Programming (Software-Defined) Networks

David Walker
Princeton University
http://www.cs.princeton.edu/~dpw
http://frenetic-lang.org
The Frenetic Team

**Cornell:**
Carolyn Anderson (alum)
Shrutarshi Basu
Rebecca Coombes
**Nate Foster (faculty)**
Dexter Kozen (faculty)
Stephen Gutz (alum)
Matthew Milano
Mark Reitblatt
Emin Gün Sirer (faculty)
Robert Soulé (alum)
Laure Thompson
Alec Story (alum)
Todd Warszawski (alum)

**Princeton:**
Ryan Beckett
Rob Harrison (alum)
Xin Jin
Nanxi Kang
Naga Praveen Katta
Jennifer Gossels
Michael Greenberg (soon Pomona)
Matthew Meola
Chris Monsanto (alum)
Srinivas Narayana
Josh Reich (alum)
**Jen Rexford (faculty)**
Cole Schlesinger (alum)
David Walker (faculty)

**UMass:**
Arjun Guha (faculty)
Hypothesis:

The verification, synthesis and PL communities have their biggest opportunity since Intel screwed up division.
Where?
Data Centers

Why?
Complexity + Money + Control
Data Center
Fun Facts

MS Chicago:
• 700,000 sq ft
• 40 ft shipping containers
  – 2k servers/container
• capacity of ~224k servers
At 12:47 AM PDT on April 21st, a network change was performed as part of our normal scaling activities...

During the change, one of the steps is to shift traffic off of one of the redundant routers...

The traffic shift was executed incorrectly and the traffic was routed onto the lower capacity redundant network.

This led to a “re-mirroring storm”...

During this re-mirroring storm, the volume of connection attempts was extremely high and nodes began to fail, resulting in more volumes left needing to re-mirror. This added more requests to the re-mirroring storm...

The trigger for this event was a network configuration change.

April 21, 2011
http://aws.amazon.com/message/65648/
Those outages cost a lot

Aug 13, 2013, Amazon was down for roughly 40 minutes.

- $1,100/second\(^1\)
- $4.8 million total\(^2\)

\(^1\) http://www.buzzfeed.com/mattlynley/the-high-cost-of-an-amazon-outage#.lm8dL268x

\(^2\) http://www.geekwire.com/2013/amazon-lost-5m-40-minutes/
These outages cost a lot

Where we you from 3:52 to 3:57pm on August 17, 2013?

Google was down. One estimate\(^1\) placed the losses at $545,000

A 40% reduction in global website traffic was observed

Lots more problems ...

Not all errors manifest themselves as user-facing problems

– Some errors "merely" cause internal packet loss, retransmission, reduced latency, increased energy usage

MS scale-out checked by Network Optimized Datalog (NoD)

– 1 alert per day
– 3-4 buggy IP ranges
– 16K faulty addresses/range
Control

A single entity controls massive data centers. They can re-engineer them for reliability using formal methods.

And they are!!

– "Use of formal methods at Amazon web services"¹
  • [Newcombe et al, 2014]

– "Checking beliefs in dynamic networks"²
  • [Lopes et al, 2015]

Start-ups: Veriflow Systems, Forward Networks

NETWORK PROGRAMMING
Traditional Networks
H1 can generate a packet with an address (say H2); the packet finds its way along some path to that address
Goal

H1 can generate a packet with an address (say H2); the packet finds its way along some path to that address
H1 can generate a packet with an address (say H2); the packet finds its way along some path to that address.
H1 can generate a packet with an address (say H2); the packet finds its way along some path to that address
Data Plane

Router A's Flow Table

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Action</th>
<th>Bytes</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>01010</td>
<td>Drop</td>
<td>200</td>
<td>10</td>
</tr>
<tr>
<td>010*</td>
<td>Forward(3)</td>
<td>100</td>
<td>3</td>
</tr>
</tbody>
</table>

H2 in this range
Somehow, someone or something must decide WHAT RULES go in to the data plane. The **CONTROL PLANE** makes those decisions.
Traditional Control Plane

Each router runs an instance of a distributed algorithm. BGP. OSPF. GRE. RIP ... 

