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

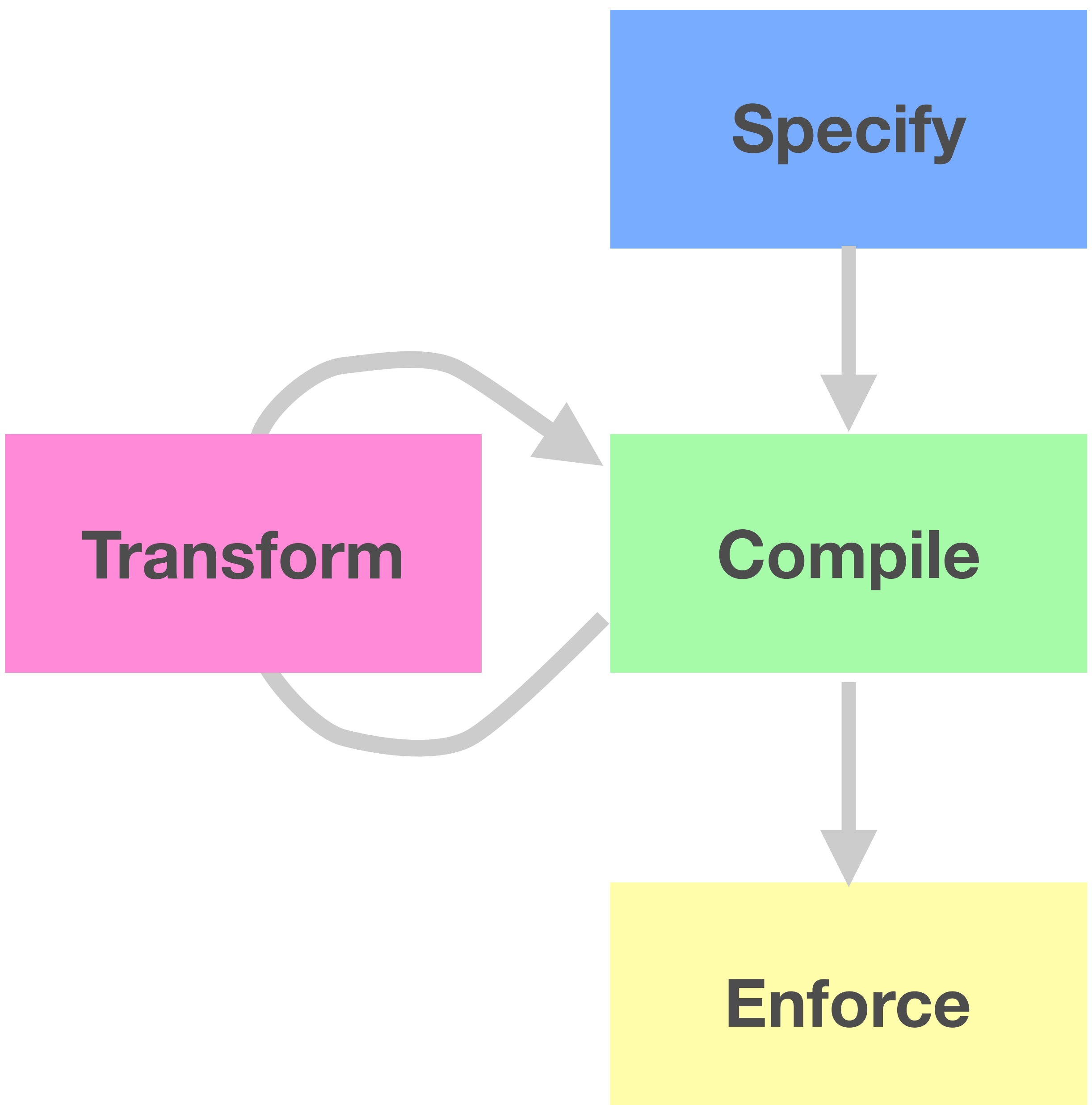
SDN Languages are Limited

- SDNs have simplified network management and increased programability
- But, existing SDN languages focus mostly on packet forwarding
- Network orchestration frameworks expose extremely simple APIs (if at all)

Need More Than Forwarding

- Support traffic engineering goals through bandwidth caps and guarantees
- Apply packet-processing functions such as NAT, DPI, load-balancers, etc.
- Provide an intuitive programming interface with compose-able policies

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

Basic Policy

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.**

Basic Policy

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 :
    (ip.src = 192.168.1.1 and
     ip.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Basic Policy

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
    (ip.src = 192.168.1.1 and
     ip.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Basic Policy

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 :
    (ip.src = 192.168.1.1 and
     ip.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Identifier

Predicates identify which traffic

Basic Policy

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 :
    (ip.src = 192.168.1.1 and
     ip.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Identifier

Predicates identify which traffic

Regular expressions for paths, functions

Basic Policy

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 :
    (ip.src = 192.168.1.1 and
     ip.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Identifier

Predicates identify which traffic

Regular expressions for paths, functions

Caps or guarantees for bandwidth

Aggregate Policy

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

Aggregate Policy

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

```
[ y : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 20) -> .* dpi .*
  z : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

Aggregate Policy

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

```
[ y : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 20) -> .* dpi .*
  z : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

} *FTP data*

Aggregate Policy

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

```
[ y : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 20) -> .* dpi .*
  z : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 21) -> .*
],
max(y + z,50MB/s)
```

} *FTP data*

} *FTP control*

Aggregate Policy

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

```
[ y : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 and
       tcp.dst = 20) -> .* dpi .*
  z : (ip.src = 192.168.1.1 and
       ip.dst = 192.168.1.2 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 := {192.168.1.1}
dsts := {192.168.1.2}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at min(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 := {192.168.1.1}
dsts := {192.168.1.2}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at min(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 := {192.168.1.1}
dsts := {192.168.1.2}
foreach (s,d) in cross(srcs,dsts):
   tcp.dst = 80 ->
   ( .* nat .* dpi .*) at min(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 := {192.168.1.1}
dsts := {192.168.1.2}
foreach (s,d) in cross(srcs,dsts):
    tcp.dst = 80 ->
    ( .* nat .* dpi .*) at min(100MB/s)
```

