler pipeline for NetKAT

- **Virtual Compiler**: abstract topologies
- **Global Compiler**: network-wide behavior
- **Local Compiler**: drop-in replacement ~ 100x speedup
Our Contribution

First *complete* compiler pipeline for NetKAT

- **Virtual Compiler**
  - abstract topologies

- **Global Compiler**
  - network-wide behavior

- **Local Compiler**
  - drop-in replacement ~ 100x speedup

Based on:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>dstpt=2</td>
<td>drop</td>
</tr>
<tr>
<td>srcpt=7</td>
<td>fwd 1</td>
</tr>
<tr>
<td>*</td>
<td>fwd 2</td>
</tr>
</tbody>
</table>
**System**

### Firewall
- **Pattern**: $\text{dstport}=22$
- **Actions**: Drop

### Route
- **Pattern**: $\text{src}=10.0.0.1$
- **Actions**: Fwd 01

---

**Compiler**

### Source Language?

### Target Language?
The Target: **Match+Action Tables**

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>port=2, srcIP=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>port=2</td>
<td>Drop</td>
</tr>
<tr>
<td>port=1</td>
<td>dstIP=10.0.0.2, Fwd 2</td>
</tr>
<tr>
<td>*</td>
<td>Fwd 1, Fwd 2</td>
</tr>
</tbody>
</table>

= "Ordered Lookup Table"
The Target: **Match+Action Tables**

<table>
<thead>
<tr>
<th>Match</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>port=2, srcIP=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>port=2</td>
<td>Drop</td>
</tr>
<tr>
<td>port=1</td>
<td>dstIP=10.0.0.2, Fwd 2</td>
</tr>
<tr>
<td>*</td>
<td>Fwd 1, Fwd 2</td>
</tr>
</tbody>
</table>

= "Ordered Lookup Table"

→ designed for efficient execution in hardware
The Goal:

Create one table for each switch in the network…
Create one table for each switch in the network…
The Goal:

Create one table for each switch in the network...

...given a high-level program in the source language
The Source: NetKAT Language

pol ::= 
  | false 
  | true 
  | field = val 
  | field := val 
  | pol₁ + pol₂ 
  | pol₁ ; pol₂ 
  | !pol 
  | pol* 
  | S → S'
The Source: **NetKAT Language**

<table>
<thead>
<tr>
<th>pol ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>false</strong></td>
</tr>
<tr>
<td><strong>true</strong></td>
</tr>
<tr>
<td>field = val</td>
</tr>
<tr>
<td>field := val</td>
</tr>
<tr>
<td>pol₁ + pol₂</td>
</tr>
<tr>
<td>pol₁ ; pol₂</td>
</tr>
<tr>
<td>!pol</td>
</tr>
<tr>
<td>pol*</td>
</tr>
<tr>
<td>S → S'</td>
</tr>
</tbody>
</table>

**Boolean Algebra**
### NetKAT Language

<table>
<thead>
<tr>
<th>pol ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>field = val</td>
</tr>
<tr>
<td>field := val</td>
</tr>
<tr>
<td>pol₁ + pol₂</td>
</tr>
<tr>
<td>pol₁ ; pol₂</td>
</tr>
<tr>
<td>!pol</td>
</tr>
<tr>
<td>pol*</td>
</tr>
<tr>
<td>S→S'</td>
</tr>
</tbody>
</table>

Boolean Algebra +
Kleene Algebra
"Regular Expressions"
The Source: **NetKAT Language**

<table>
<thead>
<tr>
<th>pol ::=</th>
</tr>
</thead>
<tbody>
<tr>
<td>false</td>
</tr>
<tr>
<td>true</td>
</tr>
<tr>
<td>field = val</td>
</tr>
<tr>
<td>field := val</td>
</tr>
<tr>
<td>pol₁ + pol₂</td>
</tr>
<tr>
<td>pol₁; pol₂</td>
</tr>
<tr>
<td>!pol</td>
</tr>
<tr>
<td>pol*</td>
</tr>
<tr>
<td>S → S'</td>
</tr>
</tbody>
</table>

---

**Boolean Algebra**

+ **Kleene Algebra**

"**Regular Expressions**"

