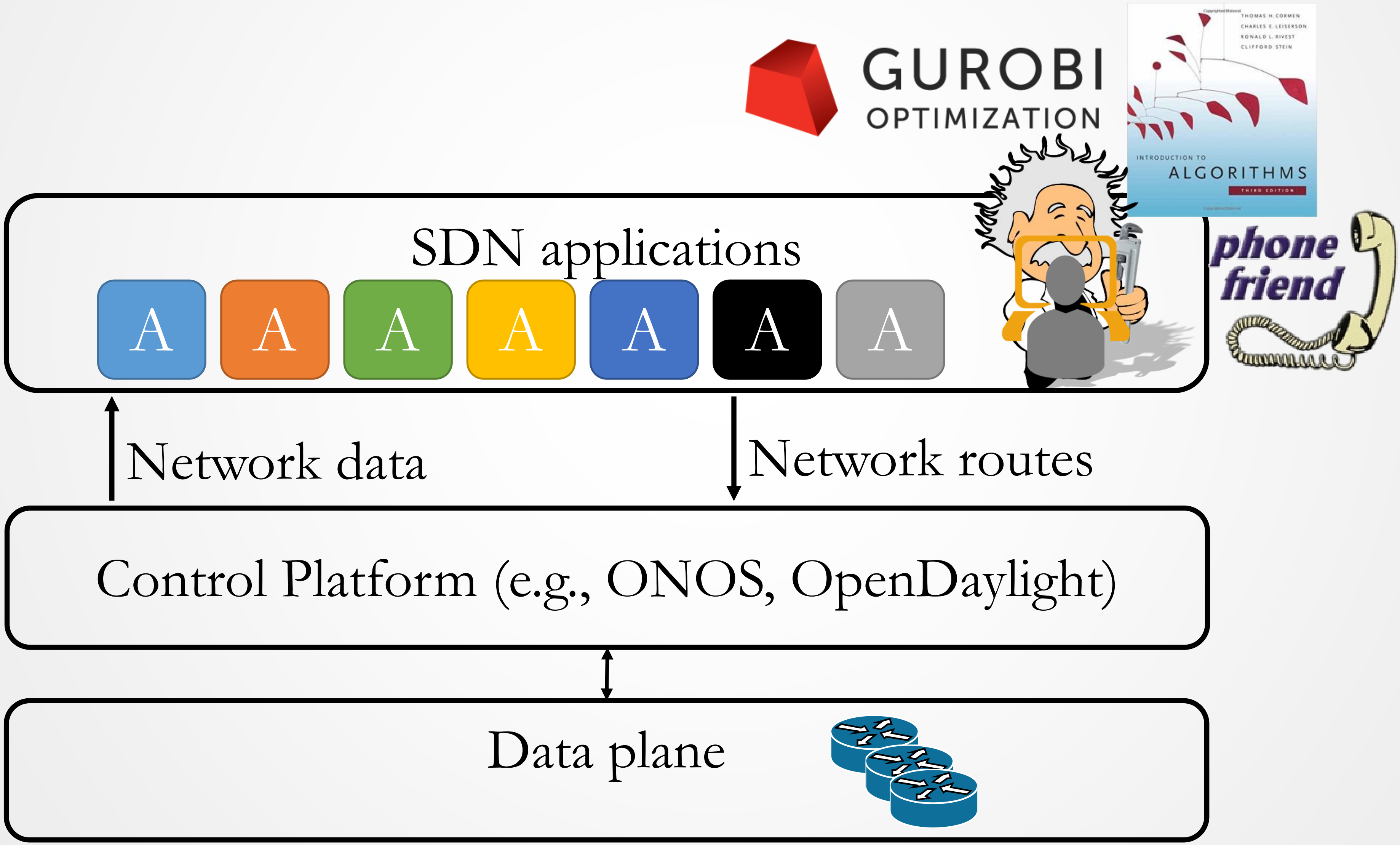
Simplifying Network Optimization for SDN Deployment

Victor Heorhiadi
UNC Chapel Hill

Michael K. Reiter
UNC Chapel Hill

Vyas Sekar
Carnegie Mellon University

Overview: SDN



SDN applications

A A A A A A A

Network data

Network routes

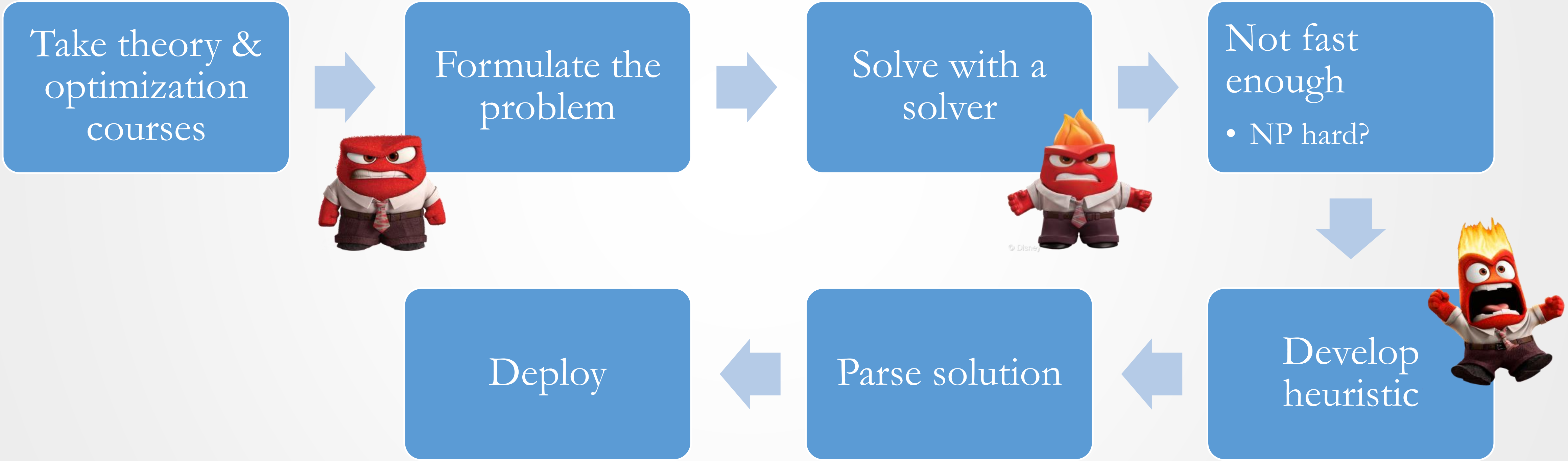
Control Platform (e.g., ONOS, OpenDaylight)