} ***Set literals***

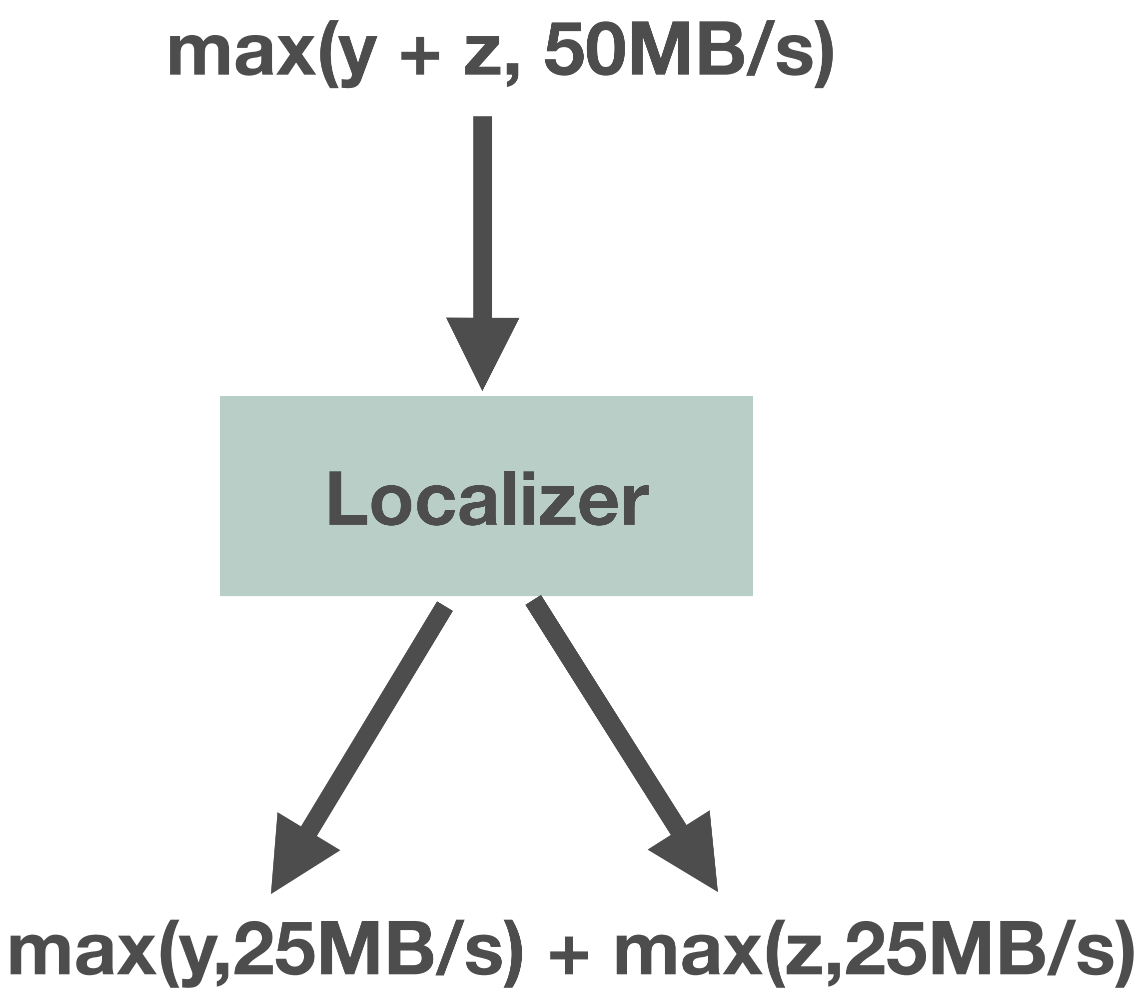
} ***Set operators and iterators***

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

Compiler

Localize policies, allocate resources, and generate target code

Remove Distributed State



- Enforcing aggregate caps requires distributed state (e.g., FTP control and data traffic)
- Compiler re-writes formulas so that they only require local state
- There is an inherent trade-off: increased scalability vs. risk of under-utilization

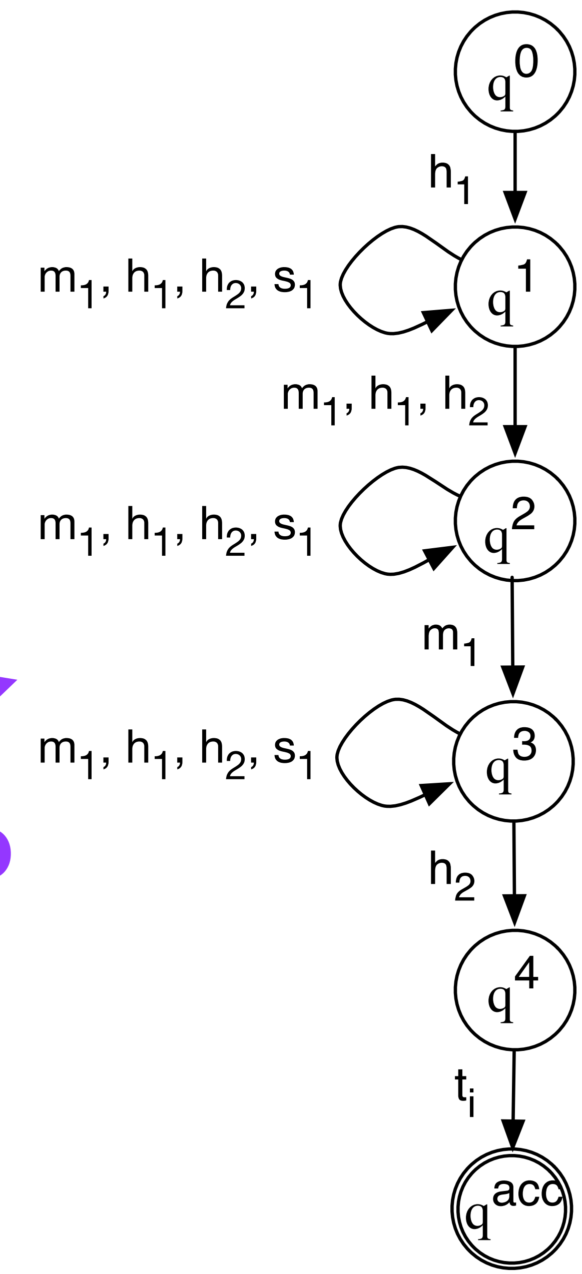
Extract Policy Constraints

```
[ x :
    (eth.src = 192.168.1.1 and
     eth.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat *. dpi .*
], min(x,100MB/s)
```