The routers exchange messages according to pre-defined protocols. Based on information received, decisions are made about how to populate the data plane.
Traditional Control Plane

Each router runs an instance of a distributed algorithm. BGP, OSPF, GRE, RIP ...

"I can reach H1 along path A"
Traditional Control Plane

Each router runs an instance of a distributed algorithm. BGP, OSPF, GRE, RIP...

"I can reach H1 along path A"

I can reach H1 along path CBA
Traditional Control Plane

Each router runs an instance of a distributed algorithm. BGP, OSPF, GRE, RIP ...

"I can reach H1 along path A"

I can reach H1 along path CBA

My data plane should send H1 packets out port 7
Configuring Traditional Control Plane

Configuration scripts set parameters of the algorithms
Hard to understand & change

Poor interfaces for programming
- Set parameters of algorithms OSPF, BGP
- Rather than program with a general, uniform language

Closed equipment
- Software bundled with hardware
- Slow protocol standardization

Few people can make changes
- "Hello? Cisco?"
- "Hello, this is Cisco! :-) How can I help you?"
- "This is Google. Please improve your load balancer."
- ... <dial tone> ...
- "We seem to have lost the connection. Damn you AT&T!"
Many Opportunities

There are \textit{a lot} of parameters to set on \textit{a lot} of separate devices in separate configs

– problem 1: check that devices have "good" configurations (and then fix the bad ones)
  • Batfish\textsuperscript{1} [NSDI 2015], BagPipe\textsuperscript{2}

– problem 2: synthesize a consistent set of parameters for devices from high-level specs
  • Narain\textsuperscript{3} [Lisa 05], Margrave\textsuperscript{4} [Lisa 10]

– problem 3: develop better languages for traditional configurations Nettle\textsuperscript{5} [DSL 09]

\textsuperscript{1} \url{http://research.microsoft.com/en-us/um/people/ratul/papers/nsdi2015-batfish.pdf}
\textsuperscript{2} \url{http://www.konne.me/bagpipe/}
\textsuperscript{3} \url{https://www.usenix.org/legacy/event/lisa05/tech/full_papers/narain/narain.pdf}
\textsuperscript{4} \url{http://www.margrave-tool.org/}
\textsuperscript{5} \url{http://haskell.cs.yale.edu/?post_type=publication&p=350}
**Traditional:**
Each router contains control + data plane

Configure by setting alg. parameters

Routers communicate data needed by control-plane algorithm using a different standardized protocol for each algorithm

**Software-Defined (SDN):**
Each router contains data plane only

Controller runs algorithm to configure global network; sends (OpenFlow) requests to routers
An Example SDN Application

Network Events
• Forwarding table miss

Control Messages
• Install rules

Problem 5:
Design a programming system that makes it easy to construct reliable, efficient SDN applications.
SDN was Great, PL people can make it Greater

The good:

- simple, uniform abstraction
  - a switch = a table of rules
- never underestimate the power of simplicity!
- also "complete for forwarding"

The bad:

- provides the "assembly language" level of abstraction
MODULAR SDN POLICIES


Anderson, Baptist, Foster, Guha, Gossels, Harrison, Jin, Kozen, Monsanto, Reich, Rexford, Story, Schlesinger, Walker
We still need all the functionality of old networks:
Traffic monitoring, routing, firewalls, load balancing
The only way to engineer it is through modular design.
A Typical Programming Problem

Forwarding

“install rule to forward packets X and Y to D”

NOX
A Typical Programming Problem

Forwarding

“install rule to forward packets X and Y to D”

NOX

“forwarding packets X and Y to D”
A Typical Programming Problem

Forwarding

“install rule to forward packets X and Y to D”

NOX

Monitoring

“install rule to count number of packets X and Z that flow to D”

“forwarding packets X and Y to D”
A Typical Programming Problem

**Forwarding**

- “install rule to forward packets X and Y to D”

**Monitoring**

- “install rule to count number of packets X and Z that flow to D”

