Merlin: A Language for Provisioning Network Resources

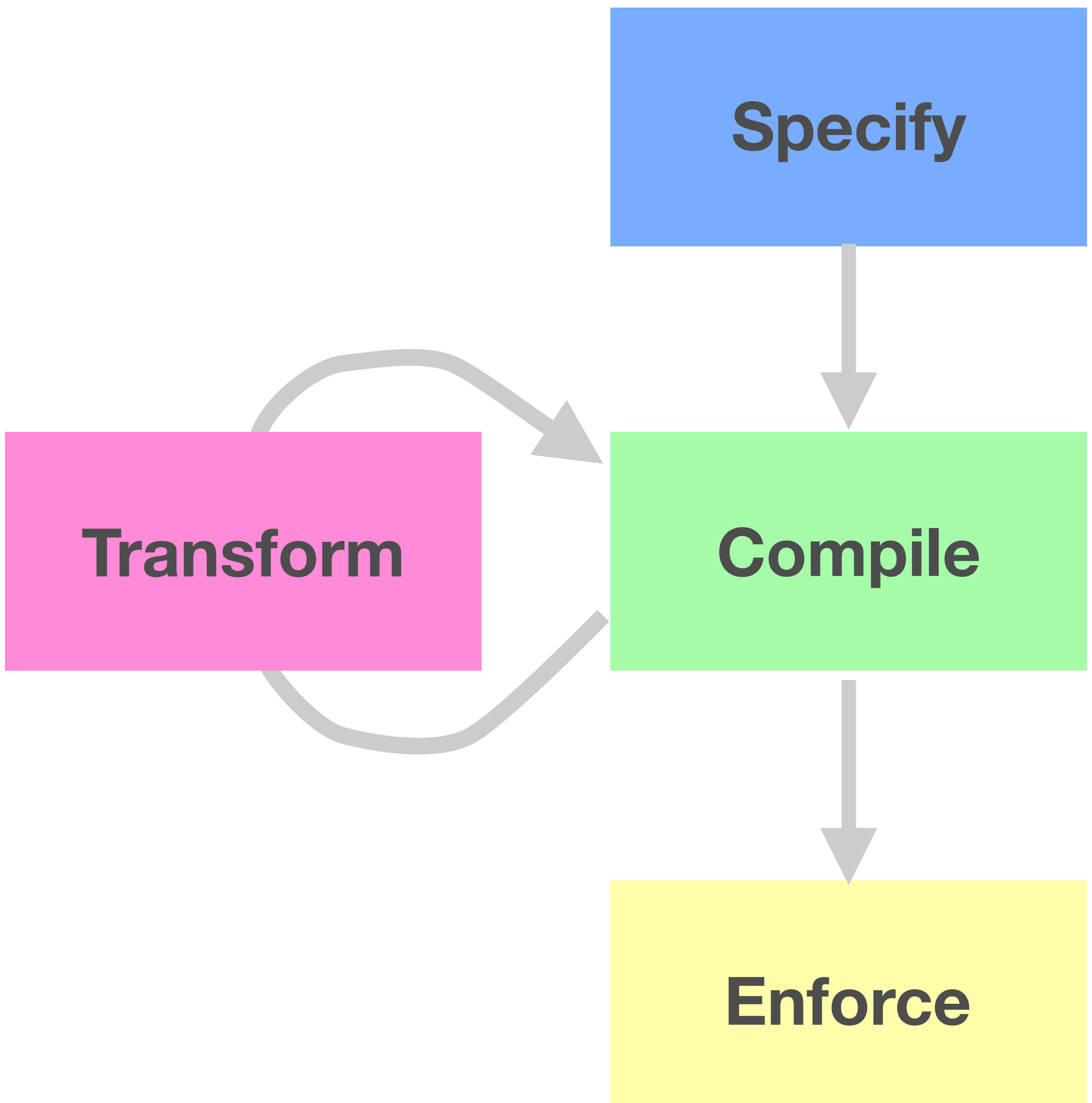
Robert Soulé, Shrutarshi Basu, Parisa Marandi, Fernando Pedone, Robert Kleinberg, Emin Gün Sirer, and Nate Foster

University of Lugano and Cornell University

How to Program The Network?

- Existing SDN languages focus mostly on packet forwarding
- Ignore other vital network features like bandwidth, packet processing, etc.
- Network orchestration frameworks expose extremely simple APIs (if at all)

Merlin Approach



Specify global network policy in a **high-level declarative language**.

Map to a constraint problem. Provision network, select paths, and decide function placement.

Delegate to tenants for refinement. Verify that modifications conform to global policy. Re-solve if necessary.

Generate device-specific code and configuration to enforce policy.

Outline of This Talk

- Motivation
- **Policy Language**
- Compiler
- Dynamic Adaptation
- Evaluation
- Conclusions

Policy Language

Specify network behavior with high-level abstractions

Policy Basics

Informally: Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.

Policy Basics

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.**

```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Policy Basics

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.**

```
[ x :                                          } Identifier
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Policy Basics

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.**

```
[ x :                                            } Identifier
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &   } Predicates identify which traffic
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Policy Basics

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.**

```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

} *Identifier*

} *Predicates identify which traffic*

} *Regular expressions for paths, functions*

Policy Basics

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s.**

```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

} *Identifier*

} *Predicates identify which traffic*

} *Regular expressions for paths, functions*

} *Caps or guarantees for bandwidth*

Expressive Formulas

Informally: **Place an aggregate bandwidth cap on FTP data and control traffic. Data traffic must be processed by a DPI function.**

Expressive Formulas

Informally: **Place an aggregate bandwidth cap on FTP data and control traffic. Data traffic must be processed by a DPI function.**

```
[ y : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 20) -> .* dpi .*
  z : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

Expressive Formulas

Informally: **Place an aggregate bandwidth cap on FTP data and control traffic. Data traffic must be processed by a DPI function.**

```
[ y : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and     } FTP data
       tcp.dst = 20) -> .* dpi .*
  z : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

Expressive Formulas

Informally: **Place an aggregate bandwidth cap on FTP data and control traffic. Data traffic must be processed by a DPI function.**

```
[ y : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 20) -> .* dpi .*
  z : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

} *FTP data*

} *FTP control*

Expressive Formulas

Informally: **Place an aggregate bandwidth cap on FTP data and control traffic. Data traffic must be processed by a DPI function.**

```
[ y : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 20) -> .* dpi .*
  z : (eth.src = 00:00:00:00:00:01 and
       eth.dst = 00:00:00:00:00:02 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

} *FTP data*

} *FTP control*

} *Bandwidth constraints written as formulas*

Syntactic Sugar

Informally: Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s (*again*).

```
srcs := {00:00:00:00:00:01}
dsts := {00:00:00:00:00:02}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at max(100MB/s)
```

Syntactic Sugar

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s (*again*).**

```
srcs := {00:00:00:00:00:01}
dsts := {00:00:00:00:00:02}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at max(100MB/s)
```

} ***Set literals***

Syntactic Sugar

Informally: **Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s (*again*).**

```
srcs := {00:00:00:00:00:01}
dsts := {00:00:00:00:00:02}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at max(100MB/s)
```

} *Set literals*

} *Set operators and iterators*

Syntactic Sugar

Informally: Ensure that HTTP traffic between two hosts is processed by NAT and DPI functions (in that order) and gets a guarantee of 100MB/s (*again*).