Extract Policy Constraints

```
[ x :
    (eth.src = 192.168.1.1 and
     eth.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Convert to DFA

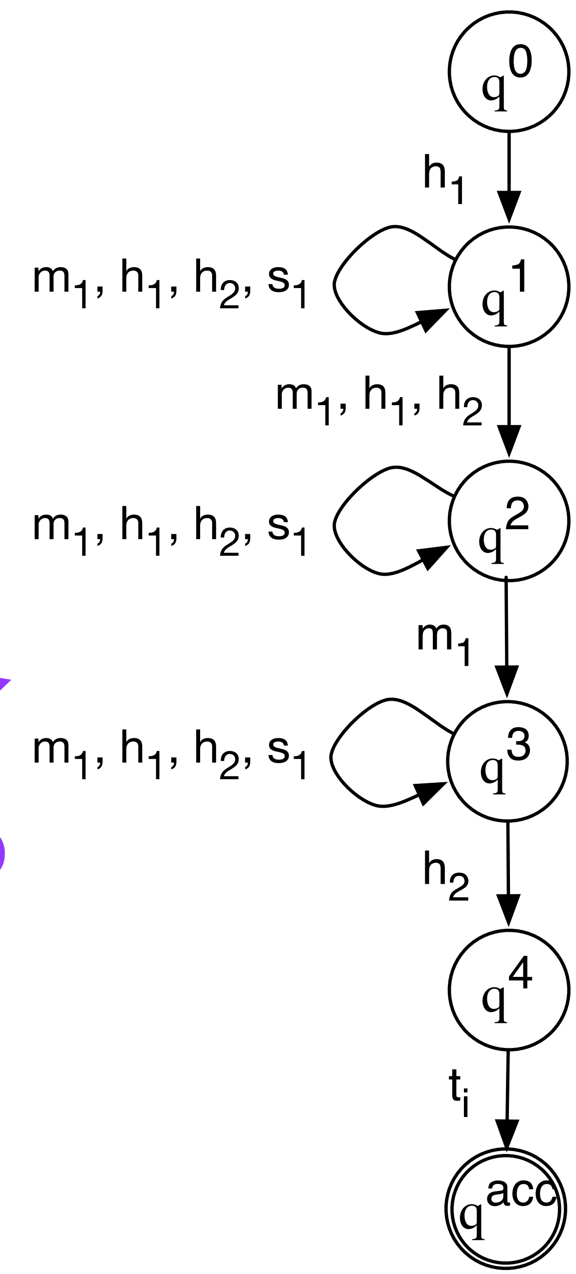


Extract Policy Constraints

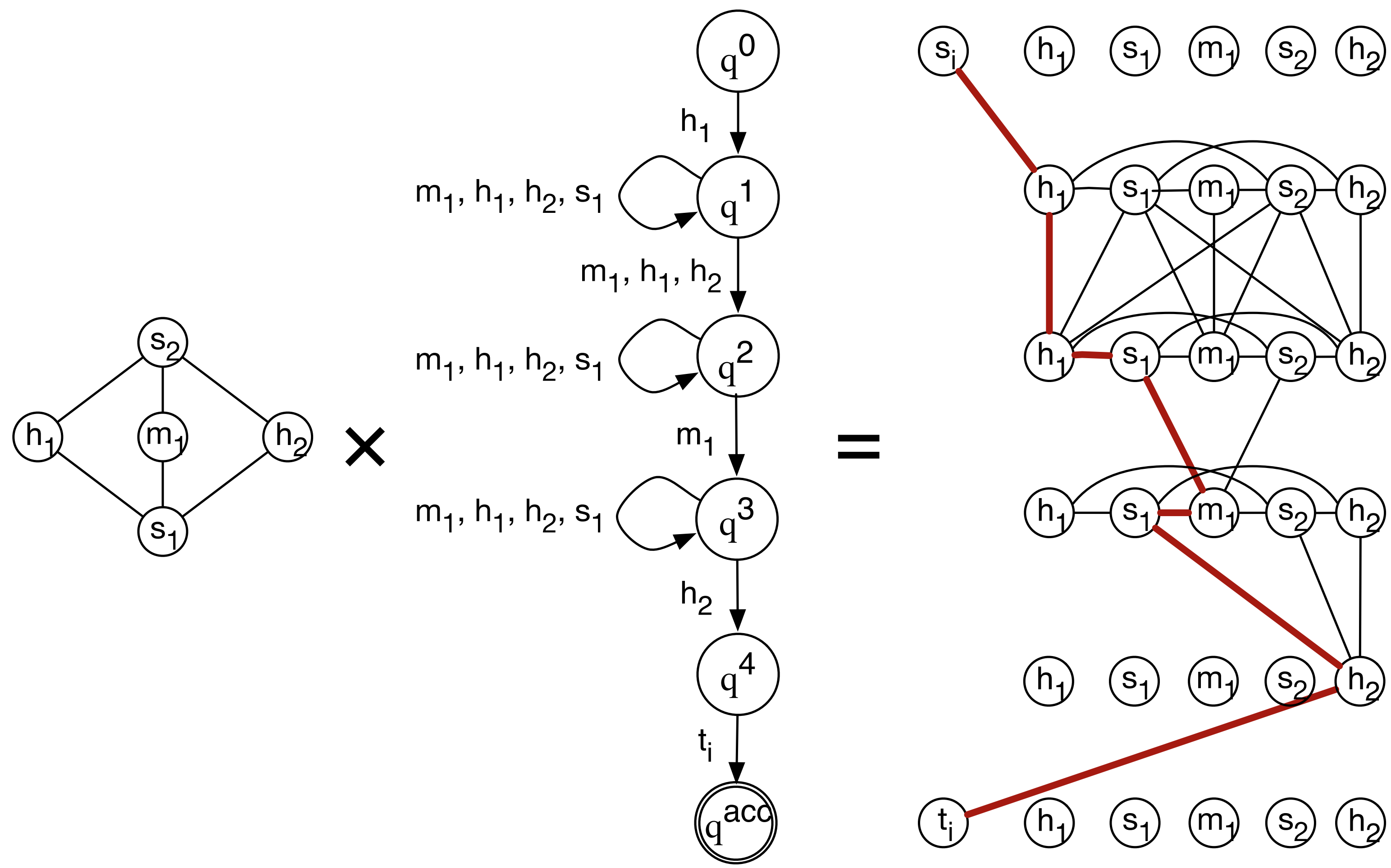
```
[ x :
    (eth.src = 192.168.1.1 and
     eth.dst = 192.168.1.2 and
     tcp.dst = 80)
     -> .* nat .* dpi .*
], min(x,100MB/s)
```

Convert to DFA

Note resource demands



Solve MIP For Paths and Placement



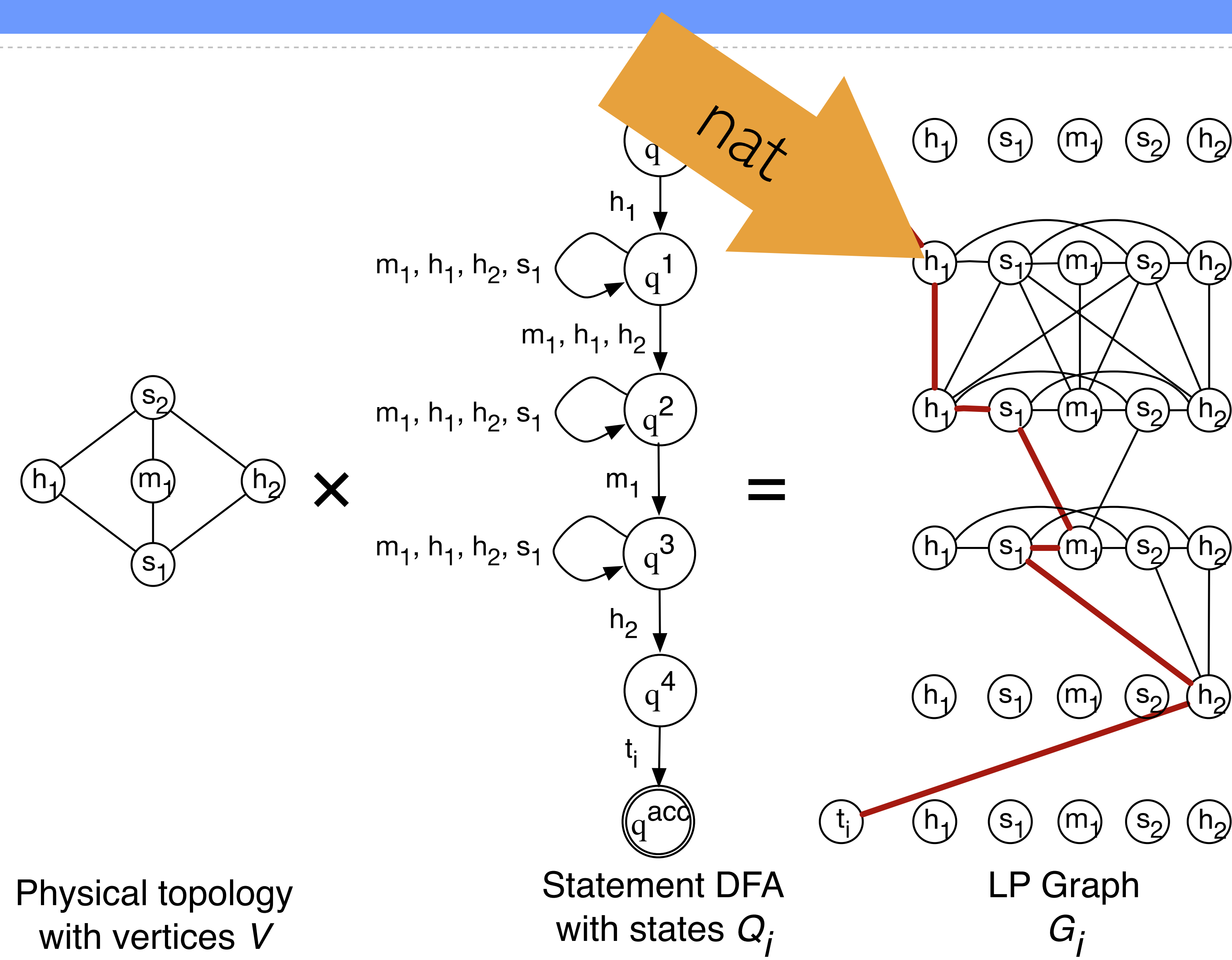
Physical topology with vertices V

Statement DFA with states Q_i

LP Graph G_i

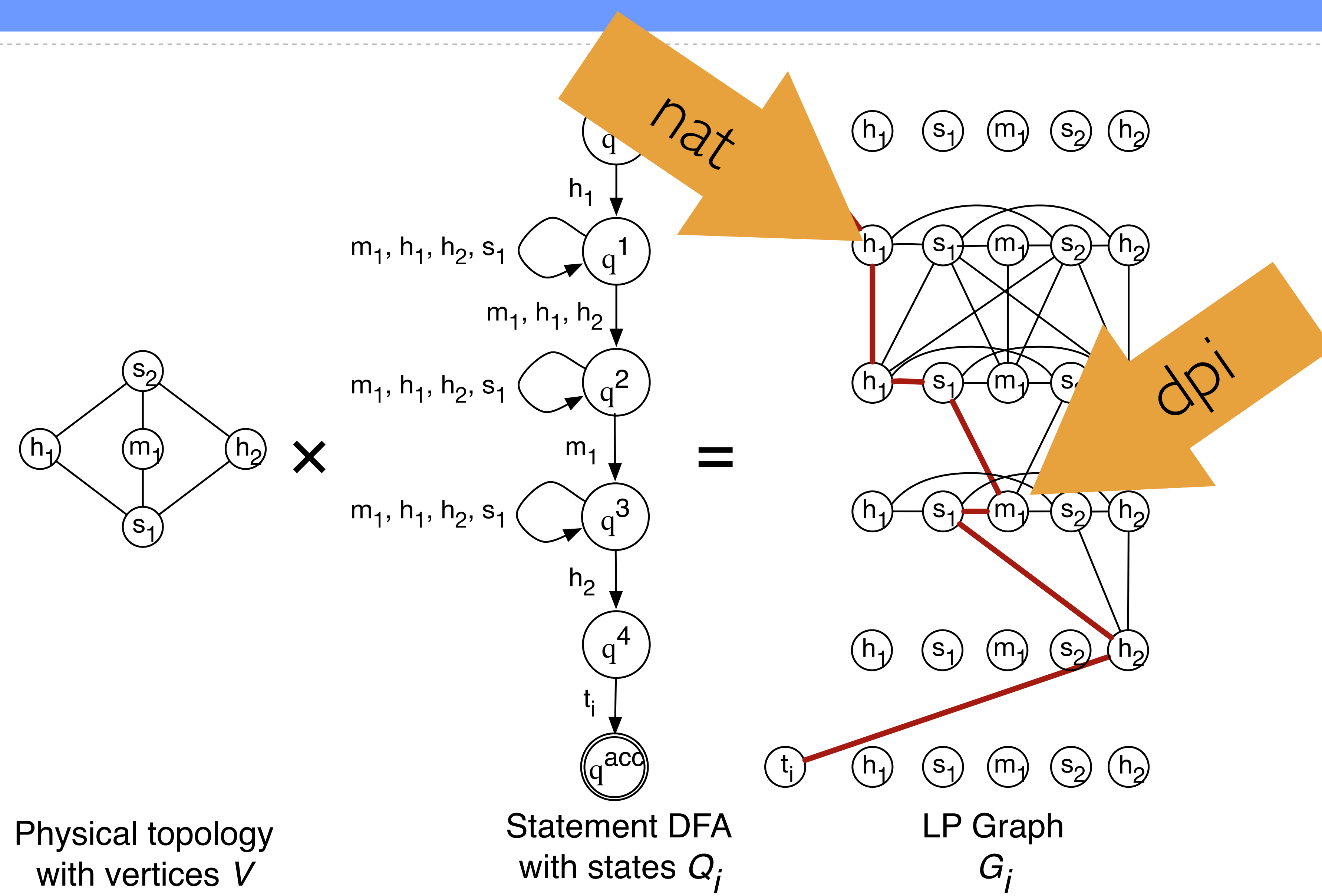