Data plane

Network Optimizations are Common

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

Current Process



Take theory & optimization courses

Formulate the problem

Solve with a solver

Not fast enough

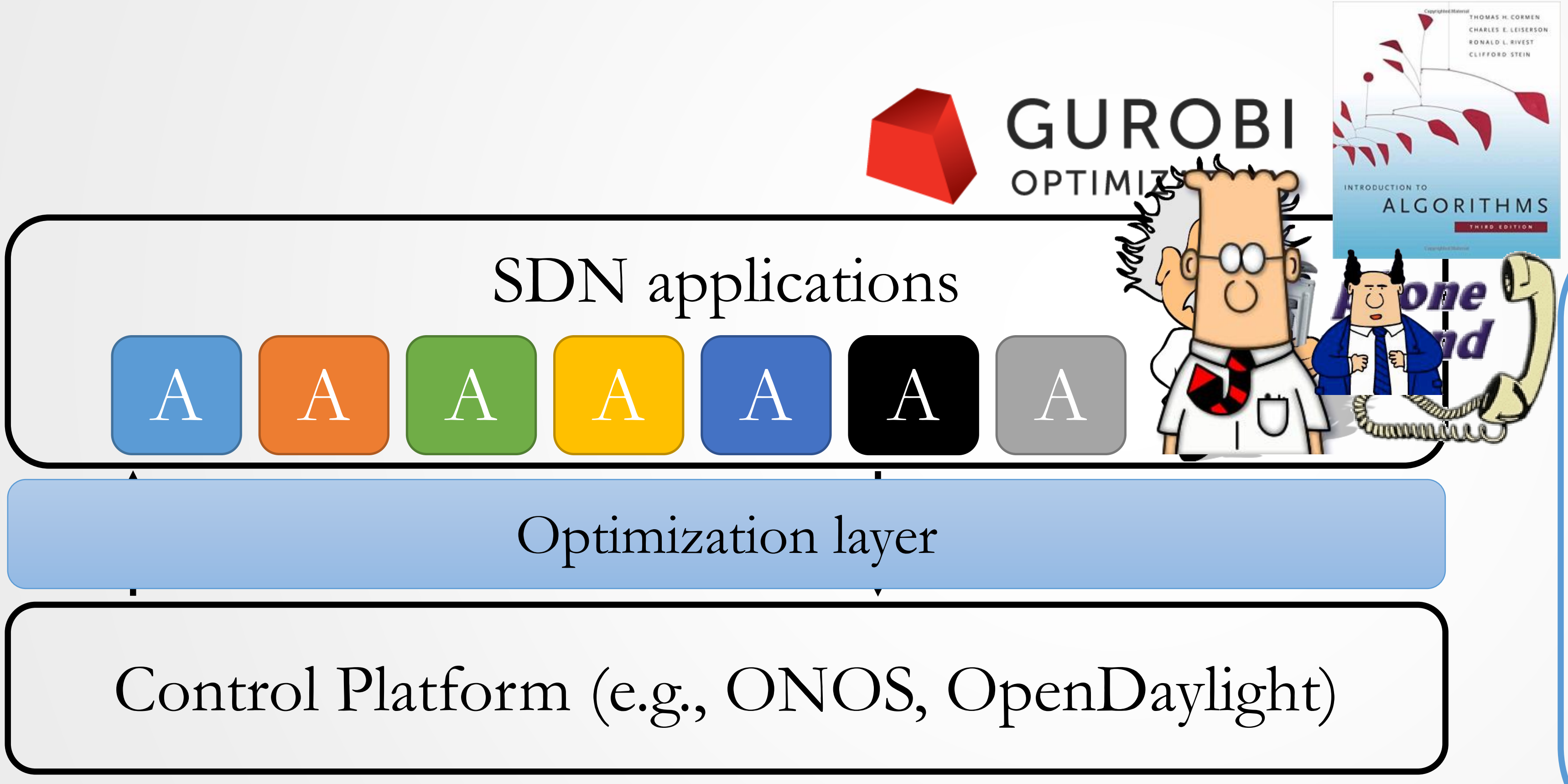
- NP hard?

Develop heuristic

Parse solution

Deploy

Our Vision



SDN applications

A A A A A A A

Optimization layer

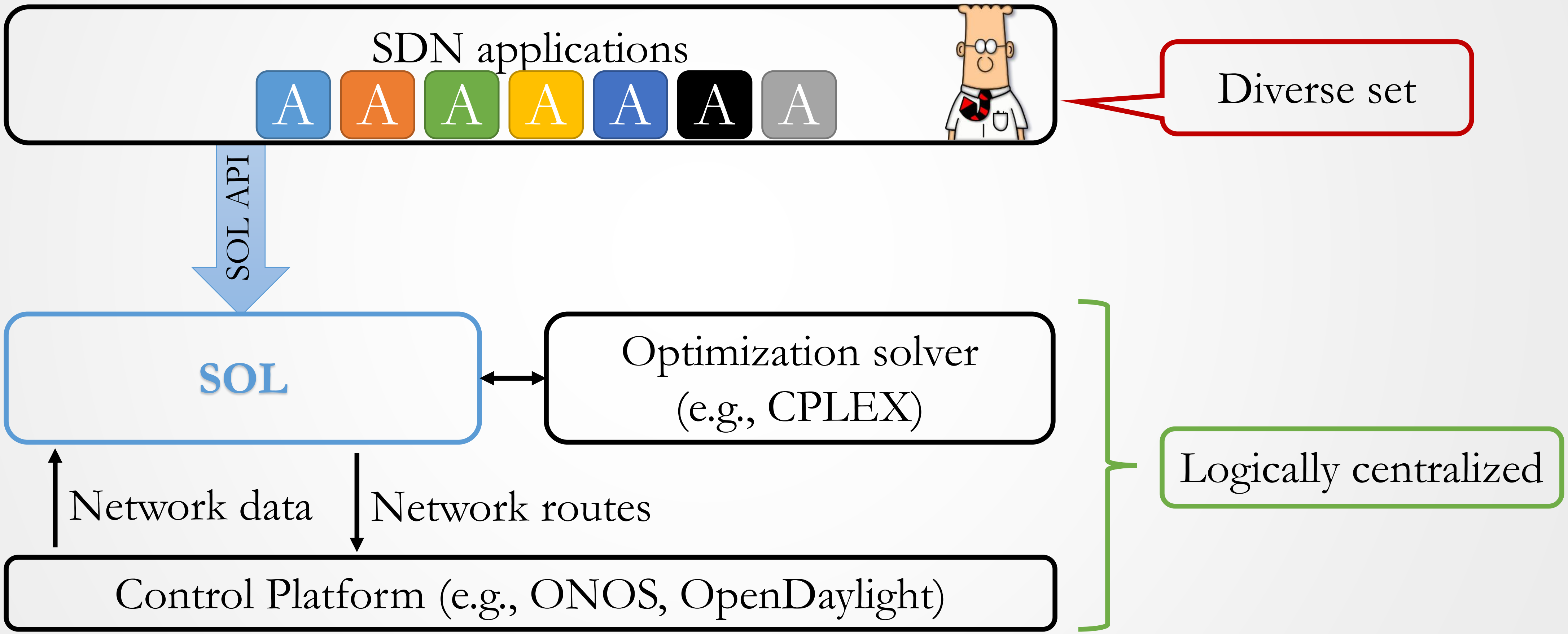
Control Platform (e.g., ONOS, OpenDaylight)