```
srcs := {00:00:00:00:00:01}
dsts := {00:00:00:00:00:02}
foreach (s,d) in cross(srcs,dsts):
    tcp.dst = 80 ->
    ( .* nat .* dpi .*) at max(100MB/s)
```

} *Set literals*

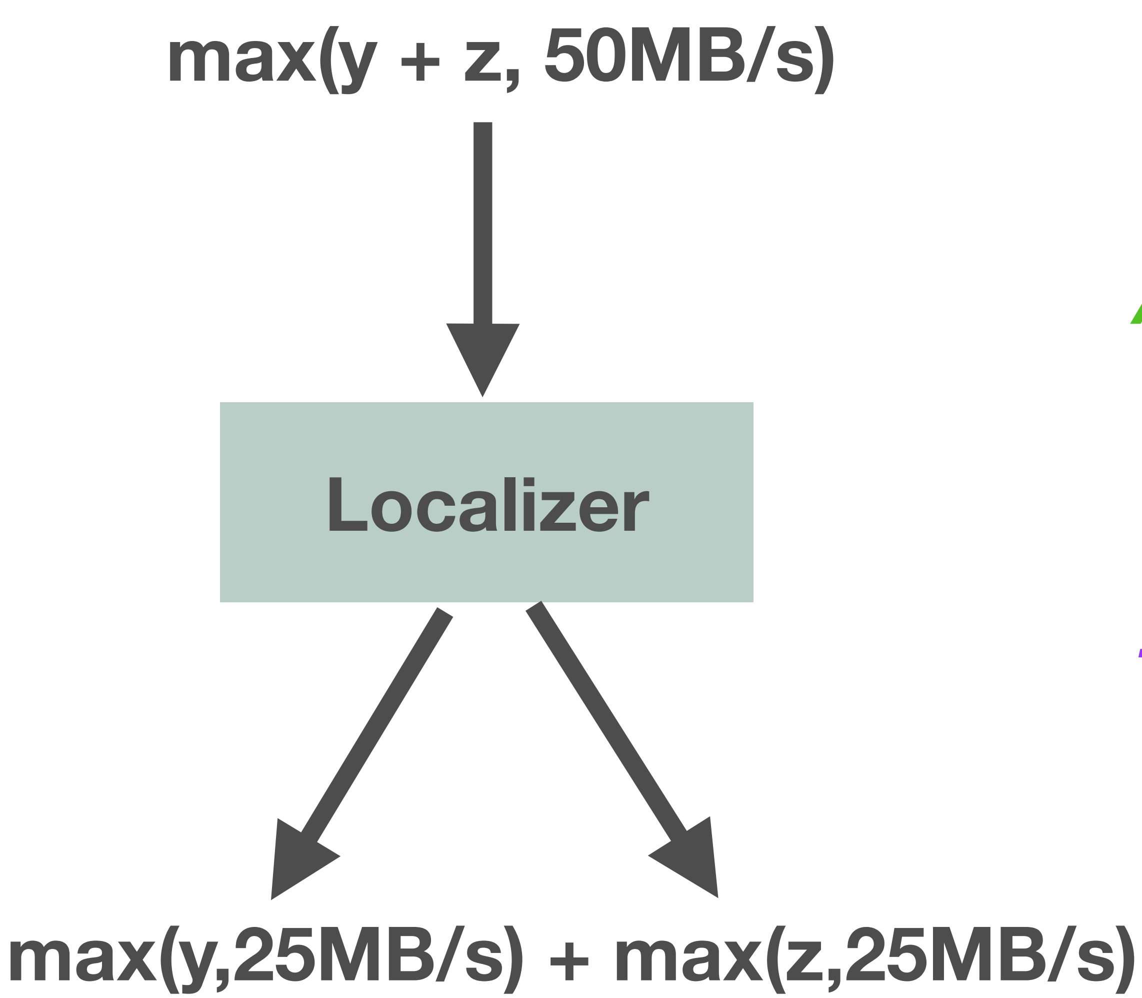
} *Set operators and iterators*

Merlin can concisely express a range of network policies. More examples in HotNets '13.

Compiler

Localize policies, allocate resources, and generate target code

Localization



Challenge:

Enforcing aggregate caps requires distributed state

Approach

- Re-write formulas so they can be locally enforced

Trade-off

- Increase scalability
- Risk underutilizing resource
- Run-time allows for dynamic adjustments

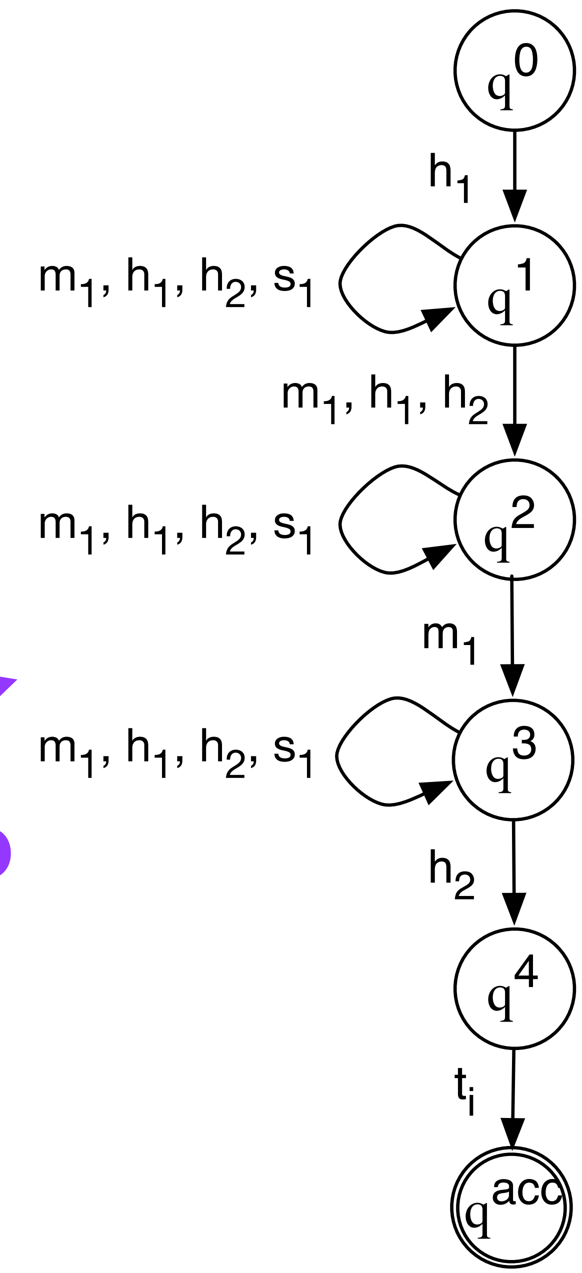
Allocate Resource: Map Policy to Constraints