**NOX**

- “counting Z flowing to D”
- “forwarding Y to D”
- “doing random stuff with X”
An Analogy

Concurrent Thread 1

“write to X”

shared state

Concurrent Thread 2

“write to X”

“race condition on X”
OpenFlow/Nox “Solution”

Just don't divide up complex problems into smaller parts!

manually rewrite and combine forwarding and monitoring applications into one big monolithic jumble!
Anti-Modularity in NOX

Repeater

```python
def switch_join(switch):
    repeater(switch)

def repeater(switch):
    pat1 = {in_port:1}
    pat2 = {in_port:2}
    install(switch, pat1, DEFAULT, None, [output(2)])
    install(switch, pat2, DEFAULT, None, [output(1)])
```

Web Monitor

```python
def monitor(switch):
    pat = {in_port:2, tp_src:80}
    install(switch, pat, DEFAULT, None, [])
    query_stats(switch, pat)

def stats_in(switch, xid, pattern, packets, bytes):
    print bytes
    sleep(30)
    query_stats(switch, pattern)
```

Repeater/Monitor

```python
def switch_join(switch):
    repeater_monitor(switch)

def repeater_monitor(switch):
    pat1 = {in_port:1}
    pat2 = {in_port:2}
    pat2web = {in_port:2, tp_src:80}
    install(switch, pat1, DEFAULT, None, [output(2)])
    install(switch, pat2, DEFAULT, None, [output(1)])
    install(switch, pat2web, HIGH, None, [output(1)])
    query_stats(switch, pat2web)

def stats_in(switch, xid, pattern, packets, bytes):
    print bytes
    sleep(30)
    query_stats(switch, pattern)
```

blue = from repeater
red = from web monitor
green = from neither
Functional Solution
1st Class SDN Policies + Combinators

- Firewall
- Repeater
- Monitoring

Controller
Functional Solution
1st Class SDN Policies + Combinators
Functional Solution

1st Class SDN Policies + Combinators

- Firewall
- Repeater
- Monitoring

combine policies from separate modules
Functional Solution
1st Class SDN Policies + Combinators
A Simple Network

Packets are *records* containing location information:

\[
\{ \text{sw} = A; \, \text{pt} = 1; \, \text{src} = H1; \, \text{dst} = H2; \, \ldots \} \]
Generic programming task: to write a function that explains how all switches in the network should process their packets.

Each such function is called a network policy:

policy : packet -> packet set
Basic Features

Predicates:
\[ sw = A \land pt = 1 \]
// packets at switch A, port 1 left untouched
// others dropped

Actions:
\[ dst \leftarrow 10.0.0.1 \]
// modify the dest field
\[ pt \leftarrow 2 \]
// modify the pt field (ie: move packet to port 2)

Sequential composition:
\[ dst \leftarrow 10.0.0.1; pt \leftarrow 2 \]
// modify the dst field \textit{then} the port field

Parallel composition:
\[ (pt \leftarrow 1) + (pt \leftarrow 2) \]
// copy the packet and \textit{do both} operations
A Simple Network

Tasks:
- **Forwarding**: Transfer packets between hosts
- **Access Control**: Block SSH packets

\[
\text{forward} = \\
\quad (\text{dst} = \text{H1}; \text{pt} < -1) \\
\quad + (\text{dst} = \text{H2}; \text{pt} < -2) \\
\text{ac} = \\
\quad \sim(\text{typ} = \text{SSH}); \text{forward}
\]

block on both A & B
A Simple Network

acA =
  sw = A; \neg (typ = SSH); forward
  + sw = B; forward

acB =
  sw = A; forward
  + sw = B; \neg (typ = SSH); forward

ac =
  \neg (typ = SSH); forward

block on both A & B

block on A

block on B
A Simple Network

Are all SSH packets dropped?

Do all non-SSH packets sent from H1 arrive at H2?