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

Challenge: Generality + Efficiency

Approach	Generality	Efficiency
Frameworks	✓	✗
Custom solutions	✗	✓
SOL	✓	✓

SOL: **S**DN **O**ptimization **L**ayer



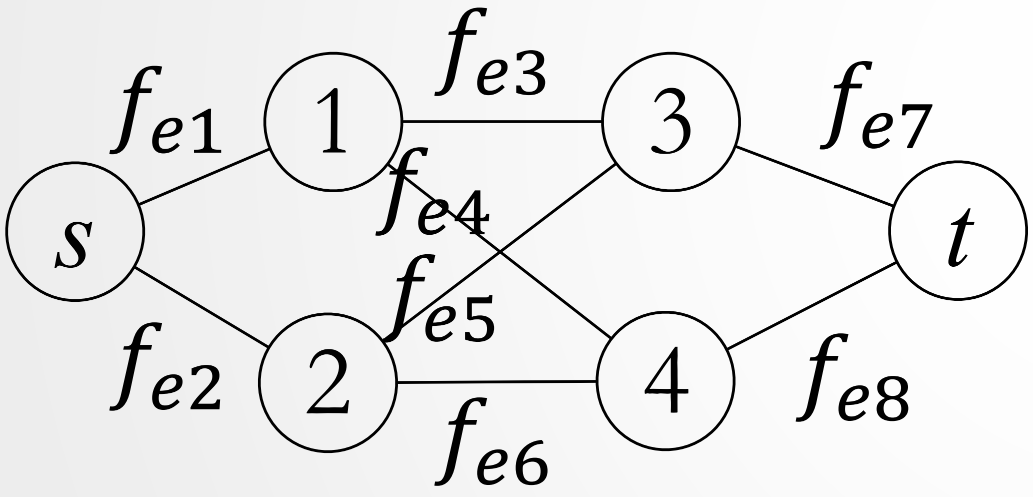
Insight: Path Abstraction

- Problems are *recast* to be **path-based**
- Policies are path predicates

Path-based Recasting

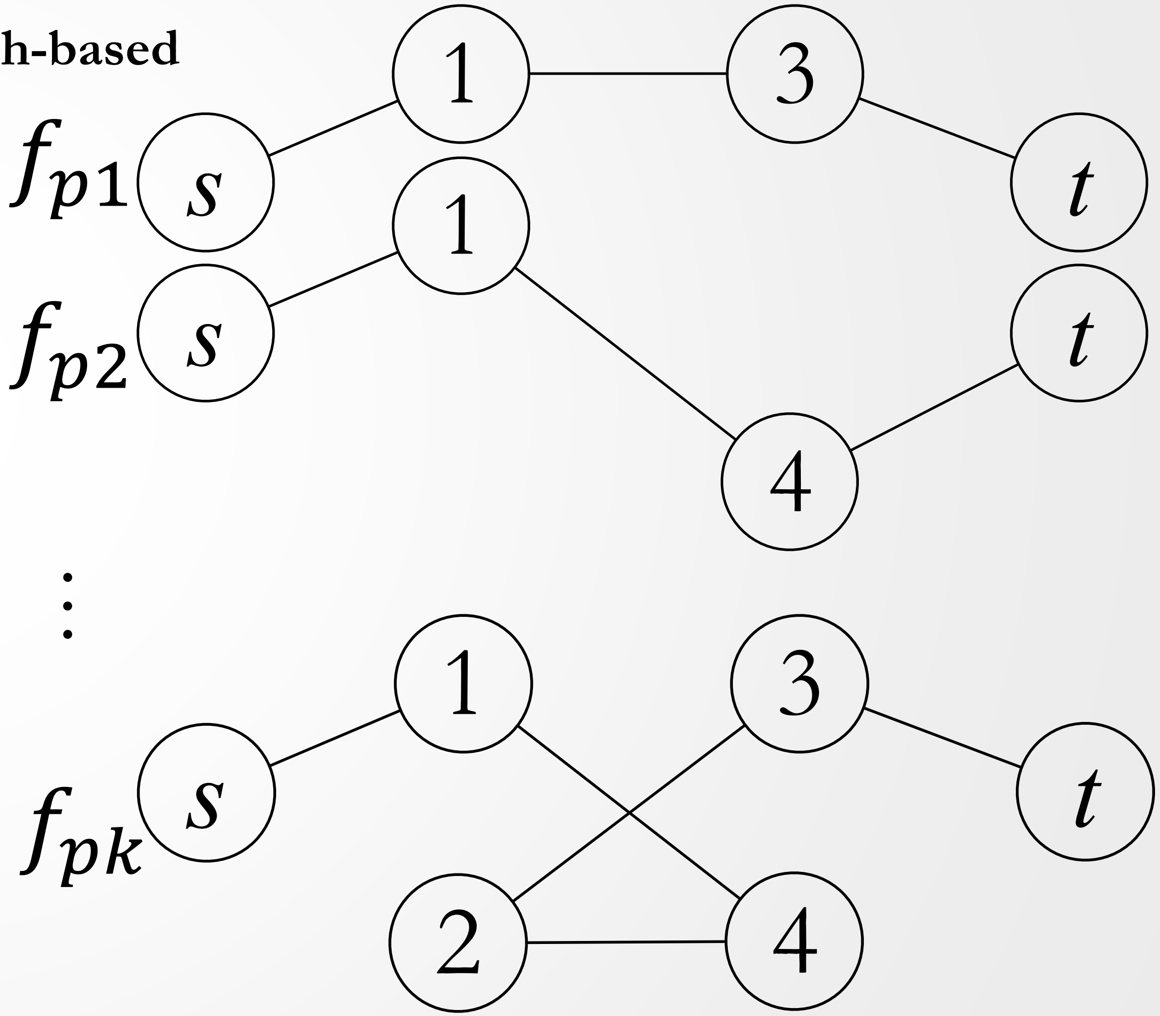
Edge-based

f: amount of flow



$$f_{e1} = f_{e3} + f_{e4}$$

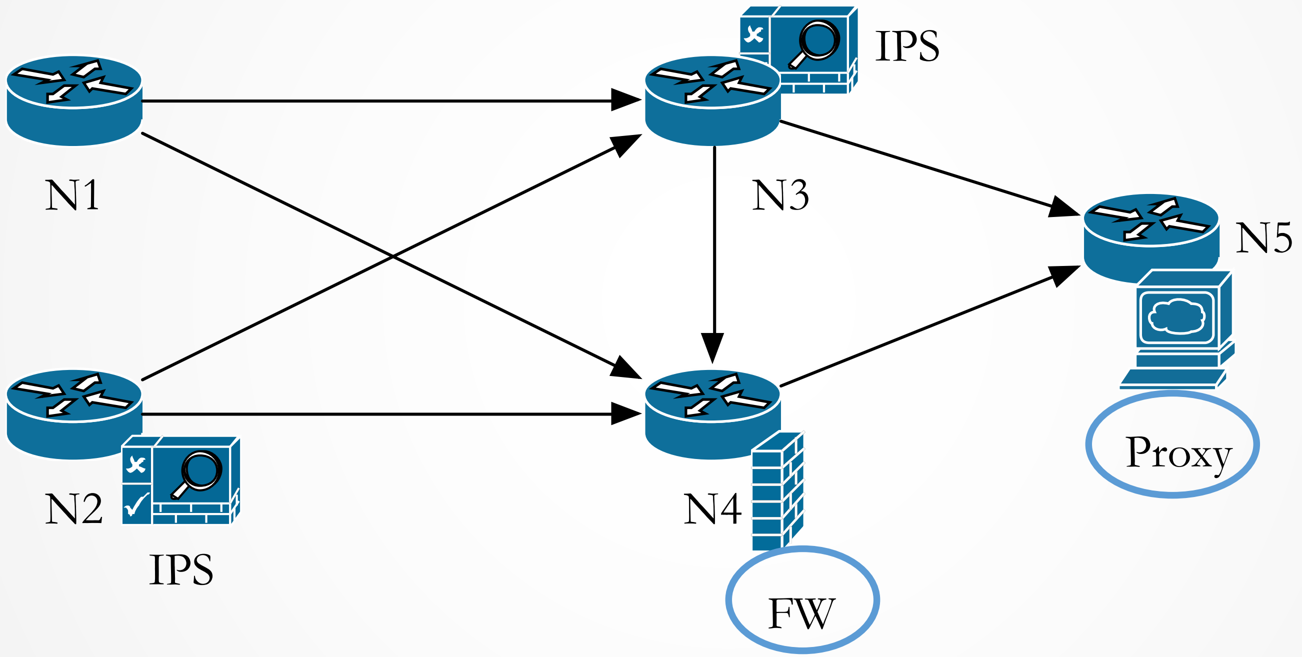
Path-based



$\sum_{i=1}^{k} f_{pi}$ = demand

Policies as Path Predicates

N1→N5
Web, 100 Mbps
FW→Proxy



Valid paths:

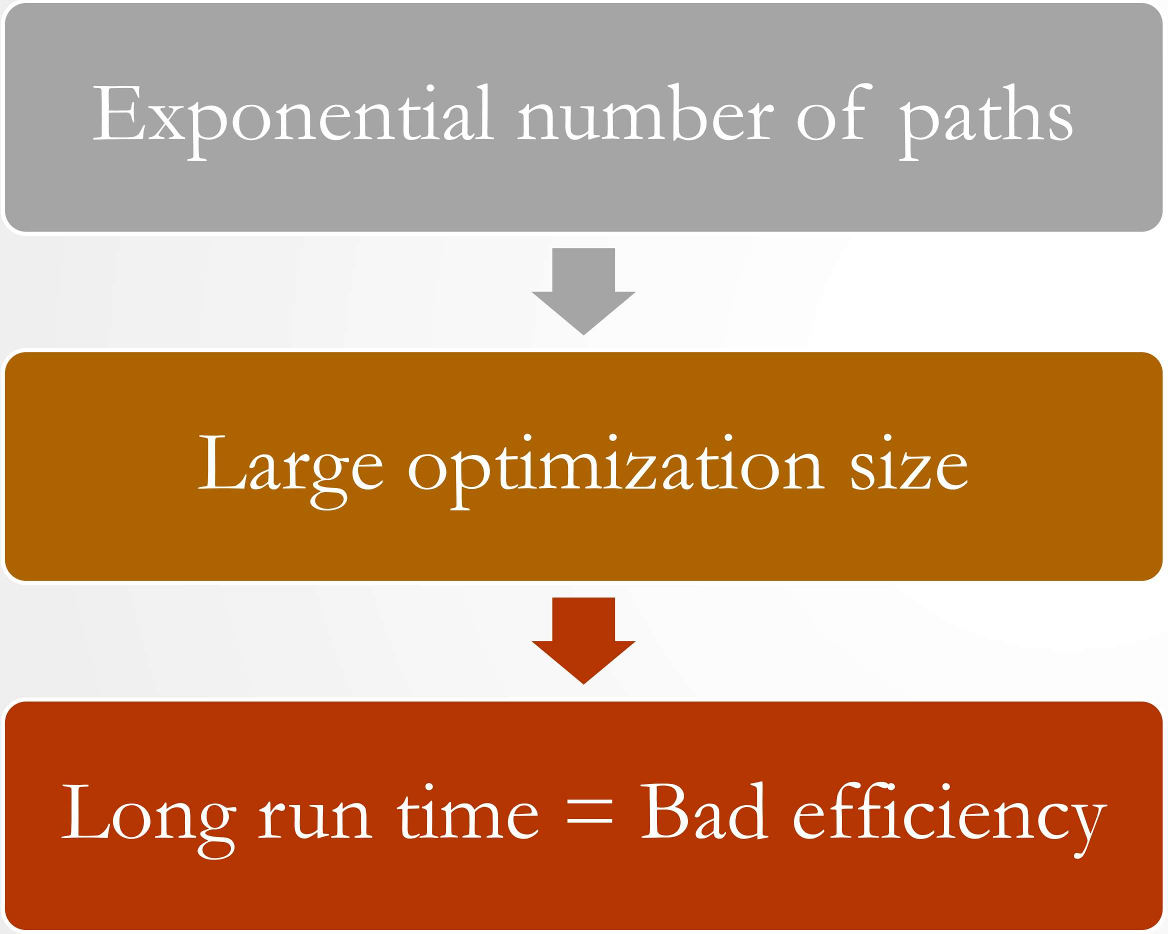
- N1-N4-N5
- N1-N3-N4-N5

Invalid paths:

- N1-N3-N5

Generality

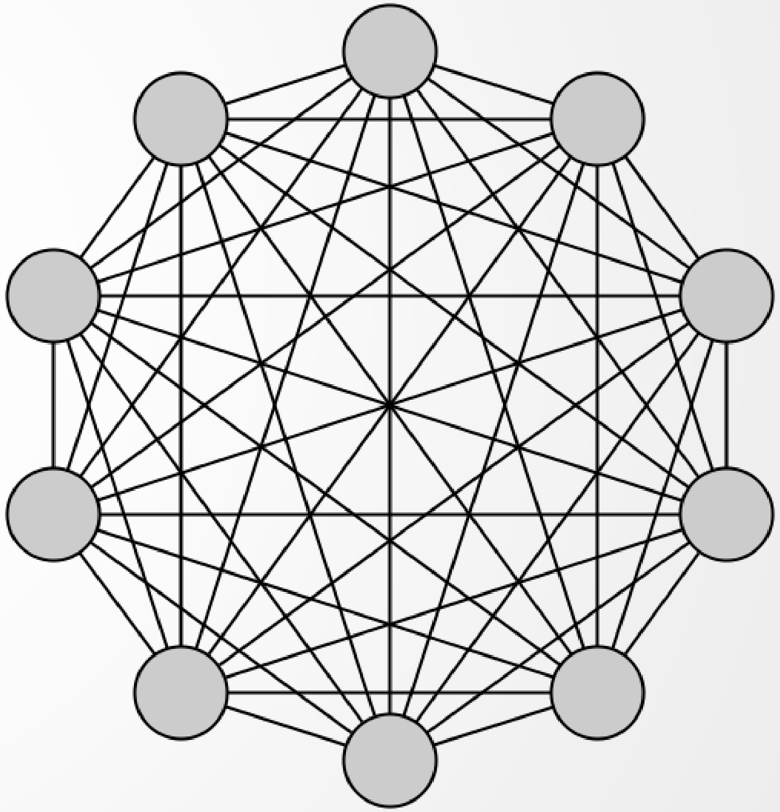
Path Challenge



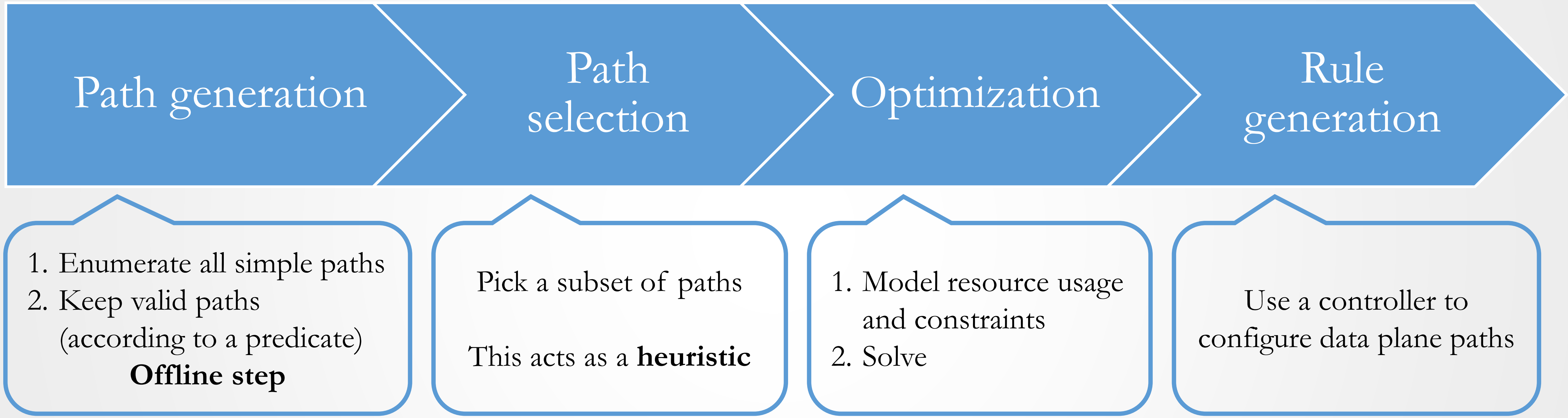
Exponential number of paths

Large optimization size

Long run time = Bad efficiency



SOL Process



Path generation

Path selection

Optimization

Rule generation

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

Offline step

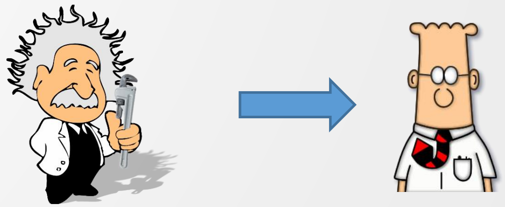
Pick a subset of paths

This acts as a **heuristic**

1. Model resource usage and constraints
2. Solve

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

Topology input

Path generation + selection

```
opt, pptc = initOptimization(topo, trafficClasses, nullPredicate, 'shortest', 5)
opt.allocateFlow(pptc)
linkcapfunc = lambda link, tc, path, resource: tc.volBytes
opt.capLinks(pptc, 'bandwidth', linkConstrCaps, linkcapfunc)
opt.maxFlow(pptc)
opt.solve()
```

Traffic flows

Resource consumption

Global goal (objective function)