```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Allocate Resource: Map Policy to Constraints

```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Convert to DFA

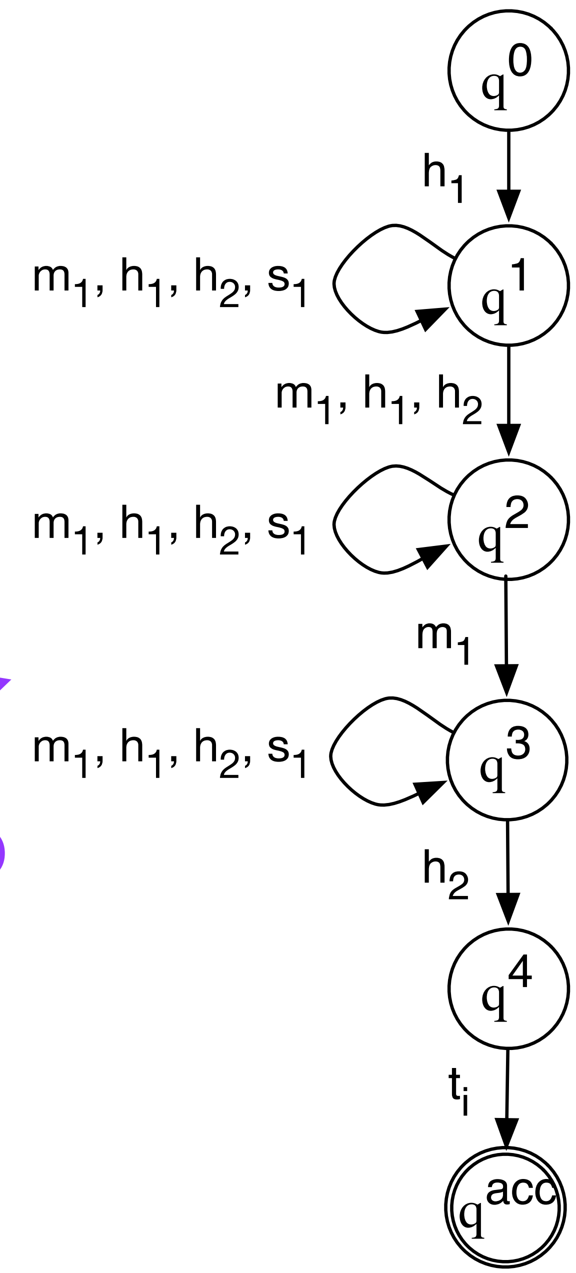


Allocate Resource: Map Policy to Constraints

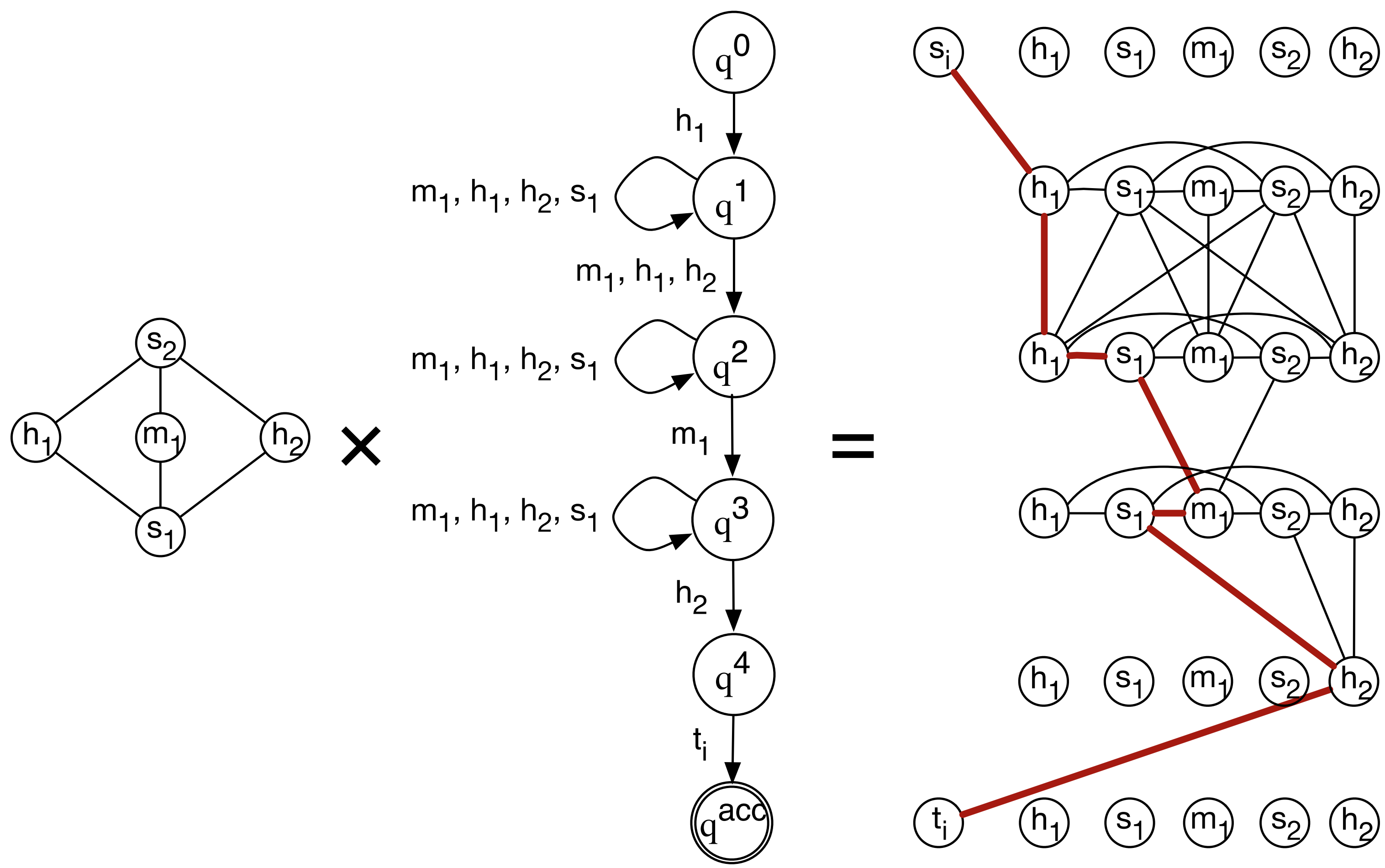
```
[ x :
    (eth.src = 00:00:00:00:00:01 &
     eth.dst = 00:00:00:00:00:02 &
     tcp.dst = 80)
     -> * nat *. dpi .*
], min(x,100MB/s)
```

Convert to DFA

Note resource demands



Solve MIP to Determine Paths and Placement



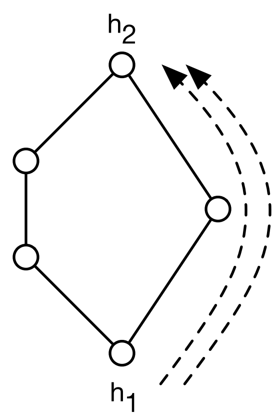
Physical topology with vertices V

Statement NFA with states Q_i

LP Graph G_i

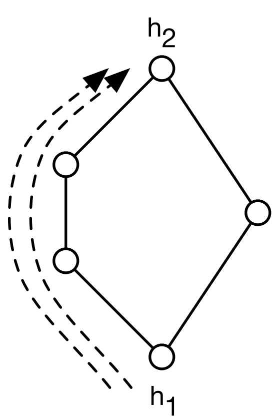
Encode with flow conservation and capacity constraints

Path Heuristics



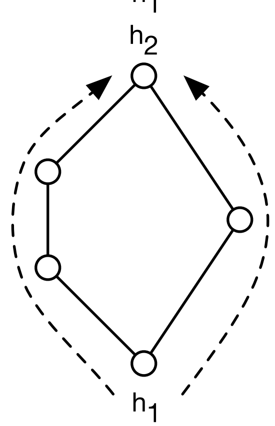
Weighted Shortest Path:

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 for routing Install rules on **OpenFlow** switches
	**Middleboxes**	Translate function to **Click** [TOCS’00] Install on software middleboxes
	**End Hosts**	Generate code for Linux **tc** and **iptables** Experimental support for Merlin kernel module based on **netfilter**

Dynamic Adaptation

Enable policy delegation and verify refined policies

Runtime Component

- *Negotiators* are runtime component for dynamic adaptation
- Distributed hierarchically throughout network
- Exchange messages amongst themselves to:
    - Modify (i.e., refine) policies
    - Verify policy modifications

Refine Policies

Informally: **Ensure that traffic between two hosts has a bandwidth cap of 100MB/s.**

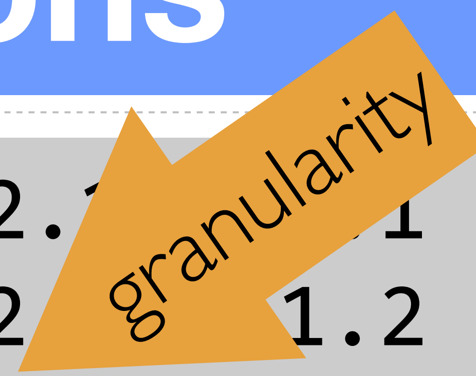
```
[x : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2) -> .*],
 max(x, 100MB/s)
```

Three Possible Transformations

Three Possible Transformations