Are the optimized policies equivalent to the unoptimized one?
Encoding Topologies

$$t = \( \left( \text{sw} = A \& \text{pt} = 2; \text{sw} \leftarrow B; \text{pt} \leftarrow 1 \right) + \left( \text{sw} = B \& \text{pt} = 1; \text{sw} \leftarrow A; \text{pt} \leftarrow 2 \right) \)$$

$$\text{net} = \text{ac}; t; \text{ac}$$
Encoding Topologies

H1

A

B

H2

\[ t = \ldots \]

\[ \text{net} = (ac; t)^*; ac \]

Kleene iteration:

\[ p^* = id + p + pp + \ldots \]
Encoding Networks

ac = ...  
t = ...  
net = (ac; t)*; ac

edge = sw=A & pt=1 || sw=B & pt=2

net = edge; (ac; t)*; ac; edge

net is a function that moves packets:
A1 ==> B2
B2 ==> A1

and also moves packets:
A1 ==> A2
A2 ==> A1
B1 ==> B2
B2 ==> B1
## Summary So Far

### Policies

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>p, q, r :=</td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>// filter according to a</td>
</tr>
<tr>
<td></td>
<td>f &lt;- v</td>
</tr>
<tr>
<td></td>
<td>// update field f to v</td>
</tr>
<tr>
<td></td>
<td>p ; q</td>
</tr>
<tr>
<td></td>
<td>// do p then q</td>
</tr>
<tr>
<td></td>
<td>p + q</td>
</tr>
<tr>
<td></td>
<td>// do p and q in parallel</td>
</tr>
<tr>
<td></td>
<td>p*</td>
</tr>
<tr>
<td></td>
<td>// do p zero or more times</td>
</tr>
</tbody>
</table>

### Predicates

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a, b, c :=</td>
<td>drop</td>
</tr>
<tr>
<td></td>
<td>// drop all packets</td>
</tr>
<tr>
<td></td>
<td>id</td>
</tr>
<tr>
<td></td>
<td>// accept all packets</td>
</tr>
<tr>
<td></td>
<td>f = v</td>
</tr>
<tr>
<td></td>
<td>// field f matches v</td>
</tr>
<tr>
<td></td>
<td>~a</td>
</tr>
<tr>
<td></td>
<td>// negation</td>
</tr>
<tr>
<td></td>
<td>a &amp; b</td>
</tr>
<tr>
<td></td>
<td>// conjunction</td>
</tr>
<tr>
<td></td>
<td>a</td>
</tr>
<tr>
<td></td>
<td>// disjunction</td>
</tr>
</tbody>
</table>

### Network Encoding

in; (policy; topology)*; policy; out
Summary So Far

**Kleene Algebra**

\[ p, q, r ::= \]
- \( a \) // filter according to \( a \)
- \( f <- v \) // update field \( f \) to \( v \)
- \( p ; q \) // do \( p \) then \( q \)
- \( p + q \) // do \( p \) and \( q \) in parallel
- \( p^* \) // do \( p \) zero or more times

**Predicates**

\[ a, b, c ::= \]
- \( \text{drop} \) // drop all packets
- \( \text{id} \) // accept all packets
- \( f = v \) // field \( f \) matches \( v \)
- \( \neg a \) // negation
- \( a \& b \) // conjunction
- \( a \| b \) // disjunction

**Boolean Algebra + Kleene Algebra**

\[ \text{Network Encoding} = \text{Kleene Algebra with Tests} \]
Equational Theory

\[ \text{net}1 \approx \text{net}2 \]

For programmers (or synthesizers):
- a system for reasoning about programs as they are written

For compiler writers:
- a means to prove their transformations correct

For verifiers:
- sound and complete with a PSPACE decision procedure [Foster et al., POPL 2015]
Equational Theory

Boolean Algebra:

- $a \land b \equiv b \land a$
- $a \land \neg a \equiv \text{drop}$
- $a \lor \neg a \equiv \text{id}$

Kleene Algebra:

- $(a; b); c \equiv a; (b; c)$
- $a; (b + c) \equiv (a; b) + (a; c)$
- $p^* \equiv \text{id} + p; p^*$

Packet Algebra:

- $f \gets n; f = n \equiv f \gets n$
- $f = n; f \gets n \equiv f = n$
- $f \gets n; f \gets m \equiv f \gets m$
- if $f \neq g$: $f = n; g \gets m \equiv g \gets m; f = n$
- $f \gets n; g \gets m \equiv g \gets m; f \gets n$
- if $m \neq n$: $f = n; f = m \equiv \text{drop}$
- $f = 0 + \ldots + f = n \equiv \text{id}$ (finite set of possible values in $f$)
Using the Theory

Are all SSH packets dropped?

\[ \text{typ} = \text{SSH}; \quad \text{net} \approx \text{drop} \]

Do all non-SSH packets sent from H1 arrive at H2?

\[ \sim \text{typ} = \text{SSH}; \quad \text{sw} = \sim \text{typ}; \quad \text{pt} \Rightarrow \text{drop} \]

forward = (dst = H1; pt <- 1) + (dst = H2; pt <- 2)
ac = ~(typ = SSH); forward
t = ...
edge = ...
net = edge; (ac; t)*; ac; edge
Using the Theory

Are all SSH packets dropped?

\[ \text{typ} = \text{SSH}; \text{net} \approx \text{drop} \]

Do all non-SSH packets destined for H2, sent from H1 arrive at H2?

\[ \sim\text{typ} = \text{SSH}; \text{dst} = \text{H2}; \text{sw} = \text{A}; \text{pt} = 1; \text{net} \approx \sim \text{typ} = \text{SSH}; \text{dst} = \text{H2}; \text{sw} = \text{A}; \text{pt} = 1; \text{sw} \leftarrow \text{B}; \text{pt} \leftarrow 2 \]

<table>
<thead>
<tr>
<th>forward</th>
<th>ac</th>
<th>t</th>
<th>edge</th>
<th>net</th>
</tr>
</thead>
<tbody>
<tr>
<td>( (\text{dst} = \text{H1}; \text{pt} \leftarrow 1) ) + ( (\text{dst} = \text{H2}; \text{pt} \leftarrow 2) )</td>
<td>( \sim(\text{typ} = \text{SSH}) ); forward</td>
<td>( \ldots )</td>
<td>( \ldots )</td>
<td>( \text{edge} ); ( (\text{ac}; \text{t})^* ); ( \text{ac} ); ( \text{edge} )</td>
</tr>
</tbody>
</table>
Traffic Isolation

Programmer 1 connects H1 and H2:

\[
\text{polA1} = \text{sw} = A; ( \begin{align*} pt &= 1; pt &\leftarrow 2 + \\ pt &= 2; pt &\leftarrow 1 \end{align*} )
\]

\[
\text{polB1} = \text{sw} = B; ( \ldots )
\]

\[
\text{pol1} = \text{polA1} + \text{polB1}
\]

\[
\text{net1} = (\text{pol1}; t)^
\]

Programmer 2 connects H3 and H4:

\[
\text{polA2} = \text{sw} = B; ( \begin{align*} pt &= 3; pt &\leftarrow 2 + \\ pt &= 1; pt &\leftarrow 3 \end{align*} )
\]

\[
\text{polB2} = \text{sw} = A; ( \ldots )
\]

\[
\text{pol2} = \text{polA2} + \text{polB2}
\]

\[
\text{net3} = ((\text{pol1} + \text{pol2}); t)^
\]

// traffic from H2 goes to H1 and H4!
Traffic Isolation

A network slice is a light-weight abstraction designed for traffic isolation:

- Traffic outside the slice satisfying \text{in} enters the slice.
- Traffic inside the slice obeys the policy.
- Traffic inside the slice obeys the policy.
- Slices are just a little syntactic sugar on top of NetKAT.
A network slice is a light-weight abstraction designed for traffic isolation:

\[
\begin{align*}
\text{edge1} &= \text{sw = A & pt = 1 || sw = B & pt = 2} \\
\text{slice1} &= \{\text{edge1}\} \text{ pol1 } \{\text{edge1}\} \\
\text{edge2} &= \text{sw = A & pt = 3 || sw = B & pt = 3} \\
\text{slice2} &= \{\text{edge2}\} \text{ pol2 } \{\text{edge2}\}
\end{align*}
\]

