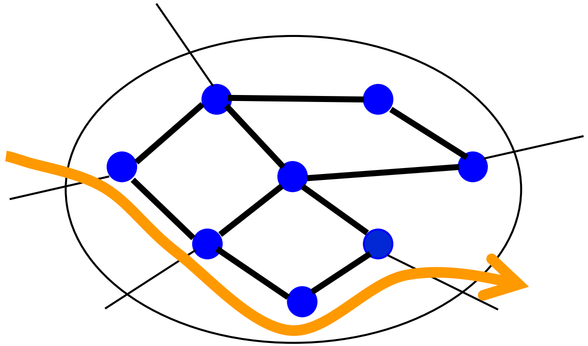
SmartEntry: Mitigating Routing Update Overhead with Reinforcement Learning for Traffic Engineering

Junjie Zhang, Zehua Guo, **Minghao Ye**, H. Jonathan Chao

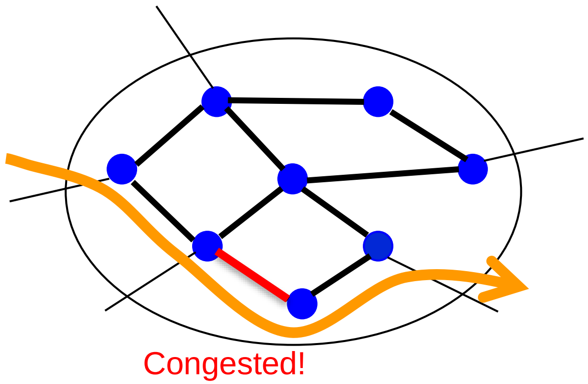
Background

- **Traffic Engineering (TE):** Configure routing to improve network performance
- **Metric:** Maximum Link Utilization (MLU) → $\frac{\text{Load}}{\text{Capacity}}$ of the most congested link



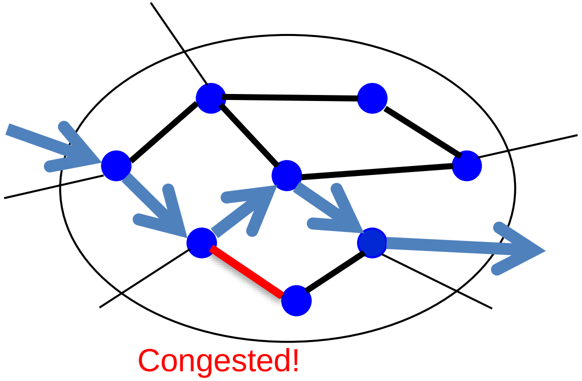
Background

- **Traffic Engineering (TE):** Configure routing to improve network performance
- **Metric:** Maximum Link Utilization (MLU) → $\frac{\text{Load}}{\text{Capacity}}$ of the most congested link



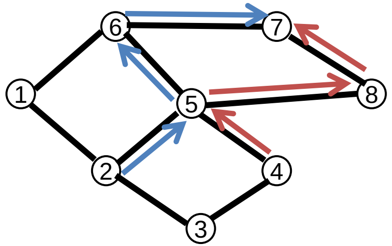
Background

- **Traffic Engineering (TE):** Configure routing to improve network performance
- **Metric:** Maximum Link Utilization (MLU) → $\frac{\text{Load}}{\text{Capacity}}$ of the most congested link



Flow-based or Destination-based Routing?

Flow-based Routing



Flow table at node 5

Match	Action
src = 2, dst = 7	Fwd to 6
src = 4, dst = 7	Fwd to 8

Two different sources can reach the same destination with different preconfigured paths

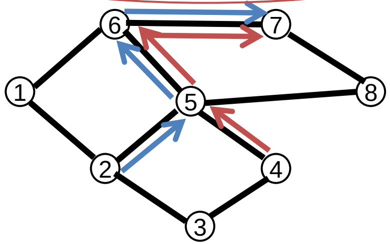
☺ Fine-grained traffic distribution control

P = # of IP routes

☹ Need to store $O(P^2)$ flow entries!

Scalability issue with limited TCAM resource

Destination-based Routing



Forwarding table at node 5

Destination	Next Hop
7	6

Paths from two different sources to same destination must coincide once they overlap

☺
- Lower forwarding complexity - $O(P)$ entries
- Widely implemented with simple RAMs

Centralized controller can be applied to update the entries when traffic changes

Motivation

However, traditional TE need to update <u>all entries</u> to improve network performance!