Example: Traffic Engineering

```
opt, pptc = initOptimization(topo, trafficClasses, nullPredicate, 'shortest', 5)
opt.allocateFlow(pptc)
linkcapfunc = lambda link, tc, path, resource: tc.volBytes
opt.capLinks(pptc, 'bandwidth', linkConstrCaps, linkcapfunc)
opt.routeAll(pptc)
opt.minLinkLoad('bandwidth')
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?

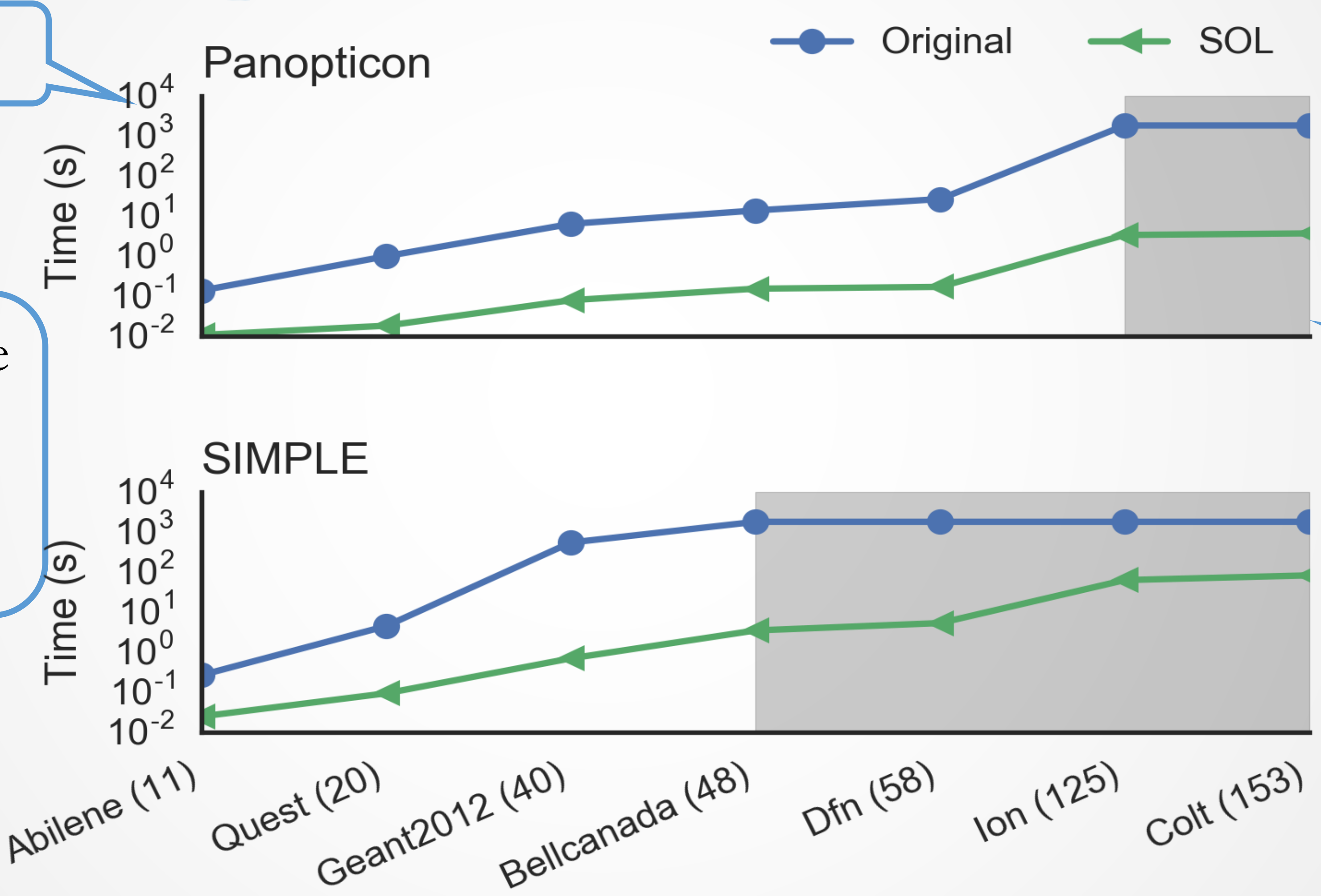
Development effort

Application	SOL lines of code	Estimated improvement
ElasticTree (Heller et al.)	16	21.8×
Panoption (Levin et al.)	13	25.7×
SIMPLE (Qazi et al.)	21	18.6×

Optimization Runtime

Log Scale

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