**Theorem:** \((\text{slice1}; t)^* + (\text{slice2}; t)^* \approx ((\text{slice1} + \text{slice2}); t)^*\)

Packet copied and sent through slice1 and slice2 networks *separately*.
Packet runs through network that *combines* slice1 and slice2.
A network slice is a light-weight abstraction designed for traffic isolation:

- edge1 = sw = A & pt = 1 || sw = B & pt = 2
- slice1 = \{edge1\} pol1 \{edge1\}
- edge2 = sw = A & pt = 3 || sw = B & pt = 3
- slice2 = \{edge2\} pol2 \{edge2\}

Theorem: edge1; (slice1; t)\* \(\approx\) edge1; ((slice1 + slice2); t)\*

Consider those packets at the edge1 of the slice.

Can't tell the difference between slice1 alone and slice1 + slice2.
NetKAT can be implemented with OpenFlow

Forward =
(dst = H1; pt <- 1)
+ (dst = H2; pt <- 2)

ac =
~(typ = SSH); forward

Flow Table for Switch 1:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>typ = SSH</td>
<td>drop</td>
</tr>
<tr>
<td>dst=H1</td>
<td>fwd 1</td>
</tr>
<tr>
<td>dst=H2</td>
<td>fwd 2</td>
</tr>
</tbody>
</table>

Flow Table for Switch 2:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>typ = SSH</td>
<td>drop</td>
</tr>
<tr>
<td>dst=H1</td>
<td>fwd 1</td>
</tr>
<tr>
<td>dst=H2</td>
<td>fwd 2</td>
</tr>
</tbody>
</table>

Theorem: Any NetKAT policy p that does not modify the switch field can be compiled into an equivalent policy in “OpenFlow Normal Form.” [POPL '12, '14]

See [Smolka et al ICFP 2015] for new, fast BDD-based compilation algorithms.
NETWORK UPDATES
Updates

Even when **old policy** is "good" and **new policy** is "good" problems can happen when transitioning
- packets dropped, forwarded incorrectly, loops
Problem 7: How to update a network from one policy to another while maintaining key semantic invariants?
Example: Distributed ACL

Security Policy

<table>
<thead>
<tr>
<th>Src</th>
<th>Traffic</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web</td>
<td>Allow</td>
<td></td>
</tr>
<tr>
<td>Non-web</td>
<td>Drop</td>
<td></td>
</tr>
<tr>
<td>Any</td>
<td>Allow</td>
<td></td>
</tr>
</tbody>
</table>

Configuration A

Process black-hat traffic on F1
Process white-hat traffic on \{F2,F3\}

Configuration B

Process black-hat traffic on \{F1,F2\}
Process white-hat traffic on F3

The ACL Example:
Just one invariant we might want to preserve.

What general properties should the network update operation have?
General Criterion

[Reitblatt, Foster, Rexford, Schlesinger, Walker, SIGCOMM 2012 ]

Per-packet consistency:

Definition: Every individual packet processed exclusively by either policy P1 or policy P2 (not a mix)

Theorem: if a trace property T holds of both P1 and P2 independently, then it holds of all packets flowing through the network

Example trace properties:
Access control, reachability, waypointing, no loops

Per-flow consistency:

Definition: Well-defined set of related packets are all processed by policy P1 or policy P2 (not a mix)
Benefits

Simplifies programming practice
- Update configurations in chunks
- Programmer doesn’t think about concurrency
- A class of programming errors goes away
- Run-time system handles the details

Cheaper mechanical verification
- Check each policy P1, P2, P3 in isolation
- No transitional states need to be verified!
- Conceptually: A generic way to bootstrap a static network policy analyzer in to a dynamic one
  - We specified LTL properties and tried NuSMV
  - You can try Header Space Analysis or Anteater!
Mechanism: Two-Phase Update

Set-up: Pre-process policies
  - Edge: Stamp packet with version #
  - Interior: Policy conditioned on version #