Encode with flow conservation and capacity constraints

Solve MIP For Paths and Placement



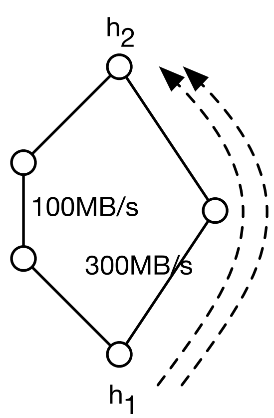
Encode with flow conservation and capacity constraints

Solve MIP For Paths and Placement



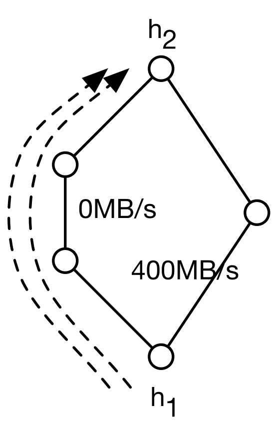
Encode with flow conservation and capacity constraints

Choose Path Heuristic



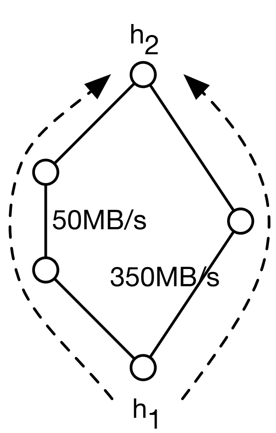
Weighted Shortest Path:

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)

Generate Code

	**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

Delegate 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)
```

