Simplifying Software-Defined Network Optimization Using SOL

Victor Heorhiadi  Michael K. Reiter  Vyas Sekar
UNC Chapel Hill  UNC Chapel Hill  Carnegie Mellon University
Overview: SDN

SDN applications

Network data

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

Network routes

Data plane
Network Optimizations are Common

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

- Take theory & optimization courses
- Formulate the problem
- Solve with a solver
- Not fast enough
  - NP hard?
- Deploy
- Parse solution
- Develop heuristic
SDN applications

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

Optimization layer

Our Vision

- No custom heuristics
- Focus on high-level network goals
- Rapid prototyping
- App = 20 lines of code
## 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
- Optimization solver (e.g., CPLEX)
- Control Platform (e.g., ONOS, OpenDaylight)
- Network data
- Network routes

Diverse set
Logically centralized
Insight: Path Abstraction

- Problems are *recast* to be *path-based*
- Policies are path predicates
Path-based Recasting: MaxFlow

Edge-based

\( f: \text{amount of flow} \)

\[ f_{e1} = f_{e3} + f_{e4} \]

Path-based

\[ \sum_{i=1}^{k} f_{pi} = \text{demand} \]
Valid paths:
- N1-N4-N5
- N1-N3-N4-N5

Invalid paths:
- N1-N3-N5
Path Challenge

Exponential number of paths

Large optimization size

Long run time = Bad efficiency
SOL Process

Path generation

1. Enumerate all simple paths
2. Keep valid paths (according to a predicate)
   **Offline step**

Path selection

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

Optimization

1. Model resource usage and constraints
2. Solve

Rule generation

Use a controller to configure data plane paths

**Efficiency**
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. `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.maxFlow(pptc)`
6. `opt.solve()`

**Topology input**

**Path generation + selection**

**Traffic flows**

**Resource consumption**

**Global goal (objective function)**
Example: Traffic Engineering

1. opt, pptic = initOptimization(topo, trafficClasses, nullPredicate, 'shortest', 5)
2. opt.allocateFlow(pptic)
3. linkcapfunc = lambda link, tc, path, resource: tc.volBytes
4. opt.capLinks(pptic, 'bandwidth', linkConstrCaps, linkcapfunc)
5. opt.routeAll(pptic)
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>
Optimization Runtime

- Orders of magnitude **faster**
- Less than 1% away from **optimal**

Log Scale

Shaded: No solution by the original within 30 minutes
Mininet Tests

Setup:
• Traffic engineering application
• Mininet + ONOS

0 → functioning network in 15 seconds

Time to deploy

<table>
<thead>
<tr>
<th>Topology (number of switches)</th>
<th>Optimization</th>
<th>Route generation</th>
<th>Route installation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abilene (11)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Quest (20)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Geant2012 (40)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bellcanada (48)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dfn (58)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Internode (66)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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