Managing the Network with Merlin

Robert Soulé
Shrutarshi Basu
Robert Kleinberg
Emin Gün Sirer
Nate Foster
This talk is not about...
This talk is not about...

NetKAT: Semantic Foundations for Networks

• Carolyn Anderson (Swarthmore)
• Nate Foster (Cornell)
• Arjun Guha (UMass)
• Jean-Baptiste Jeannin (CMU)
• Dexter Kozen (Cornell)
• Cole Schlesinger (Princeton)
• David Walker (Princeton)

To appear in POPL ’14.
Programmable Networks
Programmable Networks
Programmable Networks

Controller Application

Controller Platform

Diagram showing a network with nodes connected in a grid pattern, with arrows indicating direction of data flow.
Programmable Networks

Enabling a shift from *bits and protocols* to *abstractions and applications*
Challenges

Thousands of nodes...
Heterogeneous devices...
Complex configurations...
Difficult to reason about...
Current Abstractions

Software-Defined Networks
• Maple [SIGCOMM ’13]
• Corybantic [HotNets ’13]
• Frenetic [ICFP ’11, POPL, 12’, NSDI ’13]

Middleboxes
• CoMB [NSDI ’12]
• APLOMB [SIGCOMM ’12]
• SIMPLE [SIGCOMM ’13]

End Hosts
• PANE [SIGCOMM ’13]
• EyeQ [NSDI ’13]
• ETTM [NSDI ’11]
Limitations

We still lack *unified* abstractions for programming networks...

There are *complex interactions* between components...

Progress on *verification tools* is encouraging but nascent...

Most existing systems assume a *single point of control*...
SDN Limitations

SDN is not the right abstraction for network management!

What network operators want:
- “Ensure that all traffic traverses at least one firewall.”
- “Give Hadoop traffic priority over backup traffic”
- “Let the PPT group manage their network (in Haskell?)”

What SDN provides:
Match HTTP traffic and forward it out physical port 4
Specify global network policy in a high-level declarative language

Transform policies into ones that can be delegated or enforced locally

Interpose on network traffic to ensure policy compliance
Policy Language
Formalism

Syntax
- Logical predicates
- Path expressions
- Bandwidth constraints

Properties
- Network paths
- Function sequences
- Resource usage

\[
\begin{align*}
loc & \in \text{Locations} \\
 t & \in \text{Transformation Functions} \\
pol & ::= (s_1; \ldots; s_n) \\
s & ::= q p \rightarrow e \text{ at } r \\
q & ::= \text{foreach} \mid \text{forall} \mid \text{forsome} \\
p & ::= m \mid p_1 \text{ and } p_2 \mid p_1 \text{ or } p_2 \mid ! p_1 \\
m & ::= h.f = n \\
e & ::= . \mid c \mid e\ e \mid e\ e \mid e^* \mid ! e \\
c & ::= loc \mid t \\
r & ::= \max(n) \mid \min(n)
\end{align*}
\]
Examples

( `ethType = 0x800` and `ipProto = 0x06` )
-> `h1` .* `nat` .* `dpi` .* `h2`
@ max(1GB/s)

Informally: ensure that all TCP traffic between two hosts is processed by NAT and DPI functions (in that order) and abides by a rate limit of 1GB/s.
Examples

( ipSrc = 192.168.1.1/16 and
  ipDst = 192.168.1.1/16 and
  ipProto = 0x06 and
  ipPort = 50060 )

-> .*

@ min(100MB/s)

Informally: ensure there is at least 100MB/s of bandwidth for Hadoop traffic.
Informally: ensure resource isolation between two subnetworks for a given middlebox.
Examples

true
-> ( .* fire1 .* fire2 .* |
    .* fire2 .* fire1 .* )

Informally: ensure that all traffic traverses at least two firewalls, in either order.
Examples

forall
true
-> ( .* mb1 .* )
@ max(10GB/s)

Informally: ensure that all traffic across the middlebox is capped at 10GB/s.
Flow Quantifiers

Many statements involve multiple hosts:

true -> (h1|h2) .* h3 @ max(100MB/s)
Flow Quantifiers

Many statements involve multiple hosts:

\[
\text{true} \rightarrow (h1|h2) .\ h3 \ @ \ \text{max}(100\text{MB/s})
\]

Quantifiers determine division of bandwidth:

\[
\begin{align*}
\text{foreach} \\
\text{true} \rightarrow (h1|h2) .\ h3 \ @ \ \text{max}(100\text{MB/s}) \\
\equiv \\
\text{true} \\
\rightarrow h1 .\ h3 \ @ \ \text{max}(100\text{MB/s}) \\
\text{true} \\
\rightarrow h2 .\ h3 \ @ \ \text{max}(100\text{MB/s})
\end{align*}
\]
Flow Quantifiers

Many statements involve multiple hosts:

\[
\text{true} \rightarrow (h_1|h_2) \cdot \ast h_3 @ \text{max}(100\text{MB/s})
\]

Quantifiers determine division of bandwidth:

\[
\forall \text{true} \rightarrow \ast h_3 @ \text{max}(100\text{MB/s})
\]

\[
\equiv
\]

\[
\text{true} \\
\rightarrow h_1 \cdot \ast h_3 @ \text{max}(50\text{MB/s}) \\
\text{true} \\
\rightarrow h_2 \cdot \ast h_3 @ \text{max}(50\text{MB/s})
\]
Flow Quantifiers

Many statements involve multiple hosts:

\[ \text{true} \rightarrow (h1|h2) .\ast h3 @ \text{max}(100\text{MB}/s) \]