+ **Packet Primitives**
Semantics

\[ \text{pol ::= } \]
\[ \qquad | \text{false} \]
\[ \qquad | \text{true} \]
\[ \qquad | \text{field = val} \]
\[ \qquad | \text{field := val} \]
\[ \qquad | \text{pol}_1 + \text{pol}_2 \]
\[ \qquad | \text{pol}_1 ; \text{pol}_2 \]
\[ \qquad | \text{!pol} \]
\[ \qquad | \text{pol*} \]
\[ \qquad | S \rightarrow S' \]
**Semantics**

Local NetKAT: input-output behavior of switches

\[
\begin{align*}
pol & ::= \\
& \mid \text{false} \\
& \mid \text{true} \\
& \mid \text{field} = \text{val} \\
& \mid \text{field} ::= \text{val} \\
& \mid pol_1 + pol_2 \\
& \mid pol_1 ; pol_2 \\
& \mid !pol \\
& \mid pol^* \\
& \mid S \leadsto S' 
\end{align*}
\]
Semantics

\[ \text{pol} ::= \]
\[ \mid \text{false} \]
\[ \mid \text{true} \]
\[ \mid \text{field} = \text{val} \]
\[ \mid \text{field} ::= \text{val} \]
\[ \mid \text{pol}_1 + \text{pol}_2 \]
\[ \mid \text{pol}_1 ; \text{pol}_2 \]
\[ \mid !\text{pol} \]
\[ \mid \text{pol}^* \]
\[ \mid S \rightarrow S' \]

**Local NetKAT:** input-output behavior of switches

\[ \llbracket \text{pol} \rrbracket \in \text{Packet} \rightarrow \text{Packet Set} \]

**Global NetKAT:** network-wide behavior

\[ \llbracket \text{pol} \rrbracket \in \text{Trace} \rightarrow \text{Trace Set} \]
Local NetKAT Program

\begin{align*}
&\text{pol}_A \\
&\text{pol}_B
\end{align*}
Local NetKAT Program

port := 3
Local NetKAT Program

```
port = 1;
tag := 1;
port := 3
```

```
port = 2;
tag := 2;
port := 3
```

```
???
```
Local NetKAT Program

port = 1; tag := 1; port := 3
+ port = 2; tag := 2; port := 3

port = 5; tag := 1
+ tag = 2; port := 6
Local NetKAT Program

tedious for programmers… difficult to get right!

A

1

2

B

5

6

port=1; tag:=1; port:=3
+ port=2; tag:=2; port:=3

tag=1; port:=5
+ tag=2; port:=6
Global NetKAT Program
Global NetKAT Program

port = 1; A ⇸ B;
port := 5

port = 2; A ⇸ B; port := 6
Global NetKAT Program

simple and elegant!

port = 1; A \rightarrow B; port := 5
+ 
port = 2; A \rightarrow B; port := 6
Virtual NetKAT Program
Virtual NetKAT Program

virtual "big switch"
Virtual NetKAT Program

virtual "big switch"

```
port=1; port:=5
+  
port=2; port:=6
```

even simpler!
**Input:** local program

**Output:** collection of flow tables, one per switch

**Challenges:** efficiency and size of generated tables
let **route** =
  if ipDst = 10.0.0.1 then
    port := 1
  else if ipDst = 10.0.0.2 then
    port := 2
  else
    port := learn

let **monitor** =
  if (tcpSrc = 22 + tcpDst = 22) then
    port:=console
  else
    false
Traditional Approach

Let `route` =

\[
\begin{align*}
\text{if } \text{ipDst} = 10.0.0.1 & \text{ then} \\
\quad \text{port} & := 1 \\
\text{else if } \text{ipDst} = 10.0.0.2 & \text{ then} \\
\quad \text{port} & := 2 \\
\text{else} & \\
\quad \text{port} & := \text{learn}
\end{align*}
\]

Let `monitor` =

\[
\begin{align*}
\text{if } (\text{tcpSrc} = 22 + \text{tcpDst} = 22) & \text{ then} \\
\quad \text{port} & := \text{console} \\
\text{else} & \\
\quad \text{false}
\end{align*}
\]

