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
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.
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.
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.

\[
\{ x : \\
  (\text{eth.src} = 00:00:00:00:00:01 \& \\
   \text{eth.dst} = 00:00:00:00:00:02 \& \\
   \text{tcp.dst} = 80) \\
  \rightarrow .* \text{nat} * . \text{dpi} . *
\}, \min(x,100\text{MB/s})
\]
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.

\[
[ x : \begin{align*}
& (\text{eth.src} = 00:00:00:00:00:01 & \text{eth.dst} = 00:00:00:00:00:02 & \text{tcp.dst} = 80) \\
& \rightarrow .* \text{ nat } .* \text{ dpi } .*
\end{align*}, \min(x,100\text{MB/s})]
\]
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**
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.

\[
\begin{array}{l}
\text{[ } x : \\
\quad (\text{eth.src} = \text{00:00:00:00:00:01} \& \text{eth.dst} = \text{00:00:00:00:00:02} \& \text{tcp.dst} = 80) \\
\quad \rightarrow .* \text{ nat } *. \text{ dpi } .* \\
\text{]}, \min(x, 100\text{MB/s})
\end{array}
\]
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.

\[
[ y : (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \text{tcp.dst} = 20) \to .* \text{ dpi } .* \\
  z : (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \text{tcp.dst} = 21) \to .* \\
], \\
\text{max}(y + z, 50\text{MB/s})
\]
Expressive Formulas

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

\[
\begin{align*}
[ & y : (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \\
& \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \\
& \text{tcp.dst} = 20) \rightarrow .* \text{ dpi } .* ] \\
& z : (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \\
& \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \\
& \text{tcp.dst} = 21) \rightarrow .* ] \\
\], \\
\text{max}(y + z, 50\text{MB/s})
\end{align*}
\]
Expressive Formulas

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

\[
\begin{align*}
[y &: (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \\
& \quad \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \\
& \quad \text{tcp.dst} = 20) \rightarrow \ast \text{ dpi } \ast ] \\
\{ \text{FTP data} \} \\
\{ \text{FTP control} \}
\end{align*}
\]

\[
\begin{align*}
z &: (\text{eth.src} = 00:00:00:00:00:01 \text{ and } \\
& \quad \text{eth.dst} = 00:00:00:00:00:02 \text{ and } \\
& \quad \text{tcp.dst} = 21) \rightarrow \ast \\
\]

\[
\text{max}(y + z, 50\text{MB/s})
\]
Expressive Formulas

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

\[
\begin{align*}
  y & : \text{eth.src} = 00:00:00:00:00:01 \text{ and eth.dst} = 00:00:00:00:00:02 \text{ and tcp.dst} = 20) \rightarrow .* \text{ dpi } .* \\
  z & : \text{eth.src} = 00:00:00:00:00:01 \text{ and eth.dst} = 00:00:00:00:00:02 \text{ and tcp.dst} = 21) \rightarrow .* \\
\end{align*}
\]

\[
\text{max}(y + z, 50\text{MB/s})
\]

\{ FTP data \}

\{ FTP control \}

\{ Bandwidth constraints written as formulas \}
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).

\[
\begin{align*}
\text{srcs} & := \{00:00:00:00:00:00:01\} \\
\text{dsts} & := \{00:00:00:00:00:00:02\} \\
\text{foreach} \ (s,d) \ \text{in} \ \text{cross(srcs,dsts)}: \\
& \ \ \ \ \ \ \ \ \ tcp.dst = 80 \rightarrow \\
& \ \ \ \ \ \ \ \ \ (.* \ \text{nat} \ .* \ \text{dpi} \ .*) \ \text{at} \ \text{max}(100\text{MB/s})
\end{align*}
\]
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).*

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

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

Localize policies, allocate resources, and generate target code
Localization

\[
\text{max}(y + z, \ 50\text{MB/s})
\]

\[
\text{max}(y, 25\text{MB/s}) + \text{max}(z, 25\text{MB/s})
\]

**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

\[
[ x :
    (\text{eth.src} = 00:00:00:00:00:00:01 \&
     \text{eth.dst} = 00:00:00:00:00:00:02 \&
     \text{tcp.dst} = 80)
    \rightarrow \text{nat} \& \text{dpi} \&
] , \text{min}(x, 100\text{MB/s})
\]
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

\[ x : \]
\[
(\text{eth.src} = 00:00:00:00:00:01 \& \text{eth.dst} = 00:00:00:00:00:02 \& \text{tcp.dst} = 80) \]
\[
\rightarrow \ast \text{n} \ast . \text{dpi} . \ast
\]
\[
\min(x, 100\text{MB/s})
\]
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
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.