Quantifiers determine division of bandwidth:

\[ \text{forsome} \\
\text{true} \rightarrow (h1|h2) .\ast h3 @ \text{max}(100\text{MB}/s) \]
\[ \equiv \]
\[ \text{true} \\
\rightarrow (h1|h2) .\ast h3 @ \text{max}(100\text{MB}/s) \]
Compiler
Compiler

Tasks:
- Path selection
- Bandwidth allocation
- Code generation

Challenges:
- Heterogeneous devices
- Network-wide resources

Approach:
- Encode as a constraint problem
- Solve using linear programming
Constraint Problem

Encode with standard flow conservation and capacity constraints
Optimization Criteria

**Weighted Shortest Path:**
Minimizes total number of hops in assigned paths (standard)

**Min-Max Ratio:**
Minimizes the maximum fraction of reserved capacity (balance)

**Min-Max Reserved:**
Minimizes the maximum amount of reserved bandwidth (failures)
Code Generation

**Network Switches**
- Encode paths using NetCore [POPL ’12]
- Generate tags to identify paths
- Install rules on OpenFlow switches

**Middleboxes**
- Translate functions to Click [TOCS ’00]
- Install on software middleboxes

**End Hosts**
- Generate code for Linux tc and iptables
- Experimental support for a custom Merlin kernel module based on netfilter
Delegation
Federated Control

Every network has multiple tenants
• Campuses and enterprises
• Data centers
• Wide-area

But many platforms assume a single omnipotent programmer
• SDN controllers
• Middlebox platforms

Merlin provides mechanisms for
• Delegating functionality
• Verifying policy modifications
Policy Language

Transformations

Enforcement

**Policy restriction**
Restrict global policy to a subset of the overall traffic

**Modification:**
Tenants modify the restricted policy to suit their custom needs

**Verification:**
Owner checks that the modified policy refines the original...

**Integrate:**
...and then reintegrates the modified policy back into the global policy
Delegation Example

Global policy

```plaintext
foreach true
  -> (h1|h2) .* h3
  @ max(100MB/s)
```

Restriction to host 1

```plaintext
foreach true
  -> h1 .* h3
  @ max(100MB/s)
```
Modified Policy

```plaintext
foreach (tcpDst = 80)
  -> h1 .* lb .* h3
  @ max(50MB/s)

foreach (tcpDst = 22)
  -> h1 .* dpi .* h3
  @ max(25MB/s)

foreach (!(tcpDst = 22 | tcpDst = 80))
  -> h1 .* h3
  @ max(25MB/s)
```
Verification

**Essential operation:**

Policy inclusion: \( P_1 \subseteq P_2 \)

**Algorithm:**

- Pair-wise comparison of statements
- Check for path inclusion on overlaps
- Aggregate bandwidth constraints

**Implementation:**

- Decide predicate overlap using SAT
- Decide path inclusion using NFAs
Experience
Implementation

Prototype implementation
- OCaml
- Gurobi solver
- Z3 theorem prover
- DPrle NFA library
- Linux kernel modules
- Frenetic language

Preliminary deployment
- Pronto 3290 switches
- Dell Force10 switch
- OpenVSwitch
- Dell r720 servers
Evaluation

Can Merlin simplify network management while providing end-to-end performance improvements for applications?

- Hadoop benchmark
- Microbenchmarks

Does our compilation and verification infrastructure scale?

- Simple simulations on realistic data center topologies and policies of increasing size
- Data from Benson et al. [IMC ’10]
Performance Benchmark

Experimental setup:
- Four 16-core Dell r720 servers with 32GB RAM
- Pronto 3290 switch with 1GB links
- Two applications: sorting and word count
- Input: 10GB data
- Traffic: UDP packets generated with **iperf**
- Merlin policy: reserve 90% capacity for Hadoop
Experimental setup:
- Two 16-core Dell r720 servers with 32GB RAM
- Pronto 3290 switch with 1GB links
- Traffic: TCP/UDP flows/packets generated with `iperf`
- Merlin policy: reserve 70% capacity for TCP
Microbenchmark: TCP/TCP

Experimental setup:
• Two 16-core Dell r720 servers with 32GB RAM
• Pronto 3290 switch with 1GB links
• Traffic: TCP flows generated with iperf
• Merlin policy: reserve 70% capacity for TCP
Scalability: Compiler

Experimental setup:

- 16-core Dell r720 server with 32GB RAM
- Topologies from Benson et al. [IMC ’10]
- Statements encode shortest paths between hosts
Scalability: Compiler

Experimental setup:
- 16-core Dell r720 server with 32GB RAM
- Topologies from Benson et al. [IMC ‘10]
- Statements encode shortest paths between hosts
Scalability: Verification

Experimental setup:
• 16-core Dell r720 server with 32GB RAM
• Topologies from Benson et al. [IMC ’10]
• Statements encode shortest paths between hosts

Increasing
Statements

Increasing
Path
Expressions

Increasing
Bandwidth
Constraints
Wrapping Up
Conclusion

Merlin is a *unified* network management framework...

It supports heterogeneous devices...

It handles paths, network functions, and bandwidth...

It generates complex configurations...

It provides delegation and automatic verification...
Thank you!

Collaborators
• Robert Soulé (Postdoc)
• Shrutarshi Basu (PhD)
• Robert Kleinberg (Faculty)
• Emin Gün Sirer (Faculty)

Formal Foundations for Networks
Seminar 30-0613, February 2015
Co-organizers:
• Nikolaj Bjørner (MSR)
• Nate Foster (Cornell)
• Brighten Godfrey (UIUC)
• Pamela Zave (AT&T)