Shaded: No solution by the original within 30 minutes

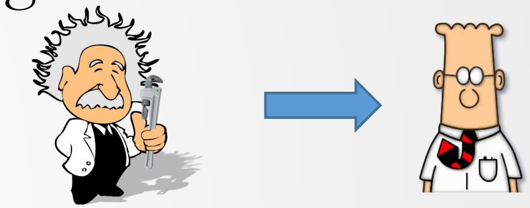
Topology (number of switches)

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

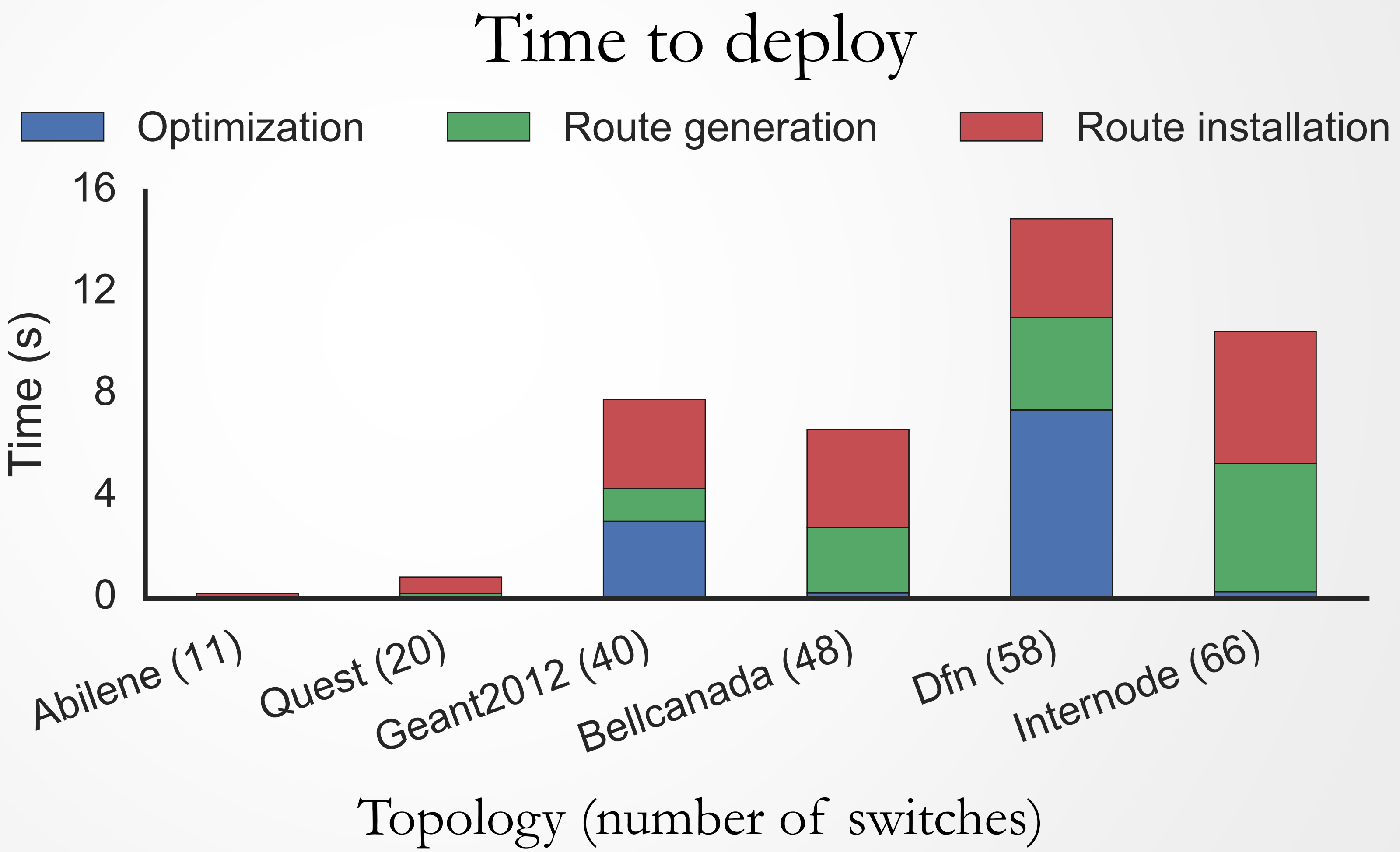
http://cs.unc.edu/~victor/papers/sol.pdf

Mininet Tests

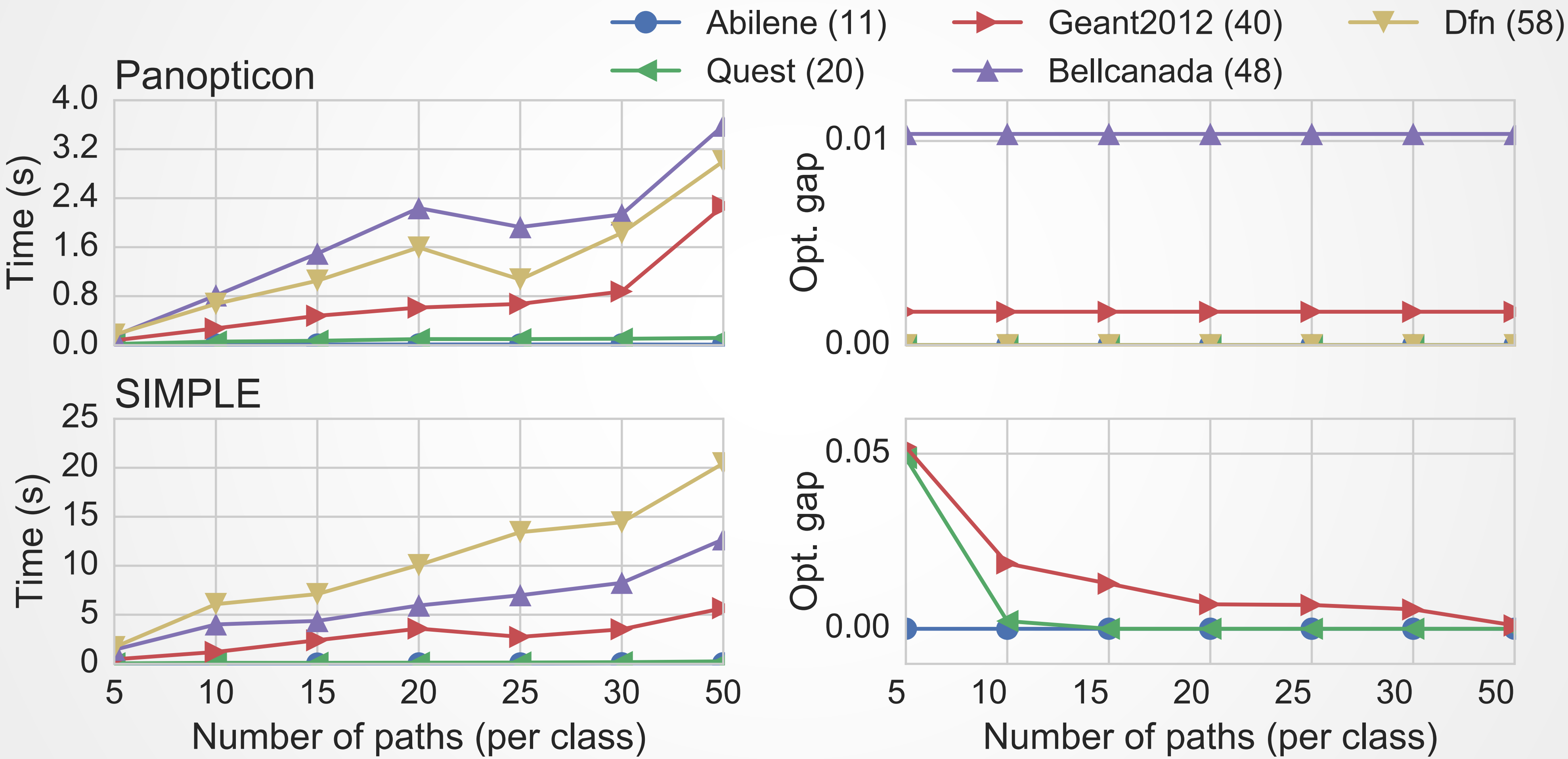
Setup:

- Traffic engineering application
- Mininet + ONOS

0 → functioning network
in 15 seconds

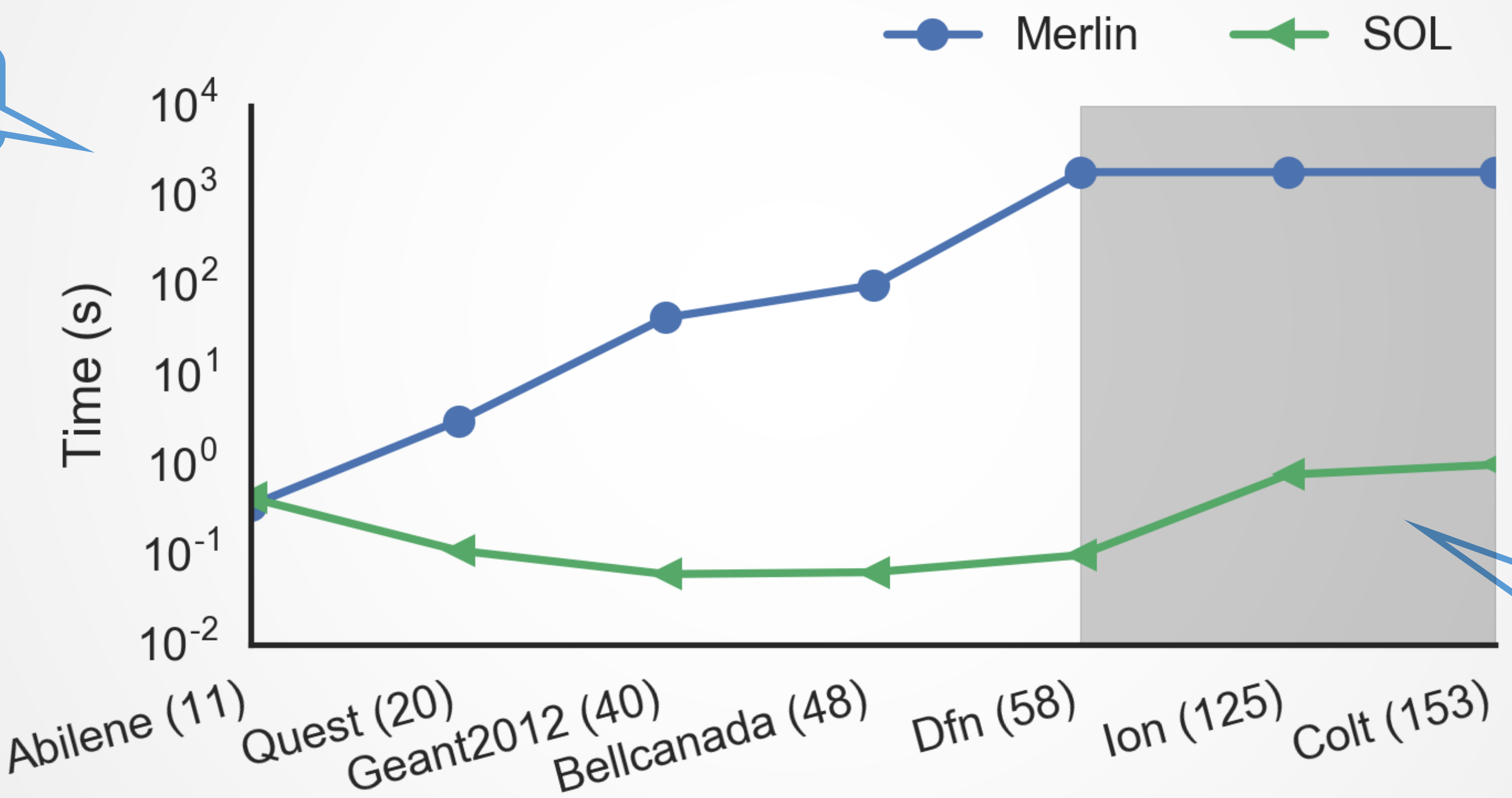


Runtime as Function of Number of Paths



Comparison to Merlin (Soulé et al.)

Log Scale

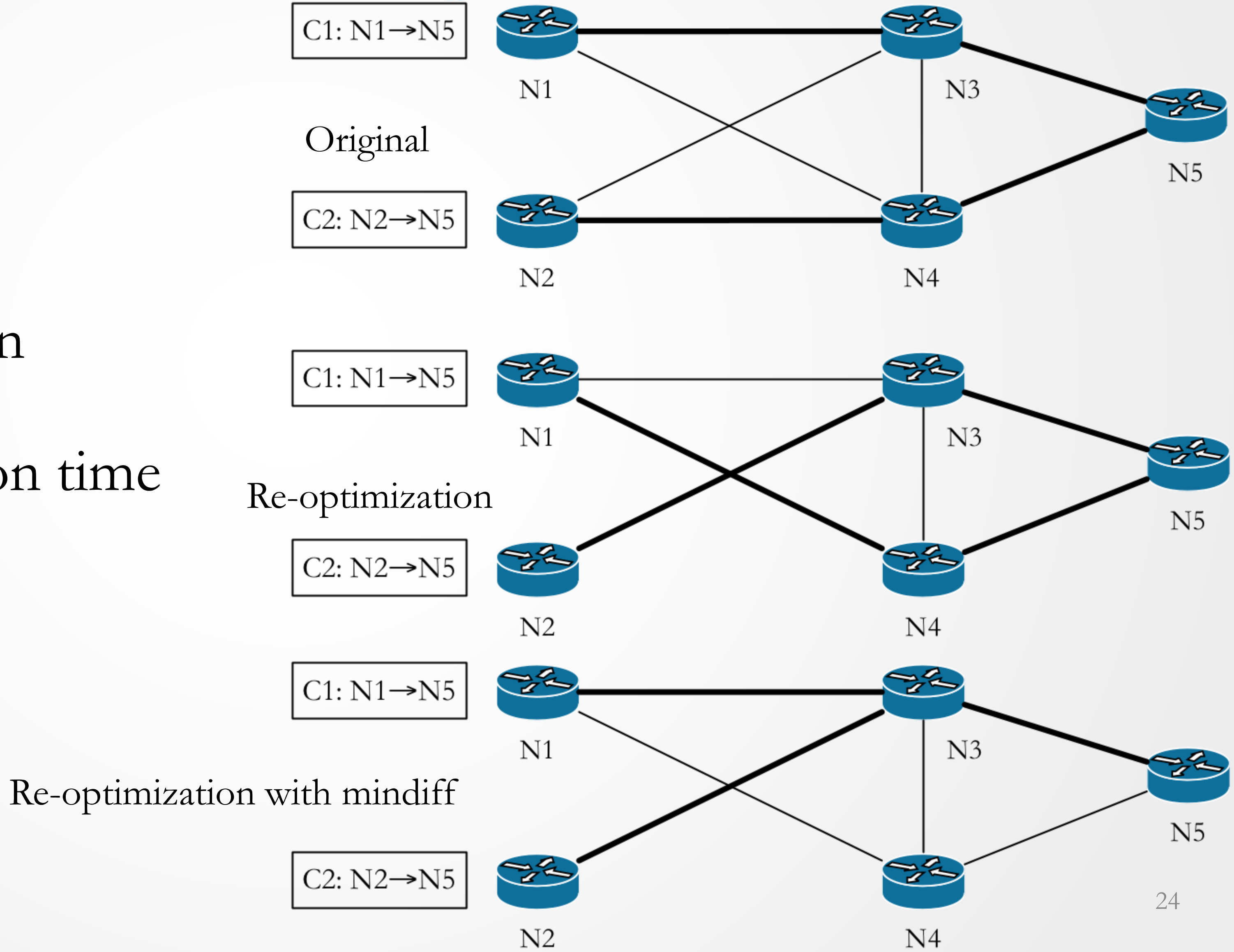


Shaded: No solution by Merlin within 30 minutes

