Simplifying Network Optimization for SDN Deployment

Victor Heorhiadi
UNC Chapel Hill

Michael K. Reiter
UNC Chapel Hill

Vyas Sekar
Carnegie Mellon University
Overview: SDN

SDN applications

Network data

Network routes

Control Platform (e.g., ONOS, OpenDaylight)

Data plane
Network Optimizations are Common

• Maxflow, Traffic engineering
• SIMPLE (SIGCOMM 2013)
• ElasticTree (NSDI 2010)
• Panopticon (Usenix ATC 2014)
• SWAN (SIGCOMM 2013)
Current Process

1. Take theory & optimization courses
2. Formulate the problem
3. Solve with a solver
   - Not fast enough
     - NP hard?
4. Develop heuristic
5. Parse solution
6. Deploy
Our Vision

- No custom heuristics
- Focus on high-level network goals
- Rapid prototyping
- App = 20 lines of code

SDN applications
Control Platform (e.g., ONOS, OpenDaylight)
Optimization layer
Challenge: Generality + Efficiency

<table>
<thead>
<tr>
<th>Approach</th>
<th>Generality</th>
<th>Efficiency</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frameworks</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Custom solutions</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>SOL</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
SOL: SDN Optimization Layer

- SDN applications
  - A A A A A A A
- SOL API
- Network data
- Network routes
- Optimization solver (e.g., CPLEX)
- Control Platform (e.g., ONOS, OpenDaylight)
- Logically centralized
- Diverse set
Insight: Path Abstraction

• Problems are *recast* to be **path-based**

• Policies are path predicates
Path-based Recasting

Edge-based

\( f : \) amount of flow

\[
\begin{align*}
fe_1 &= fe_3 + fe_4 \\
fe_2 &= fe_4 + fe_5 + fe_6 + fe_8
\end{align*}
\]

Path-based

\[
\begin{align*}
f_{p1} &= \sum_{i=1}^{k} f_{pi} = \text{demand} \\
f_{p2} &= \sum_{i=1}^{k} f_{pi} = \text{demand} \\
&\vdots \\
f_{pk} &= \sum_{i=1}^{k} f_{pi} = \text{demand}
\end{align*}
\]
Policies as Path Predicates

Valid paths:
• N1-N4-N5
• N1-N3-N4-N5

Invalid paths:
• N1-N3-N5

N1→N5
Web, 100 Mbps
FW→Proxy
Path Challenge

Exponential number of paths

Large optimization size

Long run time = Bad efficiency
1. Enumerate all simple paths
2. Keep valid paths (according to a predicate) **Offline step**

1. Pick a subset of paths
   This acts as a **heuristic**

1. Model resource usage and constraints
2. Solve

**Rule generation**

**Efficiency**

Use a controller to configure data plane paths
Implementation

• Python library; interfaces with CPLEX solver and ONOS controller

• Prototyped applications
  • MaxFlow, Traffic engineering, latency minimization
  • ElasticTree (Heller et al.), Panopticon (Levin et al.), SIMPLE (Qazi et al.)
Example: MaxFlow

1. \texttt{opt, pptc = initOptimization(topo, trafficClasses, nullPredicate, 'shortest', 5)}
2. \texttt{opt.allocateFlow(pptc)}
3. \texttt{linkcapfunc = lambda link, tc, path, resource: tc.volBytes}
4. \texttt{opt.capLinks(pptc, 'bandwidth', linkConstrCaps, linkcapfunc)}
5. \texttt{opt.maxFlow(pptc)}
6. \texttt{opt.solve()}
Example: Traffic Engineering

1. `opt, pptc = initOptimization(topo, trafficClasses, nullPredicate, 'shortest', 5)`
2. `opt.allocateFlow(pptc)`
3. `linkcapfunc = lambda link, tc, path, resource: tc.volBytes`
4. `opt.capLinks(pptc, 'bandwidth', linkConstrCaps, linkcapfunc)`
5. `opt.routeAll(pptc)`
6. `opt.minLinkLoad('bandwidth')`
7. `opt.solve()`

Route all traffic
Minimize bandwidth load
Key Questions

• Does it reduce development effort for more complex applications?

• Is it faster than the original optimization?

• Is it any worse than optimal?
## Development effort

<table>
<thead>
<tr>
<th>Application</th>
<th>SOL lines of code</th>
<th>Estimated improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>ElasticTree (Heller et al.)</td>
<td>16</td>
<td>21.8×</td>
</tr>
<tr>
<td>Panoption (Levin et al.)</td>
<td>13</td>
<td>25.7×</td>
</tr>
<tr>
<td>SIMPLE (Qazi et al.)</td>
<td>21</td>
<td>18.6×</td>
</tr>
</tbody>
</table>
- Orders of magnitude faster
- Less than 1% away from optimal

Shaded: No solution by the original within 30 minutes
Potential Future Directions

• Analytically show why path selection is effective
• Path selection that honors bounds on optimality
Summary

• Getting SDN benefits requires a lot of optimization knowledge
• SOL lowers barrier of entry for developers
• Leverages the path abstraction: generation + selection
• Efficient: deploy in seconds!
• Creates many new opportunities for future work

victor@cs.unc.edu  https://github.com/progwriter/SOL
http://cs.unc.edu/~victor/papers/sol.pdf
Mininet Tests

Setup:
• Traffic engineering application
• Mininet + ONOS

0 → functioning network in 15 seconds
Runtime as Function of Number of Paths

Panopticon

Time (s)

Abilene (11)
Quest (20)
Geant2012 (40)
Dfn (58)
Bellcanada (48)

Opt. gap

Number of paths (per class)

SIMPLE

Time (s)

Number of paths (per class)

Opt. gap
Comparison to Merlin (Soulé et al.)

Log Scale

Shaded: No solution by Merlin within 30 minutes

Topology (number of switches)
“Mindiff” Across Optimizations

- Minimize network churn
- Minimize reconfiguration time
- Application agnostic
Results: reconfiguration

Traffic engineering application; Change in traffic demands triggers re-computation

Lower is better
Path Generation Time

- SIMPLE
- Panopticon
Limitations

• Mediocre performance on large networks with no chaining policies

• Limited theoretical insight into good path selection strategies