Transform Policies

Transform Policies

```
[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)
```

Transform Policies

granularity

```
[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)
```

Transform Policies

```
[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)
```

granularity

path

Transform Policies

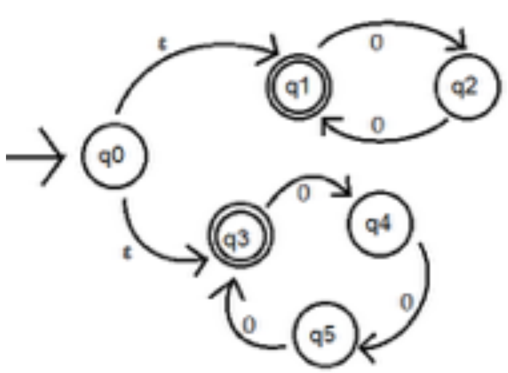
```
[x : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 22) -> .* ],
[y : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      tcp.dst = 80) -> .* log .* ],
[z : (ip.src = 192.168.1.1 and
      ip.dst = 192.168.1.2 and
      !(tcpDst=22 or tcpDst=80)) -> .* dpi .*],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

granularity

path

allocation

Verify Transformed Policies



Essential operation:

Ensure that new policy implies the old (i.e., $P_1 \subseteq P_2$)

Algorithm

- Perform pair-wise comparison of statements
- Check for path inclusion in overlaps
- Check aggregate bandwidth constraints

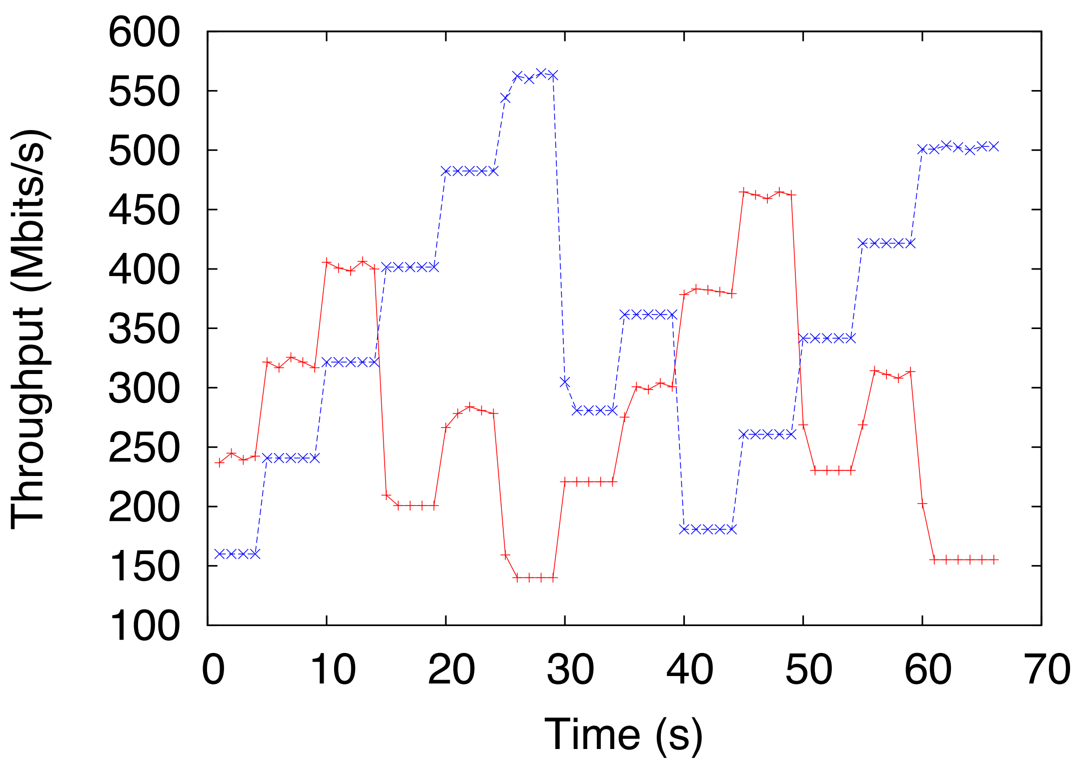
Implementation

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

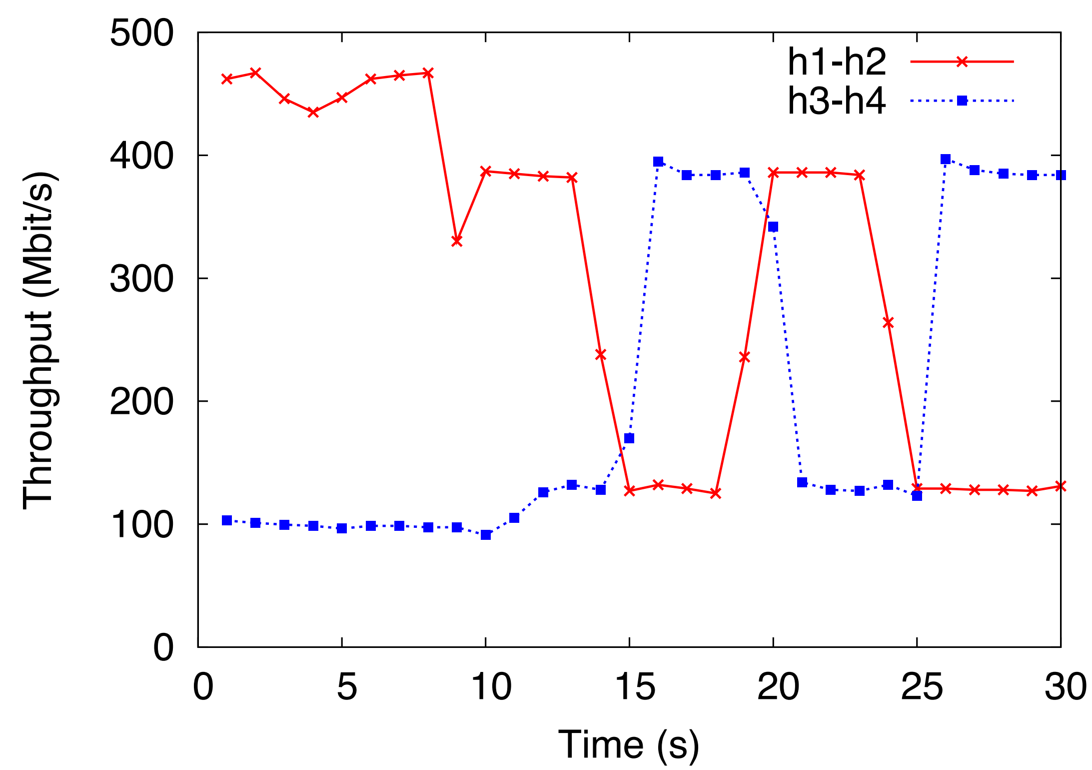
Adapt to Network Changes

- A small runtime component, called a *negotiator,* is distributed in a hierarchical overlay of the network
- Negotiators exchange messages amongst themselves to:
    - Modify (i.e., refine) policies
    - Verify policy modifications
- They can be instantiated with different adaptation schemes

Negotiator Implementations