```
[x : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 22) -> .* ],
[y : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 80) -> .* log .* ],
[z : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      !(tcpDst=22|tcpDst=80)) -> .* dpi .*],
max(x, 50MB/s)
and max(y, 25MB/s)
and max(z, 25MB/s)
```

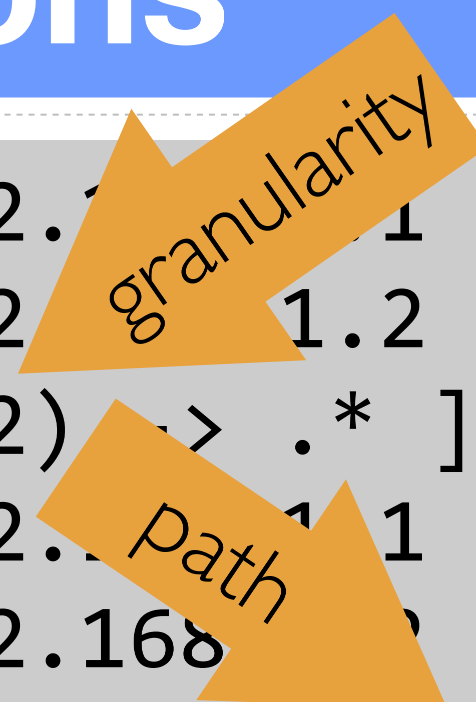
Three Possible Transformations




```
[x : (ip.src = 192.1[illegible]1 and
      ip.dst = 192[illegible]1.2 and
      tcp.dst = 22) -> .* ],
[y : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 80) -> .* log .* ],
[z : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      !(tcpDst=22|tcpDst=80)) -> .* dpi .*],
max(x, 50MB/s)
and max(y, 25MB/s)
and max(z, 25MB/s)
```

Three Possible Transformations