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>src=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpSrc=22</td>
<td>Controller</td>
</tr>
<tr>
<td>tcpDst=22</td>
<td>Controller</td>
</tr>
<tr>
<td>*</td>
<td>Drop</td>
</tr>
</tbody>
</table>
let \textbf{route} = 
  \begin{align*}
  &\text{if } \text{ipDst} = 10.0.0.1 \text{ then} \\
  &\quad \text{port} := 1 \\
  &\text{else if } \text{ipDst} = 10.0.0.2 \text{ then} \\
  &\quad \text{port} := 2 \\
  &\text{else} \\
  &\quad \text{port} := \text{learn}
  \end{align*}

let \textbf{monitor} =
  \begin{align*}
  &\text{if } (\text{tcpSrc} = 22 + \text{tcpDst} = 22) \text{ then} \\
  &\quad \text{port} := \text{console} \\
  &\text{else} \\
  &\quad \text{false}
  \end{align*}

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>src=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpSrc=22</td>
<td>Controller</td>
</tr>
<tr>
<td>tcpDst=22</td>
<td>Controller</td>
</tr>
<tr>
<td>*</td>
<td>Drop</td>
</tr>
</tbody>
</table>
Traditional Approach

let route =  
    if ipDst = 10.0.0.1 then  
        port := 1  
    else if ipDst = 10.0.0.2 then  
        port := 2  
    else  
        port := learn

let monitor =  
    if (tcpSrc = 22 + tcpDst = 22) then  
        port := console  
    else  
        false

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>src=10.0.0.1</td>
<td>Fwd 1</td>
</tr>
<tr>
<td>src=10.0.0.2</td>
<td>Fwd 2</td>
</tr>
<tr>
<td>*</td>
<td>Controller</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>tcpSrc=22</td>
<td>Controller</td>
</tr>
<tr>
<td>tcpDst=22</td>
<td>Controller</td>
</tr>
<tr>
<td>*</td>
<td>Drop</td>
</tr>
</tbody>
</table>

Inefficient!

Tables are a hardware abstraction, not an efficient data structure!!
Our Approach

let route =
  if ipDst = 10.0.0.1 then
    port := 1
  else if ipDst = 10.0.0.2 then
    port := 2
  else
    port := learn

let monitor =
  if (tcpSrc = 22 + tcpDst = 22) then
    port := console
  else
    false
Our Approach

let route =
  if ipDst = 10.0.0.1 then
    port := 1
  else if ipDst = 10.0.0.2 then
    port := 2
  else
    port := learn

let monitor =
  if (tcpSrc = 22 + tcpDst = 22) then
    port := console
  else
    false
Our Approach

let route =
  if ipDst = 10.0.0.1 then
    port := 1
  else if ipDst = 10.0.0.2 then
    port := 2
  else
    port := learn

let monitor =
  if (tcpSrc = 22 + tcpDst = 22) then
    port := console
  else
    false

Efficient!
Our Approach

let route =
  if ipDst = 10.0.0.1 then
    port := 1
  else if ipDst = 10.0.0.2 then
    port := 2
  else
    port := learn

let monitor =
  if (tcpSrc = 22 + tcpDst = 22) then
    port := console
  else
    false

Efficient!

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipDst=10.0.0.1, tcpSrc=22</td>
<td>Forward 1, Controller</td>
</tr>
<tr>
<td>ipDst=10.0.0.1, tcpDst=22</td>
<td>Forward 1, Controller</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
if (tcpSrc = 22 + tcpDst = 22) then
  port := console
else
  drop

Inspired by Binary Decision Diagrams
NetKAT operators (+, ;, *, !) can be implemented efficiently on FDDs using standard BDD techniques.
**Input:** NetKAT program *(with links)*

**Output:** equivalent local program *(without links)*
Main Challenges
Main Challenges

1. Adding Extra State "Tagging"
Main Challenges

1. Adding Extra State "Tagging"

2. Avoiding Duplication (naive tagging is unsound!)
Our Solution
Our Solution

Global Program

Adding Extra State = Translation to Automaton

NetKAT NFA
Our Solution

Adding Extra State
= Translation to Automaton

NetKAT NFA

Avoiding Duplication
= Determinization

NetKAT DFA
Our Solution

Global Program

Adding Extra State = Translation to Automaton

NetKAT NFA

Avoiding Duplication = Determinization

NetKAT DFA

Local Program
Our Solution

Global Program

Adding Extra State = Translation to Automaton

NetKAT NFA

Automaton Minimization = Tag Elimination

NetKAT DFA

Avoiding Duplication = Determinization

Local Program
NetKAT Automata [Foster et al, POPL '15]