☹ Take considerable time → cannot react to traffic changes in a responsive manner

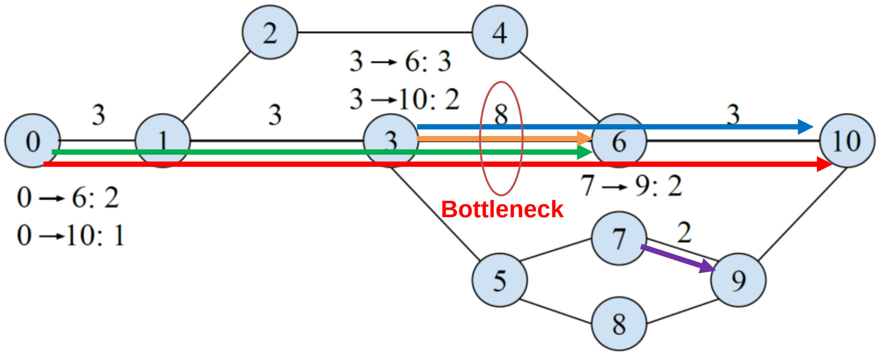
<u>Q: Can we mitigate routing update overhead?</u>

<u>A: Differentiate and route flows with a new traffic abstraction!</u>

(1) Only update some critical entries at some critical nodes to reroute traffic

(2) The remaining unaffected traffic are forwarded by ECMP

Equal-Cost Multipath (ECMP)



Motivation

However, traditional TE need to update <u>all entries</u> to improve network performance!

☹ Take considerable time → cannot react to traffic changes in a responsive manner

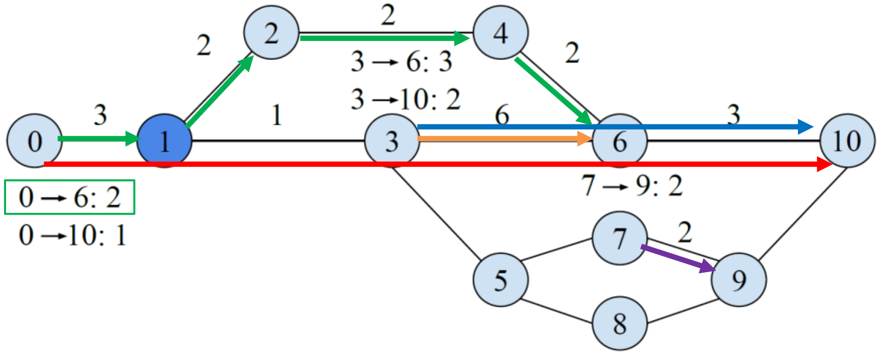
<u>Q: Can we mitigate routing update overhead?</u>

<u>A: Differentiate and route flows with a new traffic abstraction!</u>

(1) Only update some critical entries at some critical nodes to reroute traffic

(2) The remaining unaffected traffic are forwarded by ECMP

Update Critical Entries



Critical Entries

Forwarding table at node 1

Destination	Next Hop
6	2 (100%)

Motivation

However, traditional TE need to update <u>all entries</u> to improve network performance!

☹ Take considerable time → cannot react to traffic changes in a responsive manner

Key Problem: which pairs are 'critical'?
There are too many (node, dst) combinations!

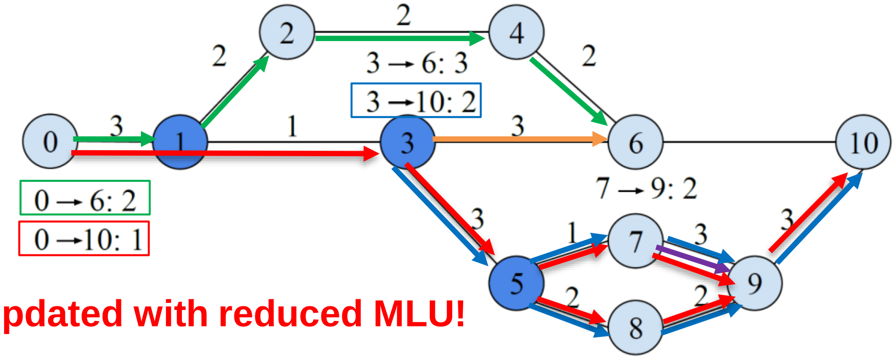
<u>Q: Can we mitigate routing update overhead?</u>

<u>A: Differentiate and route flows with a new traffic abstraction!</u>

(1) Only update some critical entries at some critical nodes to reroute traffic

(2) The remaining unaffected traffic are forwarded by ECMP

Update Critical Entries



Updated with reduced MLU!