```
[x : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 22) -> .* ],
[y : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 80) -> .* log .* ],
[z : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      !(tcpDst=22|tcpDst=80)) -> .* dpi .*],
max(x, 50MB/s)
and max(y, 25MB/s)
and max(z, 25MB/s)
```



Three Possible Transformations

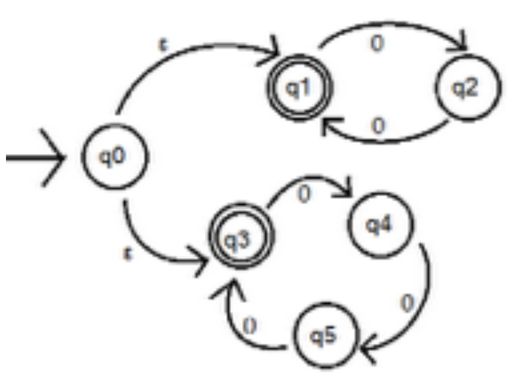
```
[x : (ip.src = 192.1 1 and
      ip.dst = 192 1.2 and
      tcp.dst = 22) -> .* ],
[y : (ip.src = 192. 1 and
      ip.dst = 192.168 and
      tcp.dst = 80) -> .* log .* ],
[z : (ip.src = 192.168.1.1 and
      ip.dst = 192.1 .2 and
      !(tcpDst=22 st=80)) -> .* dpi .*],
max(x, 50MB/s)
and max(y, 25MB/s)
and max(z, 25MB/s)
```

granularity

path

allocation

Automatic Verification



Essential operation:

policy inclusion (i.e., $P_1 \subseteq P_2$)

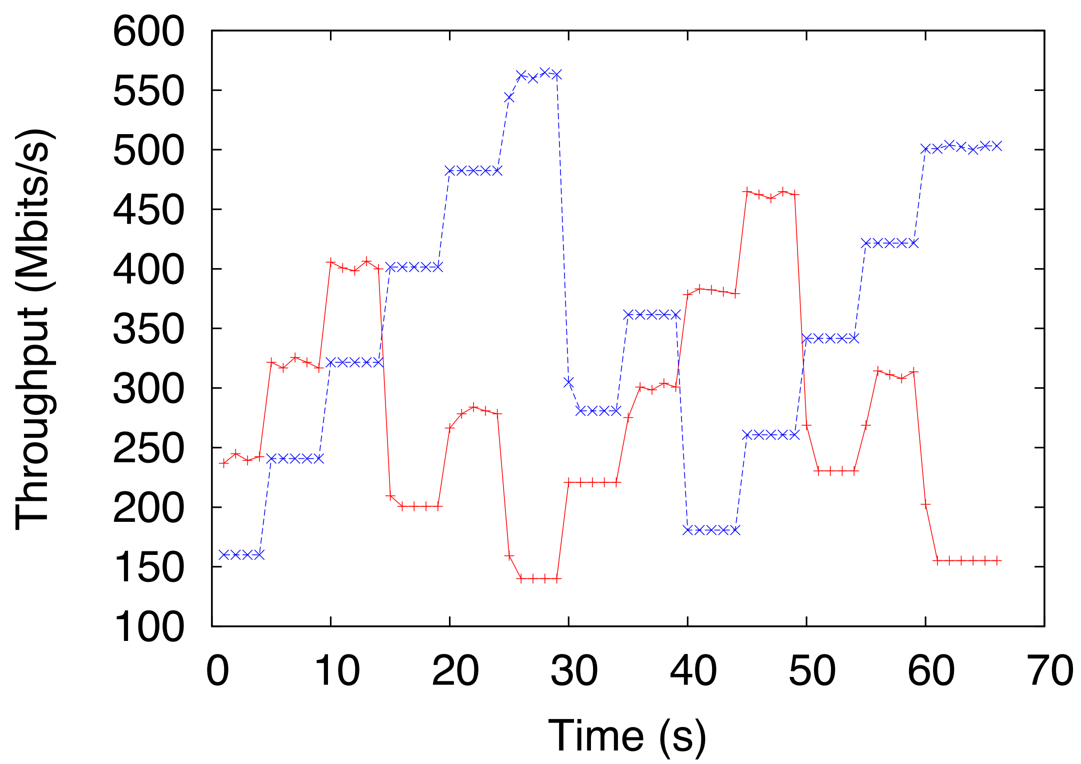