Additive-Increase, Multiplicative-Decrease



Max-Min Fair Sharing

Evaluation

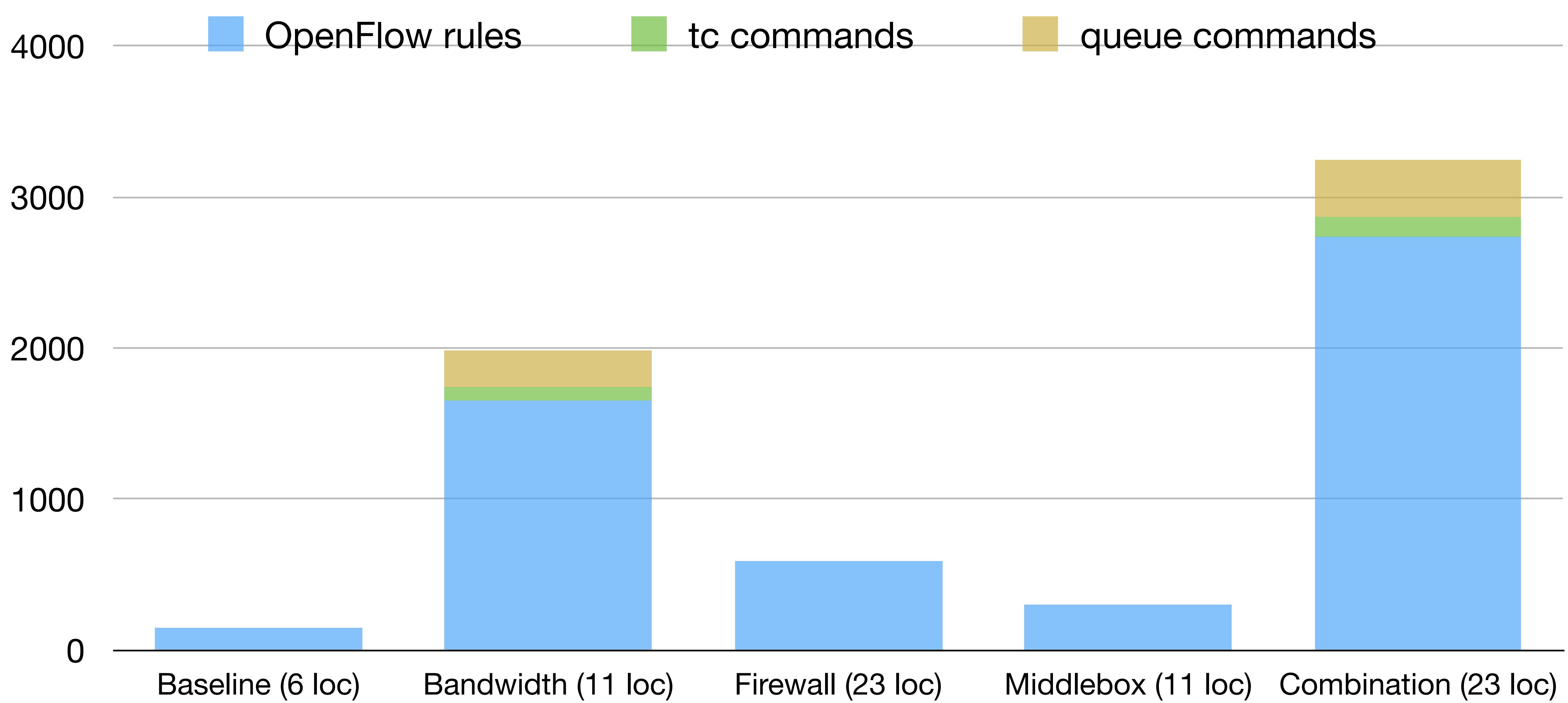
Demonstrating Merlin's expressiveness, ability to manage the network, and scalability

Example Network Policies

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

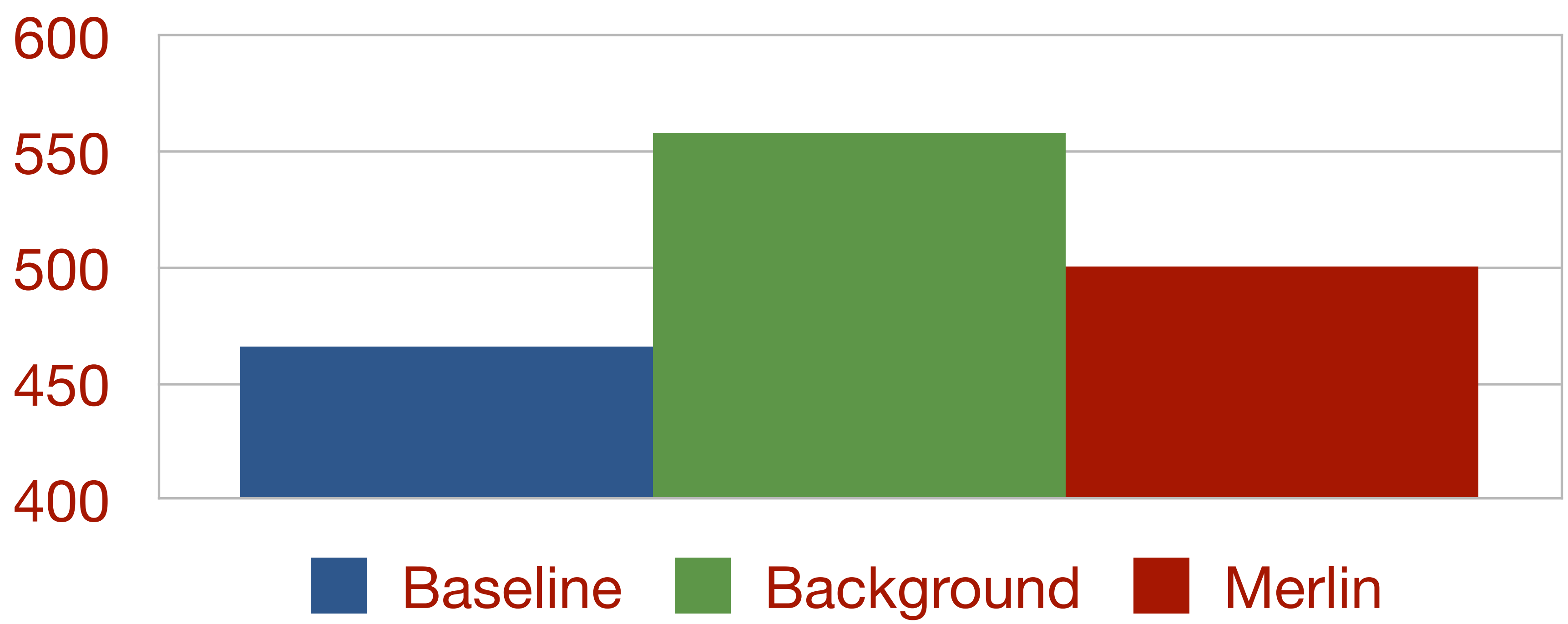
Policies to manage Stanford network topology

Merlin Is Expressive



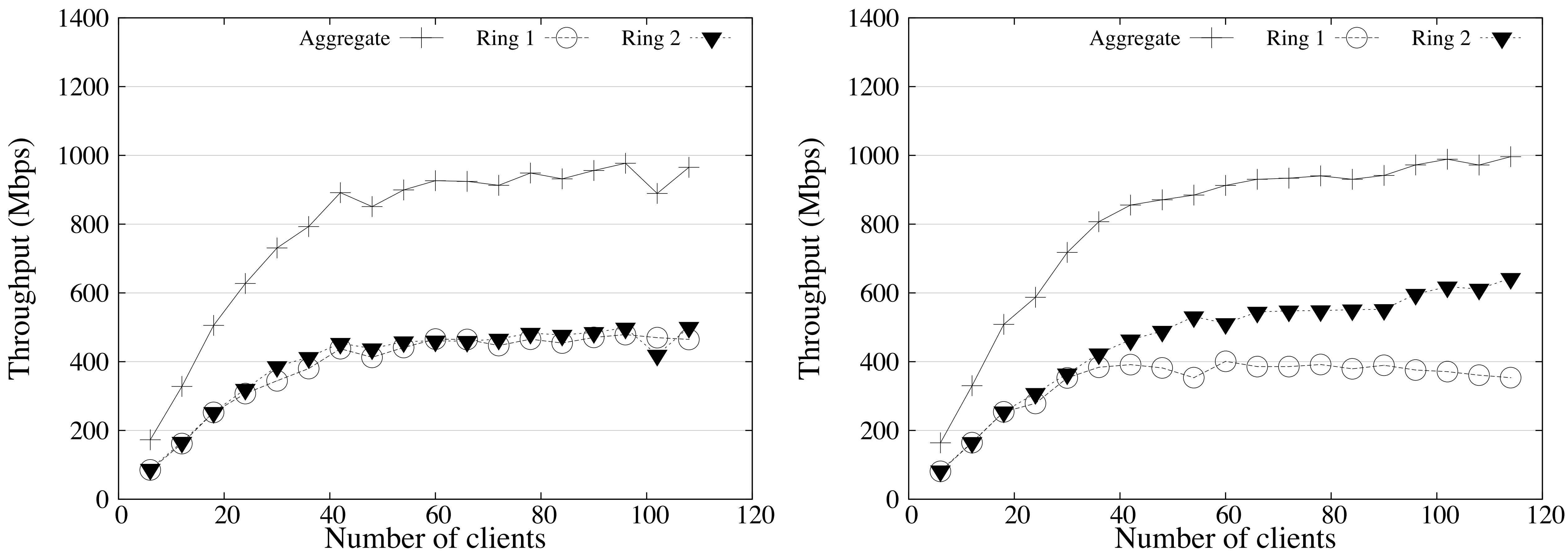
Merlin Managing Hadoop

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



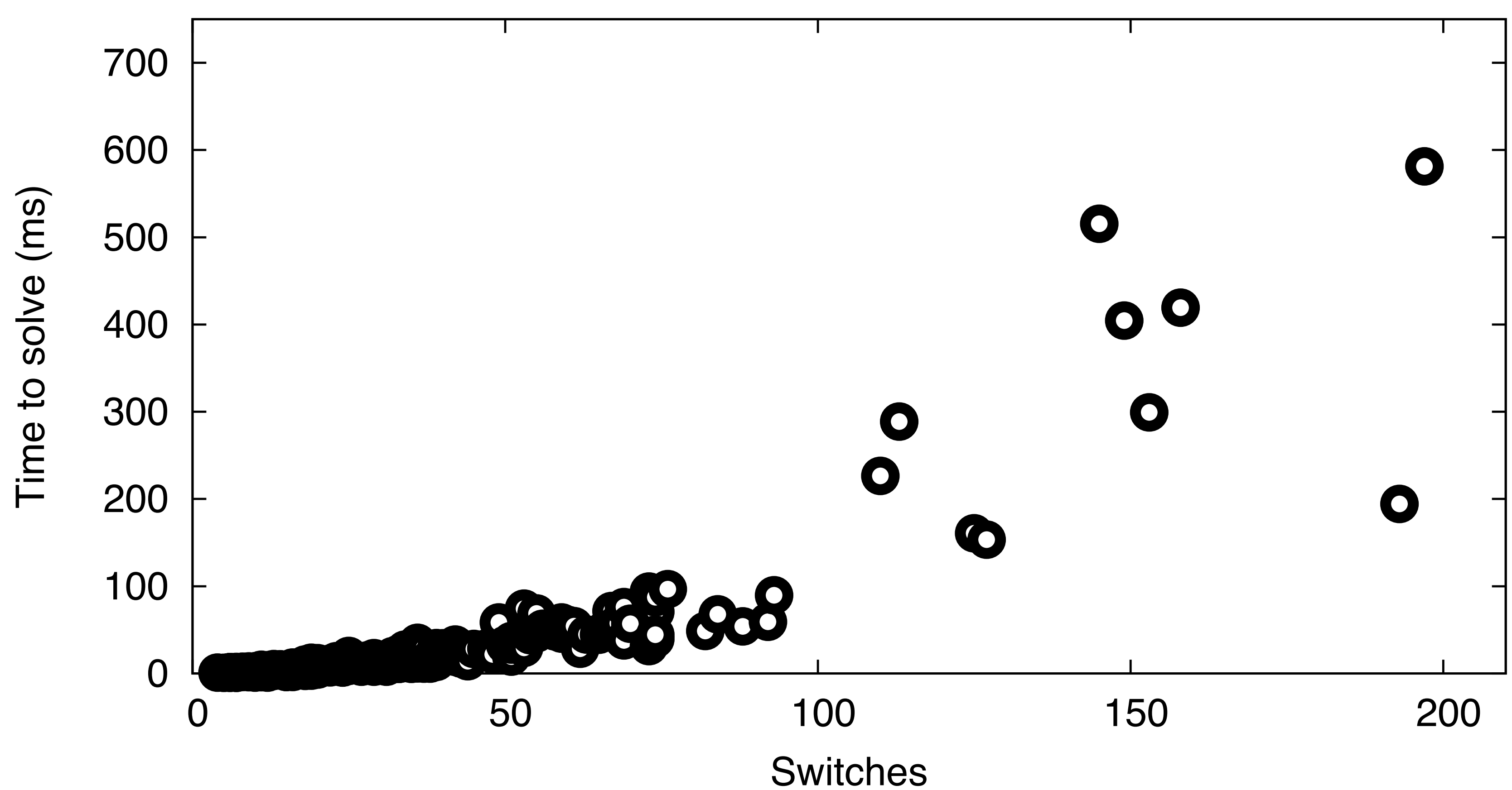
Merlin Managing Ring Paxos

- Measured throughput for co-located key-value stores backed by state machine replication
- Merlin prioritizes traffic for one service



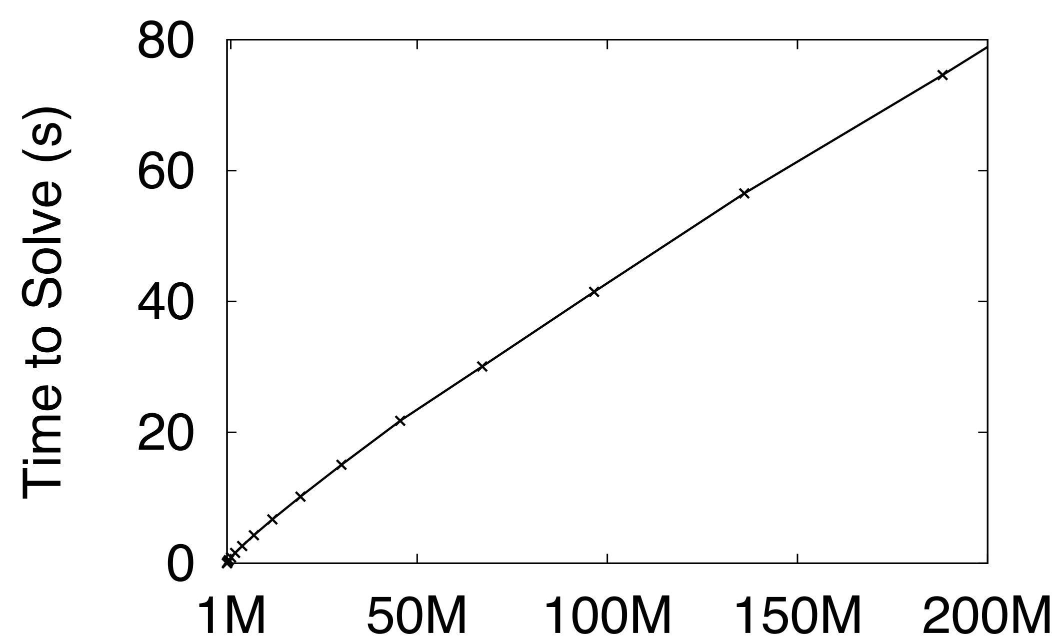