\[
x : (\text{ip.src} = 192.168.1.1 \text{ and } \text{ip.dst} = 192.168.1.2) \rightarrow .*,\]
\[
\text{max}(x, 100\text{MB/s})
\]
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

\[
[x : (ip.src = 192.168.1.1 \text{ and } ip.dst = 192.168.1.2 \text{ and } tcp.dst = 22) \rightarrow .* ],
[y : (ip.src = 192.168.1.1 \text{ and } ip.dst = 192.168.1.2 \text{ and } tcp.dst = 80) \rightarrow .* \log .* ],
[z : (ip.src = 192.168.1.1 \text{ and } ip.dst = 192.168.1.2 \text{ and } !(tcpDst=22|tcpDst=80)) \rightarrow .* \ dpi .* ],
\]

max(x, 50MB/s) and max(y, 25MB/s) and max(z, 25MB/s)
Three Possible Transformations

\[
[x : (\text{ip.src} = 192.168.1.1 \text{ and } \text{ip.dst} = 192.168.1.2 \text{ and } \text{tcp.dst} = 22) \rightarrow \ast ],
\]

\[
[y : (\text{ip.src} = 192.168.1.1 \text{ and } \text{ip.dst} = 192.168.1.2 \text{ and } \text{tcp.dst} = 80) \rightarrow \ast \text{ log } \ast ],
\]

\[
[z : (\text{ip.src} = 192.168.1.1 \text{ and } \text{ip.dst} = 192.168.1.2 \text{ and } !(\text{tcpDst}=22|\text{tcpDst}=80)) \rightarrow \ast \text{ dpi } \ast ],
\]

max(x, 50MB/s)
and max(y, 25MB/s)
and max(z, 25MB/s)
Three Possible Transformations

\[
\begin{align*}
[x &: (ip.src = 192.168.1.1 \text{ and }
  ip.dst = 192.168.1.2 \text{ and }
  tcp.dst = 22) \rightarrow .* ]], \\
[y &: (ip.src = 192.168.1.1 \text{ and }
  ip.dst = 192.168.1.2 \text{ and }
  tcp.dst = 80) \rightarrow .* \text{ log .* }], \\
z &: (ip.src = 192.168.1.1 \text{ and }
  ip.dst = 192.168.1.2 \text{ and }
  !(tcpDst=22|tcpDst=80)) \rightarrow .* \text{ dpi .*}], \\
\text{max}(x, 50\text{MB/s}) \\
\text{and max}(y, 25\text{MB/s}) \\
\text{and max}(z, 25\text{MB/s})
\end{align*}
\]
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

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

Policies to manage Stanford network topology
Merlin Reduces Management Effort

- OpenFlow rules
- tc commands
- queue commands

Baseline (6 loc)
Bandwidth (11 loc)
Firewall (23 loc)
Middlebox (11 loc)
Combination (23 loc)
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 a 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
Delegated Merlin policies can be declared resource requirements ahead of time. The negotiators support two common approaches: (AIMD), tenants adjust resource demands by incrementally add additive-increase, multiplicative max-min fair-sharing (MMFS). With (AIMD), we ran three experiments to benchmark our negotiator verifies, the regular expressions, or the bandwidth allocations. We modified by negotiators in three ways: by changing the predicates, the regular expression's abstract syntax tree is used as a measure of its complexity. Finally, we increased the number of nodes in the parse tree. However, since regular expressions denote paths around 41 minutes to find a solution. To put that number of additional predicates generated in the delegated verification is extremely fast for increasing predicates and the regular expression in a network with reasonable overhead.

Our mixed-integer programming approach used for guaranteeing bandwidth scales to large networks quickly and our mixed-integer programming approach used for guaranteeing bandwidth scales in a network with reasonable overhead.

Figure 8 shows compilation times for an increasing number of traffic classes for (a) all pairs connectivity on a balanced tree, (b) 5% of the traffic with guaranteed priority on a balanced tree, (c) all pairs connectivity on a fat tree, (d) 5% of the traffic classes with guaranteed bandwidth.

Figure 7 shows the results. As expected, providing connectivity when 5% of the traffic classes receive bandwidth guarantees. Table 3 shows a sample of topology sizes and solution times for fat tree topologies with 5% of the traffic classes for various network sizes and guarantees. For both types of topologies, balanced trees, and fat trees, Merlin finds solutions for each traffic class.

Table 2 shows the results. As expected, providing connectivity when 5% of the traffic classes receive bandwidth guarantees.

Figure 8 shows compilation times for an increasing number of traffic classes for (a) all pairs connectivity on a balanced tree, (b) 5% of the traffic with guaranteed priority on a balanced tree, (c) all pairs connectivity on a fat tree, (d) 5% of the traffic classes with guaranteed bandwidth.

Figure 7 shows the results. As expected, providing connectivity when 5% of the traffic classes receive 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

![Graphs showing time (ms) vs. number of statements, number of regular expression nodes, and number of allocations.](image)

- 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