Phase 1: Unobservable updates
  - Add rules for P2 in the interior
  - … matching on version # of P2

Phase 2: One-touch updates
  - Add rules to stamp packets with version #P2 at the edge

Clean-up: Remove old rules
  - Wait for some time, then remove all version # P1 rules
Alternatives

Goal: Avoid naive two-phase update
- Naïve version touches every switch
- Doubles rule space requirements

Optimization: Incremental update [Katta, Rexford, Walker]
- Shift policy one traffic slice at a time
- Trades update time vs. rule space

Optimization: Supply invariants [McClurg, Hojjat, Cerny, Foster]
- Specify the LTL invariants you need to preserve
- Use a model checker to search for update sequences

Improved specs: Regular update lang. [Saha, Prabhu, Mahdu]
- Specify the invariants you need to preserve
- Map to SAT problem; synthesize update
Consistent Updates

Main point:

• *Programmers need only concern themselves with the properties of each policy (A, B, ...) in isolation.*

Effects:

• Another kind of modularity for SDN programmers.
• Brain power shifted away from tricky transitions.
MOVING FORWARD
NetKAT: Still low level & limited

- Explicitly programmed paths
- Stateless
- Lacking QOS controls
- Lacking monitoring and debugging support
- Limited actions (get and set packet headers)
- Operates over fixed protocols (IPv4 vs. DESTP)

Dave's Extra-Special Transport Protocol (DESTP)
The Big Switch Model

Big switch model abstracts away from physical topology

Operator specifies constraints:
• access control, reachability, QOS constraints, fault tolerance

Problem 5: System synthesizes concrete policy
• SIMPLE [Qasi et al 2013], MERLIN [Soule et al, 2013], L [Padon et al 2015]
• Optimization [He 2008, Ghosh et al 2013]
Alternate Platforms: P4 & more

- Openflow was designed as a simple first-cut interface
- It doesn't support state, contains a small set of primitive actions and doesn't allow programmers to control packet formats.

- Alternate architectures:
  - P4
  - CPU
  - CPU + FPGA
  - CPU + GPU?

- Problem 6: Designing high-level programming models & automatically splitting computation between devices
Many networks contain middleboxes
  • can be as many middleboxes as routers

Middleboxes contain stateful computations
  • dynamic access control and intrusion detection
  • content caching
  • programming and/or verification and synthesis support:
    • Maple, FlowLog, NetEgg, Slick, L, Vericon
WRAP UP
C.2.3 optimize one or more objective functions; we address the controller modules, which manage different aspects of the network (e.g., link bandwidth or switch table slots). Our multiple controller functions may be competing for resources focused on making them scalable, reliable, and efficient.

Tools

Open Networking Summit: Boosting EEC's Call for Papers

Topical Areas:
- Architecture
- Dataplane
- Platforms
- Policy
- Security
- Technologies

ONF ONOS
- P4

P4

[Bosshart, Daly, Izzard, McKeown, Rexford, Schlesinger, Talayco, Vahdat, Varghese, Walker]

The next generation of the OpenFlow protocol: Custom packet specs & configurable table graphs
Networks Need Abstractions

More Tools to Support Them

- Declarative languages
- Program synthesis
- Debugging
- Testing
- Verification

Relevant stats:
- Google lost $545,000 during a 5-minute outage
- Amazon lost $1,100/second during a 25-minute outage
- Nicira bought for 1.2 billion
Cornell:
Carolyn Anderson
Shrutarshi Basu
Rebecca Coombes
Nate Foster
Dexter Kozen
Stephen Gutz
Matthew Milano
Mark Reitblatt
Emin Gün Sirer
Robert Soulé
Laure Thompson
Alec Story
Todd Warszawski

Princeton:
Michael Greenberg
Rob Harrison
Nanxi Kang
Naga Praveen Katta
Xin Jin
Matthew Meola
Chris Monsanto
Nayden Nedev
Josh Reich
Jen Rexford
Cole Schlesinger
David Walker

UMass:
Arjun Guha

http://frenetic-lang.org
http://cs.princeton.edu/~dpw