Compilation Is Fast For Basic Connectivity

- Measured compilation time for all-pairs connectivity on Internet Topology Zoo dataset
- Majority of topologies completed in <50ms

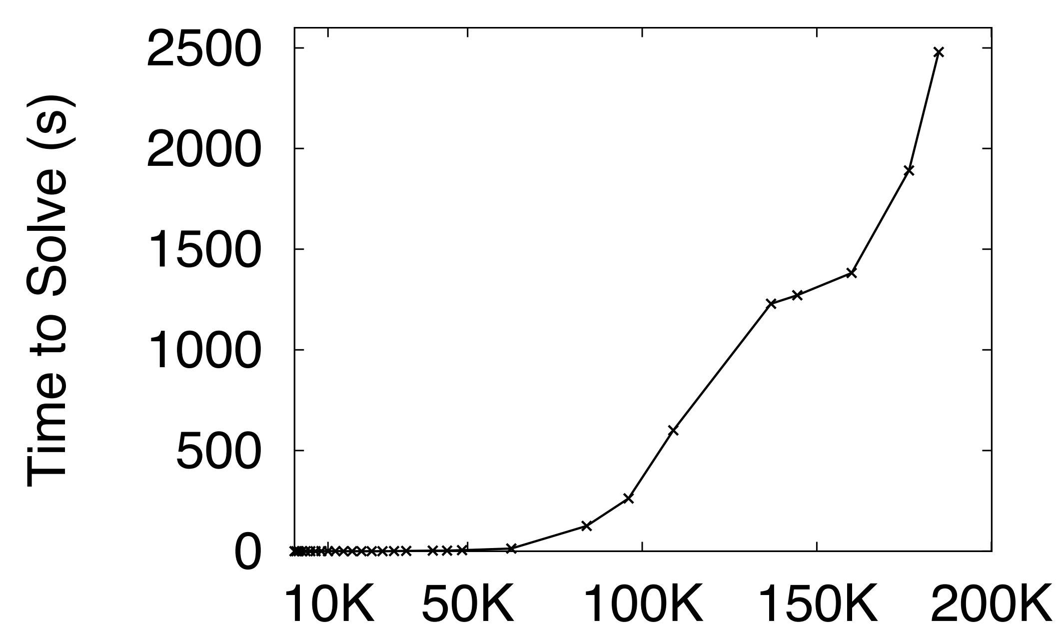


Solver Adds Reasonable Overhead

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



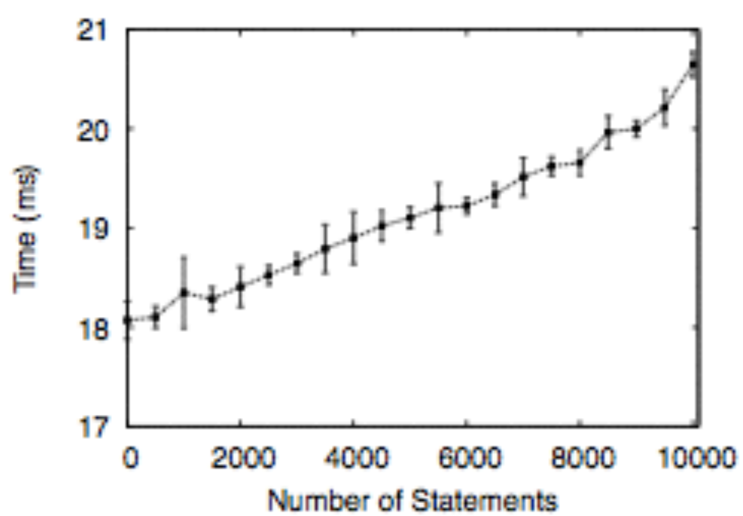
All-pairs connectivity



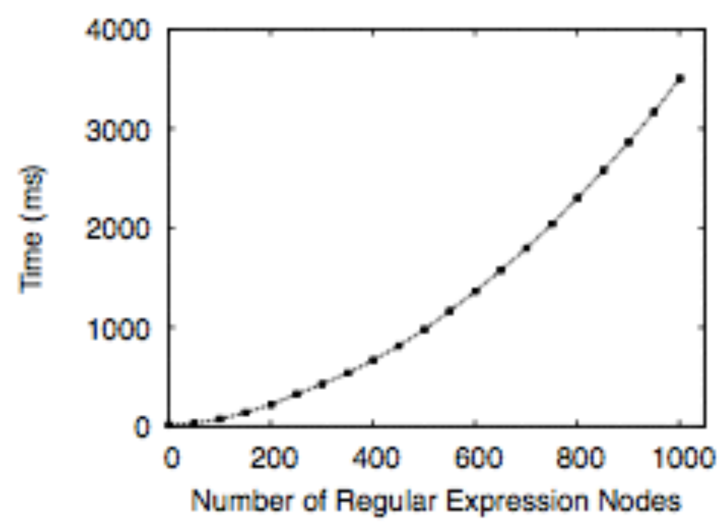
5% of traffic with bandwidth guarantees

Verification Is Very Fast

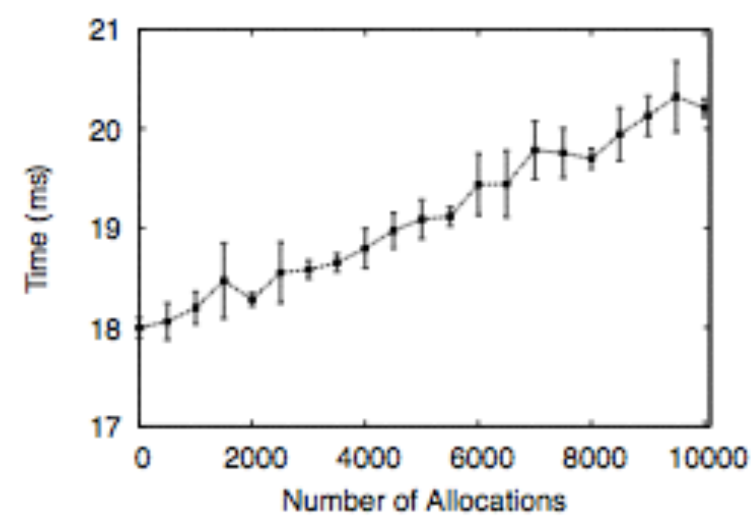
- 10,000 statements verified in less than 21ms
- Verifying resource allocations is very fast
- Verifying paths scales with complexity of the expression



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