Algorithm

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

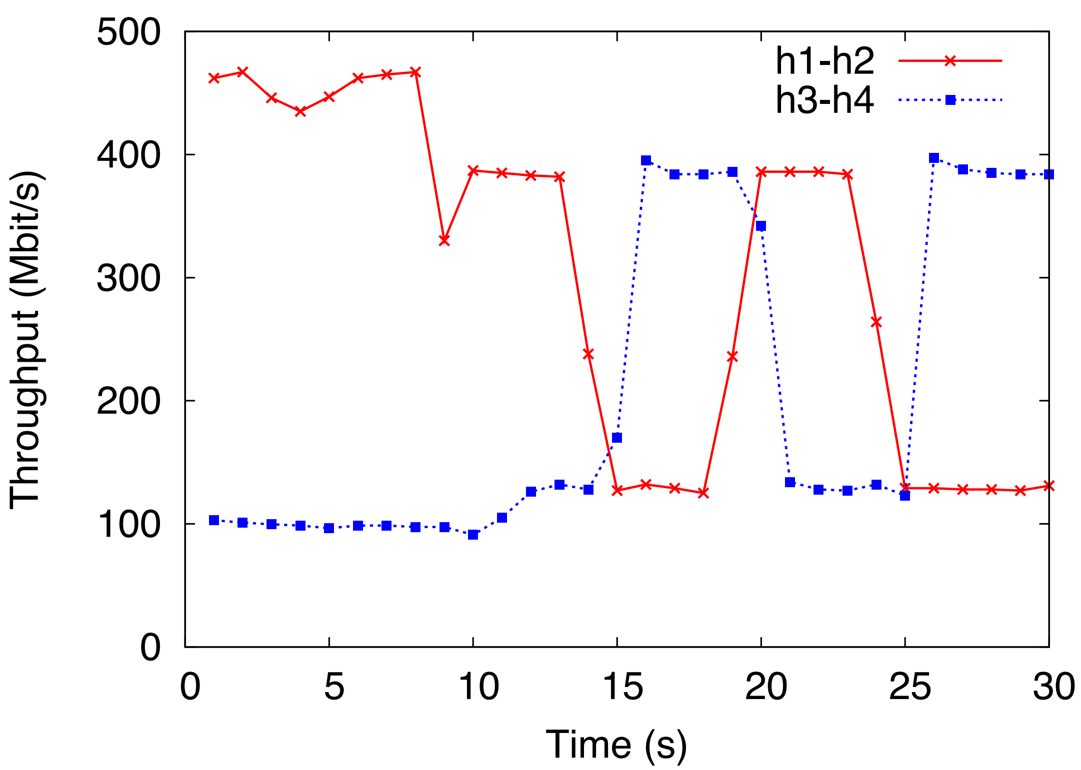
Implementation

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

Negotiator Implementations



Additive-Increase, Multiplicative-Decrease



Max-Min Fair Sharing

Evaluation

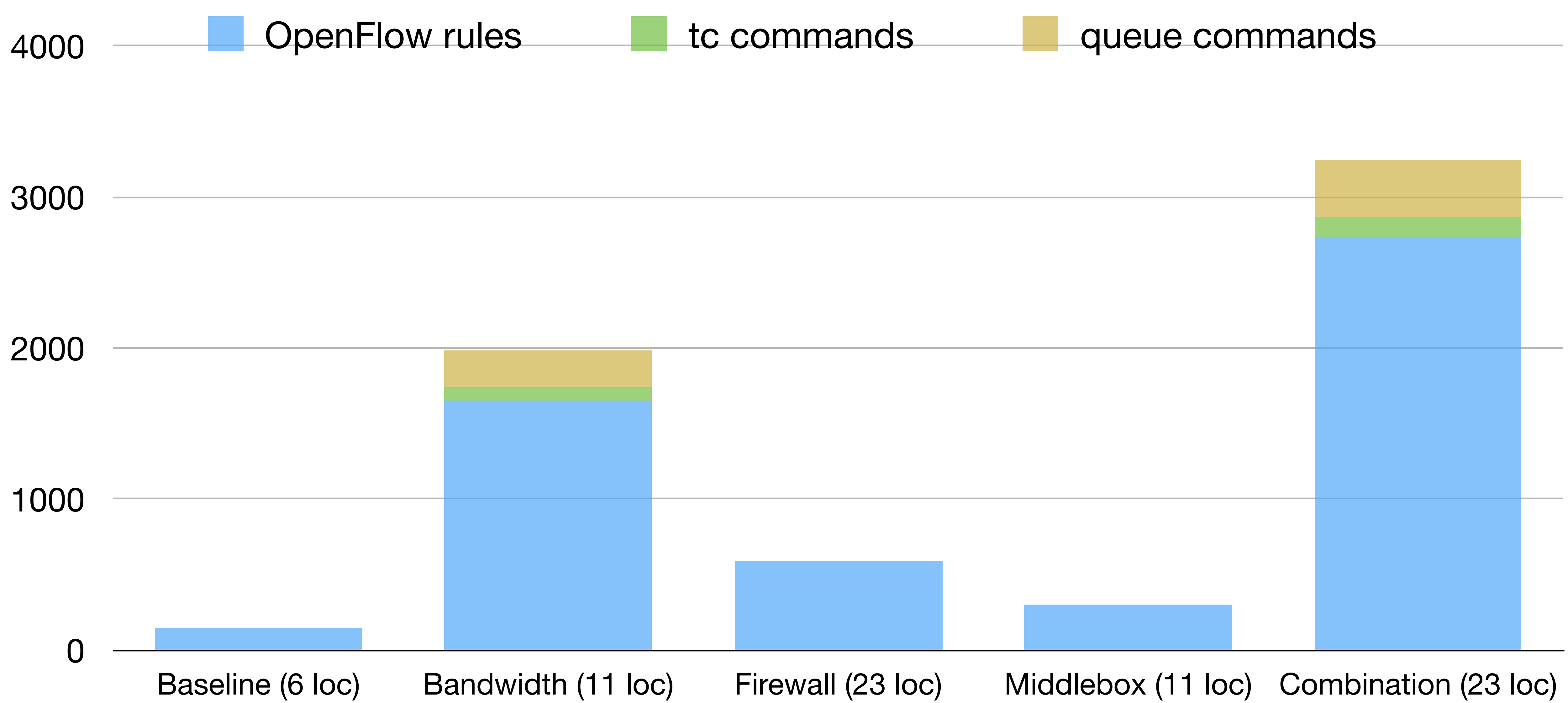
Demonstrating Merlin's expressiveness,
ability to manage the network, and scalability

Merlin Network Management

Baseline	Basic all-pairs connectivity between hosts
Bandwidth	10% of traffic classes get a guarantee of 1Mbps, and a cap of 1Gbps
Firewall	All packets with tcp.dst = 80 are routed through a firewall
Middlebox	Hosts are partitioned into two sets (trusted and untrusted). Inter-set traffic must pass through a middle box.
Combination	All of the above

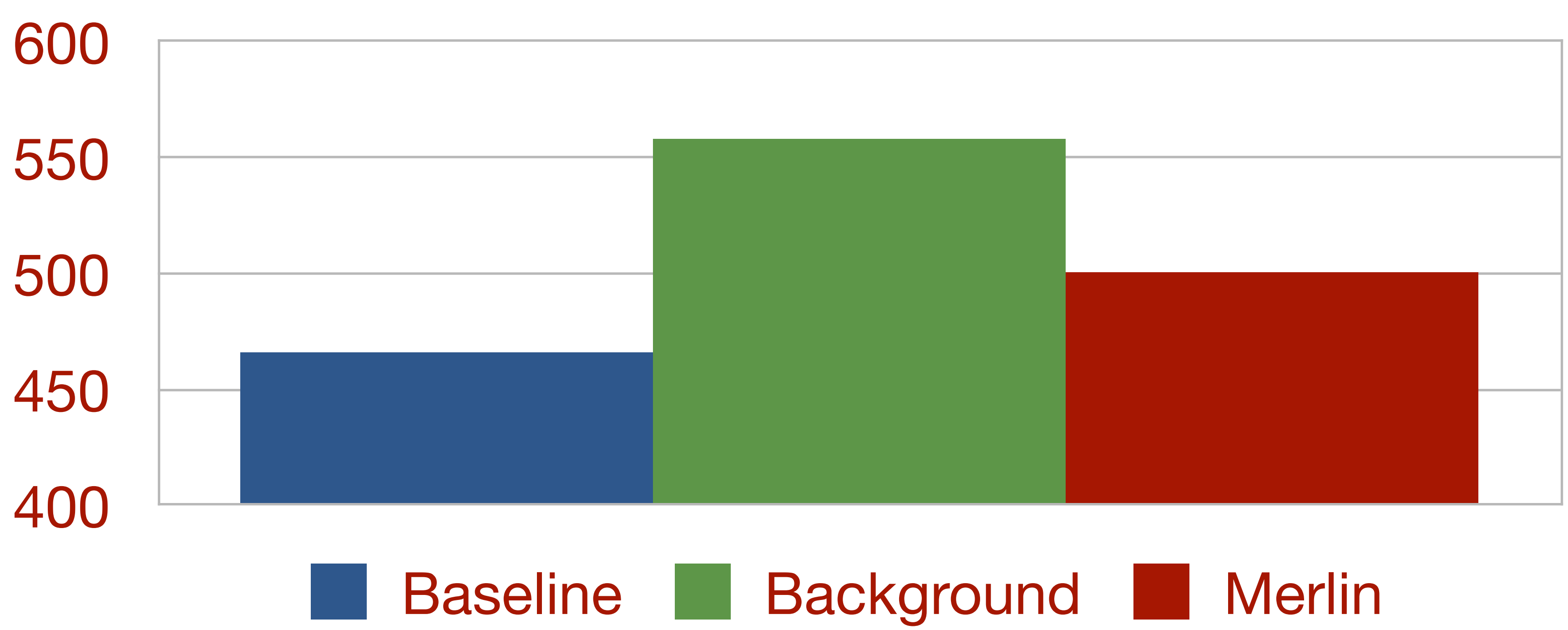
Policies to manage Stanford network topology