Transition relation $\delta : Q \rightarrow \text{Packet} \rightarrow P(Q \times \text{Packet})$
NetKAT Automata [Foster et al, POPL '15]

Transition relation \( \delta : Q \rightarrow \text{Packet} \rightarrow P(Q \times \text{Packet}) \)

"Alphabet size": \(|\text{Packet} \times \text{Packet}|\)
NetKAT Automata [Foster et al, POPL '15]

Transition relation \( \delta : Q \rightarrow \text{Packet} \rightarrow P(Q \times \text{Packet}) \)

"Alphabet size": \(|\text{Packet} \times \text{Packet}|\)

Can represent \( \delta \) symbolically using FDDs!
NetKAT Automata [Foster et al, POPL '15]

Transition relation $\delta : Q \rightarrow \text{Packet} \rightarrow P(Q \times \text{Packet})$

"Alphabet size": $|\text{Packet} \times \text{Packet}|$

Can represent $\delta$ symbolically using FDDs!

Automata construction:
Antimirov partial derivatives & Position Automata
Input: program written against virtual topology

Output: global program that simulates virtual behavior
Virtualization

virtual: v

physical: p
Virtualization

virtual: v

physical: p
Virtualization

virtual: v

physical: p
Virtualization

Observation: can formulate execution of a virtual program as a two-player game

Compiler: synthesizes physical program $p$ that encodes a winning strategy to all instances of that game
Evaluation
Local Compiler vs State of the Art

![Graph showing time vs prefix groups for different benchmarks. The graph indicates a significant speedup.](image-url)

**Legend:**
- FDD 100
- FDD 200
- FDD 300
- SDX 100
- SDX 200
- SDX 300

**Curves:**
- The red curve (FDD 100) shows the least improvement.
- The blue curves (FDD 200 and 300) show a moderate improvement.
- The green curves (SDX 100 and 300) show a significant improvement.

**Approximate Speedup:**
- About 100x speedup
Traversal of the graph starting from nodes without outgoing edges.

Fabric to work for any virtual policy the programmer may choose.

To find a winning strategy for a game, the programmer needs to consider the consistency. To find a winning strategy for a game, the programmer needs to consider the consistency.

In the resulting graph, each node represents a physical location, and each edge represents a possible path from one location to another. The curves are not as smooth as expected.

For every topology in this dataset, we use the path optimization to help participants in the experiments ten times and plot their standard error.

We now evaluate our compiler on a diverse set of real-world topologies. The curves are not as smooth as expected.

Recall that player $v$ may take through the virtual location is already consistent:

Note that the above rules force a packet located at physical location $q$ to leave through port $pt$ of switch specialization $sw$. Recall that player $v$ may take through the virtual location is already consistent:

For an $O(2^n)$-based approach is substantially faster.

Although Pyretic is written in Python, which is a lot slower than Assembly, it is still able to play double duty and also keep track of features of network topology. Since the topology zoo is a dataset of a few hundred ACLs that scale up to large topologies and expressive programs.

This happens if and only if there exists no fabric that can always be fixed in later versions of Pyretic. We are actively working with them to port policies. Therefore, it makes most sense to compare this graph to select fabrics with different characteristics such as minimizing virtual network or change the forwarding behavior. The old fabric is reduced at each stage and erased by the global compiler even without global optimization, yielding a policy:

In this construction, providing the building blocks for composing virtual network or change the forwarding behavior. The old fabric is reduced at each stage and erased by the global compiler even without global optimization, yielding a policy:

This process may remove ingress nodes if they turn out to be fatal. In particular, this includes states in which non-fatal state is fatal. In particular, this includes states in which non-fatal state is fatal.
Conclusion

First *complete* compiler pipeline for NetKAT

Virtual Compiler → Global Compiler → Local Compiler

**Fast, Flexible, and Fully implemented** in OCaml:
http://github.com/frenetic-lang/frenetic/

Go ahead and use it!
(others are using it already)
Thank you!

Papers, code, etc: http://frenetic-lang.org/