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.
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.

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

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

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

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

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

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

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

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

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

\],
\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.

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

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

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

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

```
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)
```
**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 := {00:00:00:00:00:01}
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).

```latex
\begin{verbatim}
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)
\end{verbatim}
```
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}
for (s,d) in cross(srcs,dsts):
    tcp.dst = 80 ->
    ( .* nat .* dpi .* ) at max(100MB/s)
```

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

Localize policies, allocate resources, and generate target code
Localization

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

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

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

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

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

\[ x : \\
(\text{eth.src} = 00:00:00:00:00:01 \land \text{eth.dst} = 00:00:00:00:00:02 \land \text{tcp.dst} = 80) \\
\rightarrow * \text{nat} * . \text{dpi} . * \]

\[ \min(x, 100\text{MB/s}) \]

Convert to DFA

Note resource demands
Solve MIP to Determine Paths and Placement

Physical topology with vertices $V$  

Statement NFA with states $Q_i$  

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
**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

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

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

\[ x : (ip.\text{src} = 192.168.1.1 \text{ and } \text{ip.\text{dst}} = 192.168.1.2 \text{ and } \text{tcp.\text{dst}} = 22) \rightarrow \ast ] ,
\[ y : (ip.\text{src} = 192.168.1.1 \text{ and } \text{ip.\text{dst}} = 192.168.1.2 \text{ and } \text{tcp.\text{dst}} = 80) \rightarrow \ast \text{ log } \ast ] ,
\[ z : (ip.\text{src} = 192.168.1.1 \text{ and } \text{ip.\text{dst}} = 192.168.1.2 \text{ and } \neg (\text{tcp\text{\text{Dst}}=22} \text{ or } \text{tcp\text{\text{Dst}}=80}) ) \rightarrow \ast \text{ dpi } \ast ] ,
\]

\[
\text{max}(x, 50\text{MB/s}) \\
\text{and max}(y, 25\text{MB/s}) \\
\text{and max}(z, 25\text{MB/s})
\]
Three Possible Transformations

\[ x : (ip\.src = 192\cdot168\cdot1.1 \text{ and } ip\.dst = 192\cdot168\cdot1.2 \text{ and } tcp\.dst = 22) \rightarrow .* ], \]
\[ y : (ip\.src = 192\cdot168\cdot1.1 \text{ and } ip\.dst = 192\cdot168\cdot1.2 \text{ and } tcp\.dst = 80) \rightarrow .* \log .* ], \]
\[ z : (ip\.src = 192\cdot168\cdot1.1 \text{ and } ip\.dst = 192\cdot168\cdot1.2 \text{ and } ! (tcpDst=22 | tcpDst=80) ) \rightarrow .* dpi .* ], \]
max(x, 50MB/s) 
and max(y, 25MB/s) 
and max(z, 25MB/s)
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
Evaluation

Demonstrating Merlin’s expressiveness, ability to manage the network, and scalability
Merlin Network Management

<table>
<thead>
<tr>
<th>Policies</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Baseline</strong></td>
<td>Basic all-pairs connectivity between hosts</td>
</tr>
<tr>
<td><strong>Bandwidth</strong></td>
<td>10% of traffic classes get a guarantee of 1Mbps, and a cap of 1Gbps</td>
</tr>
<tr>
<td><strong>Firewall</strong></td>
<td>All packets with tcp.dst = 80 are routed through a firewall</td>
</tr>
<tr>
<td><strong>Middlebox</strong></td>
<td>Hosts are partitioned into two sets (trusted and untrusted). Inter-set traffic must pass through a middle box.</td>
</tr>
<tr>
<td><strong>Combination</strong></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 fundamental approach for fault-tolerant services
- Measure throughput for co-located key-value store service backed by SMR
- Merlin prioritizes traffic for one service
Compiliation 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

![Graphs showing increasing time with increasing statements, path expressions, and bandwidth constraints.](image)
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