Merlin Reduces Management Effort



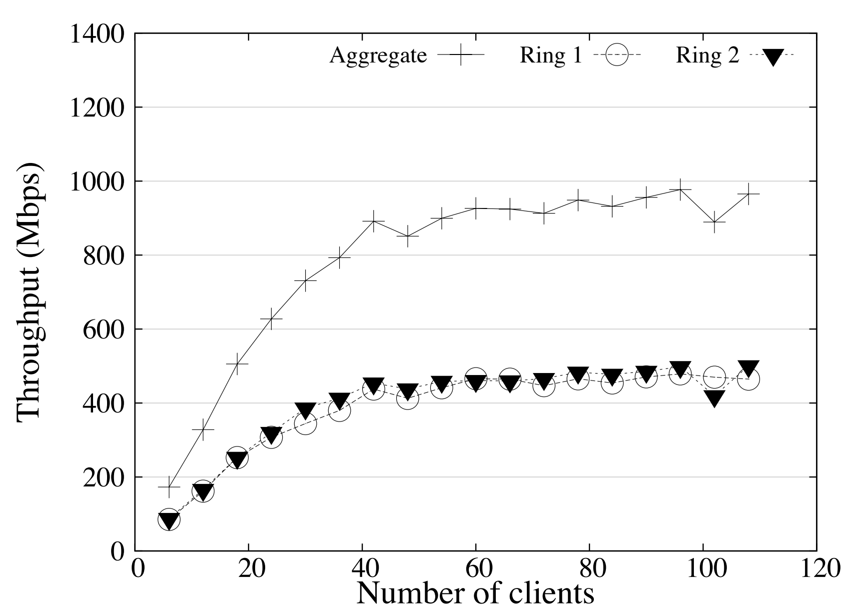
Merlin Managing Hadoop

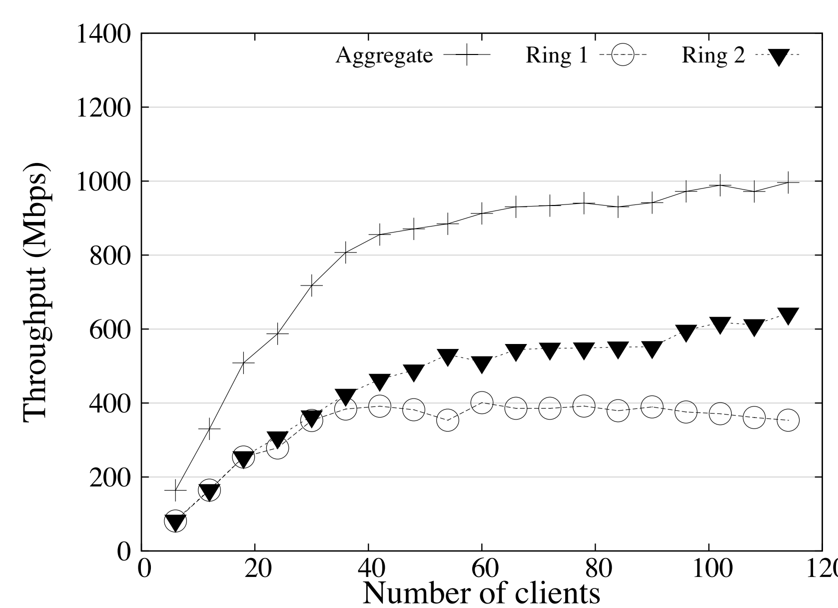
- Measured completion time for word count:
    1. Without background traffic
    2. With background traffic
    3. With background traffic + Merlin reserve 90% capacity



Merlin Managing Ring Paxos

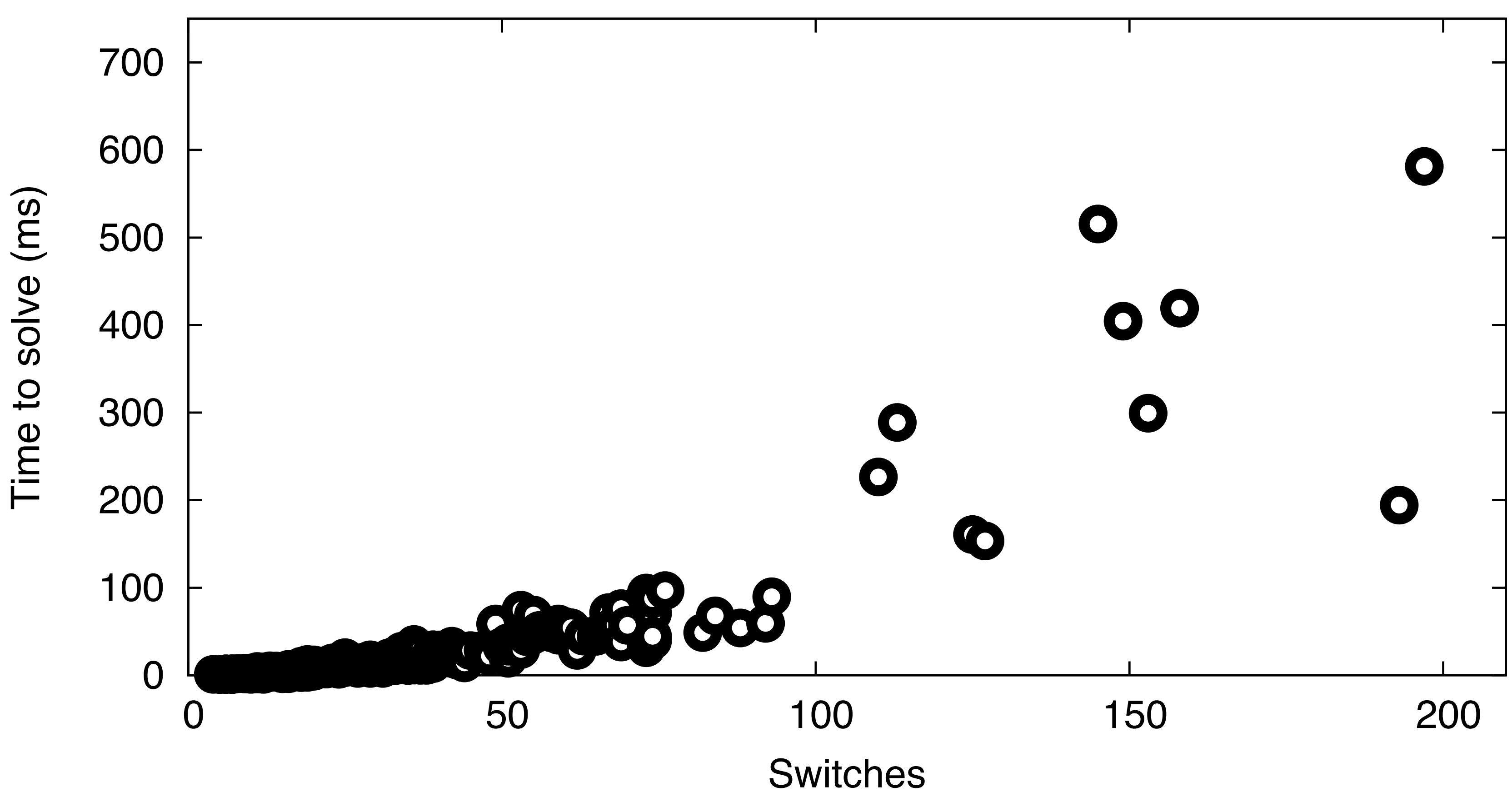
- State machine replication (SMR) is fundamental approach for fault-tolerant services
- Measure throughput for co-located key-value store service backed by SMR
- Merlin prioritizes traffic for one service





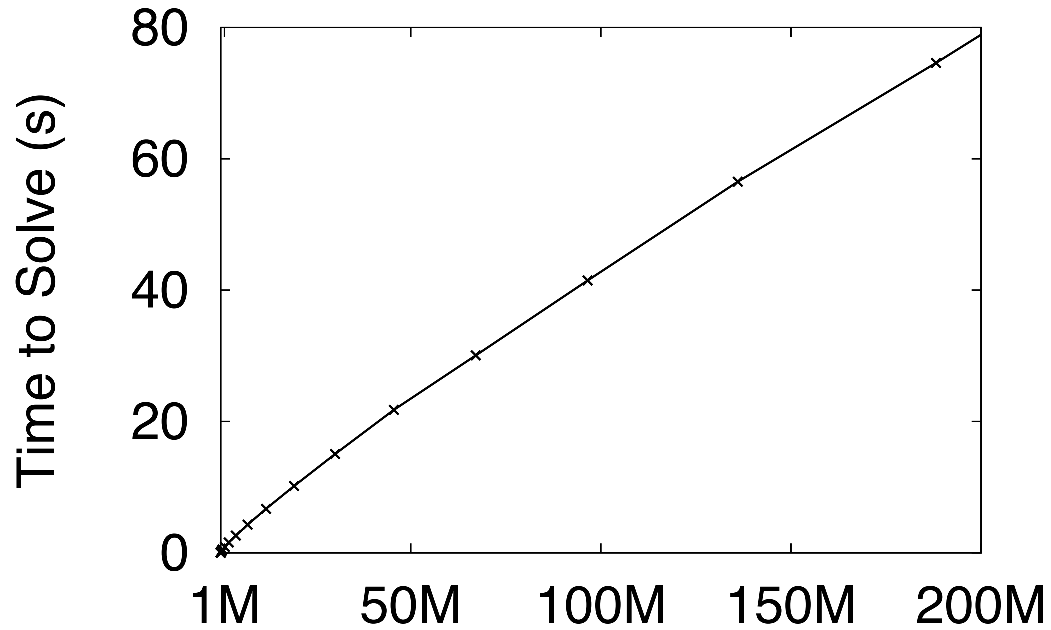
Compilation is Fast For Basic Connectivity

- All-pairs connectivity for Internet Topology Zoo dataset
- Majority of topologies completed in <50ms

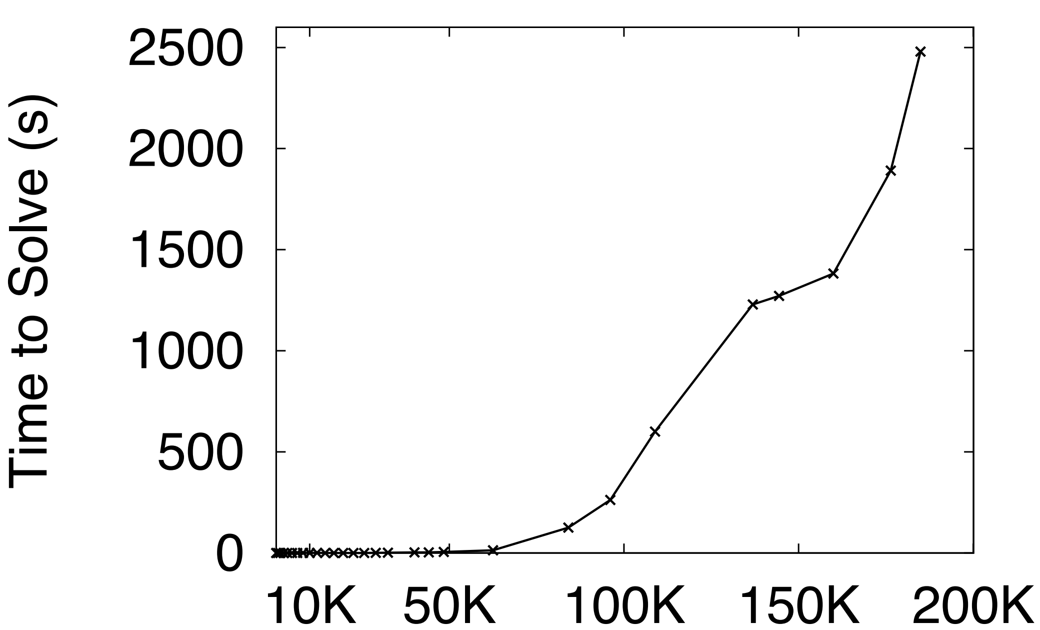


Solver Adds Reasonable Overhead