Topology (number of switches)

“Mindiff” Across Optimizations

- Minimize network churn
- Minimize reconfiguration time
- Application agnostic



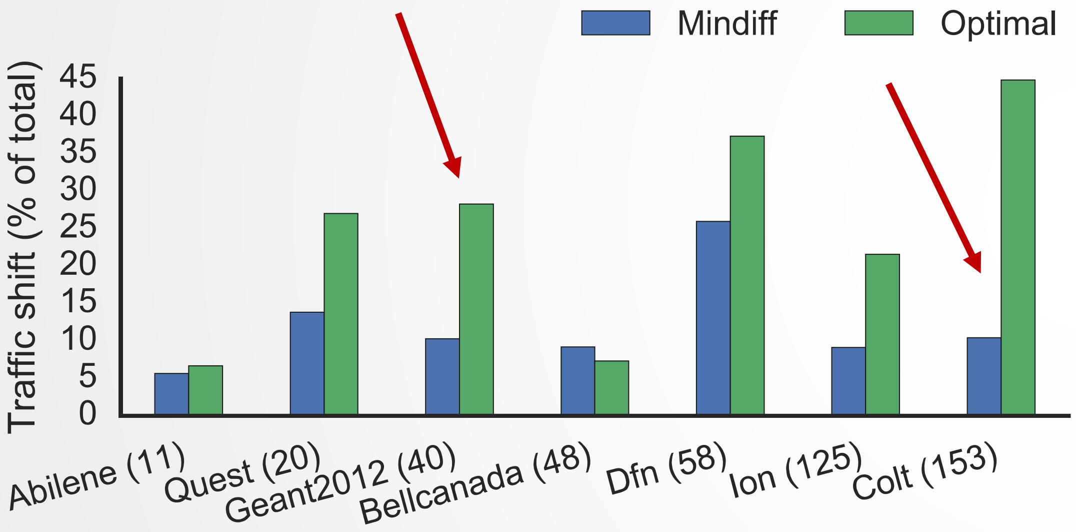
Original

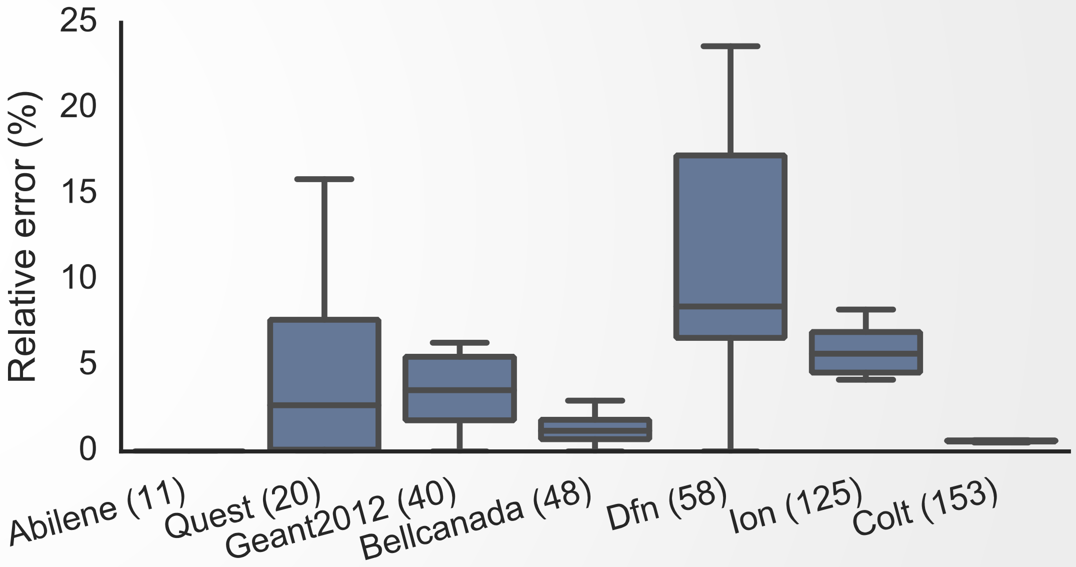
Re-optimization

Re-optimization with mindiff

Results: reconfiguration

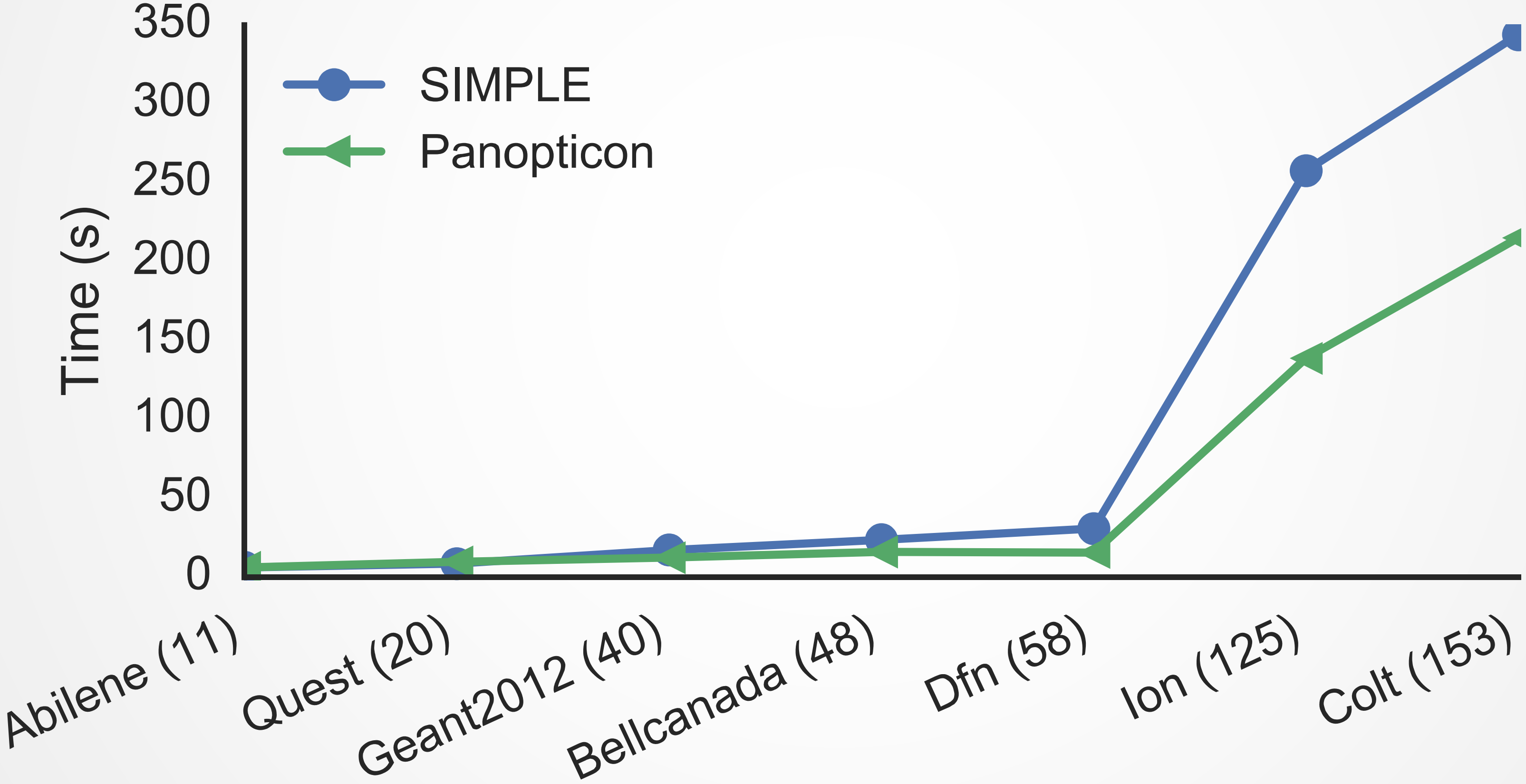
Traffic engineering application; Change in traffic demands triggers re-computation





Lower is better

Path Generation Time



Limitations

- Mediocre performance on large networks with no chaining policies
- Limited theoretical insight into good path selection strategies