- Measured compilation time for fat tree topologies for an increasing number of traffic classes
- 100 traffic classes for 125 switch network in 5 sec



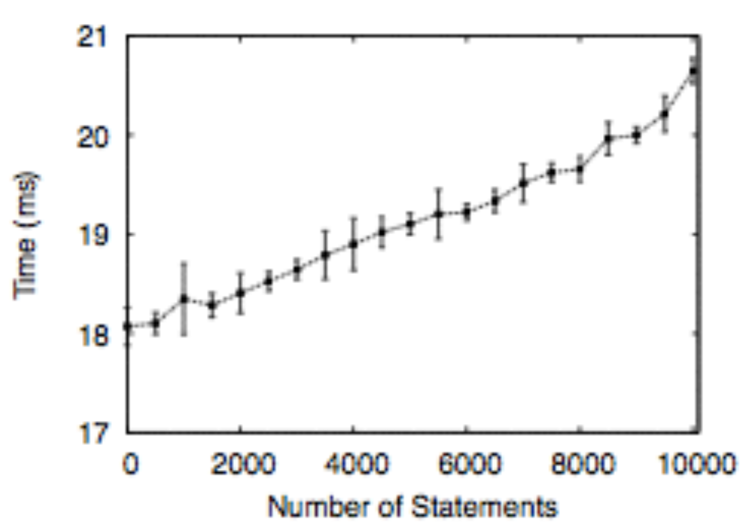
All-pairs connectivity



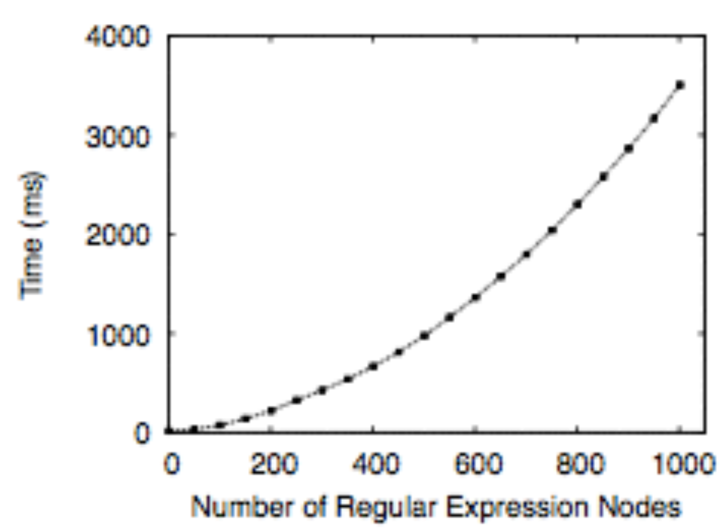
5% of traffic with bandwidth guarantees

Verification is Very Fast

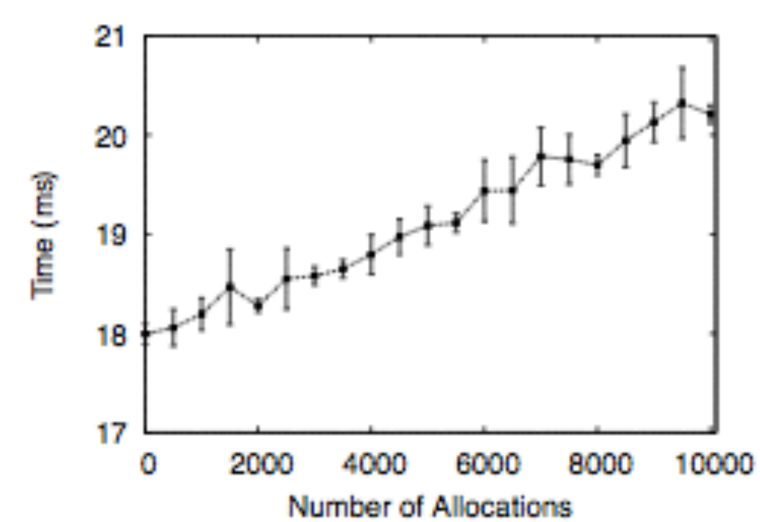
- 10,000 statements verified in less than 21ms
- Verifying resource allocations is very fast
- Verifying paths scales with complexity of the expression



Increasing Statements



Increasing Path Expressions



Increasing Bandwidth Constraints

Conclusion

- Merlin dramatically simplifies network management
- It provides abstractions that:
  - Let developers program the network as a unified entity
  - Allow mapping to a constraint problem for provisioning
  - Enable delegation and automatic verification

http://frenetic-lang.org/merlin