Critical Entries

Forwarding table at node 1

Destination	Next Hop
6	2 (100%)

Forwarding table at node 3

Destination	Next Hop
10	5 (100%)

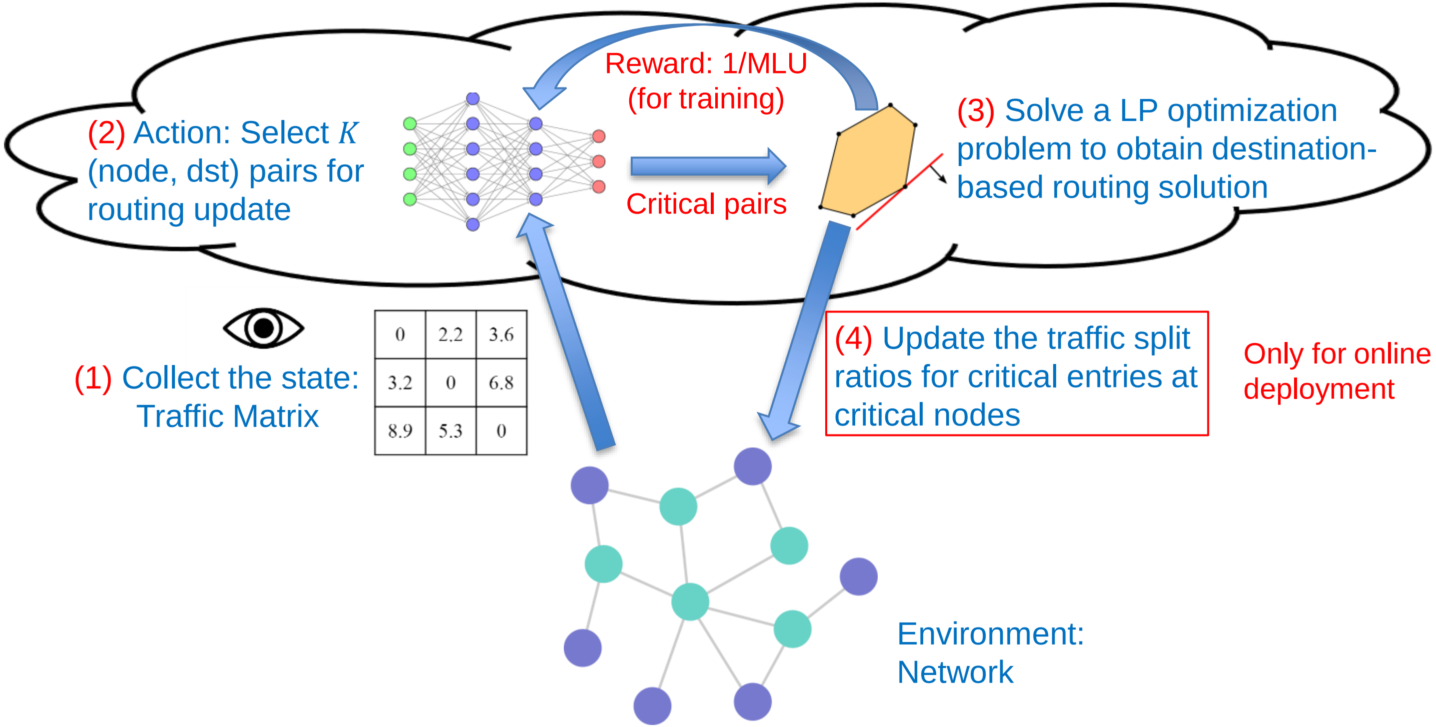
Forwarding table at node 5

Destination	Next Hop
10	7 (33.3%), 8 (66.6%)

SmartEntry: RL + LP combined approach

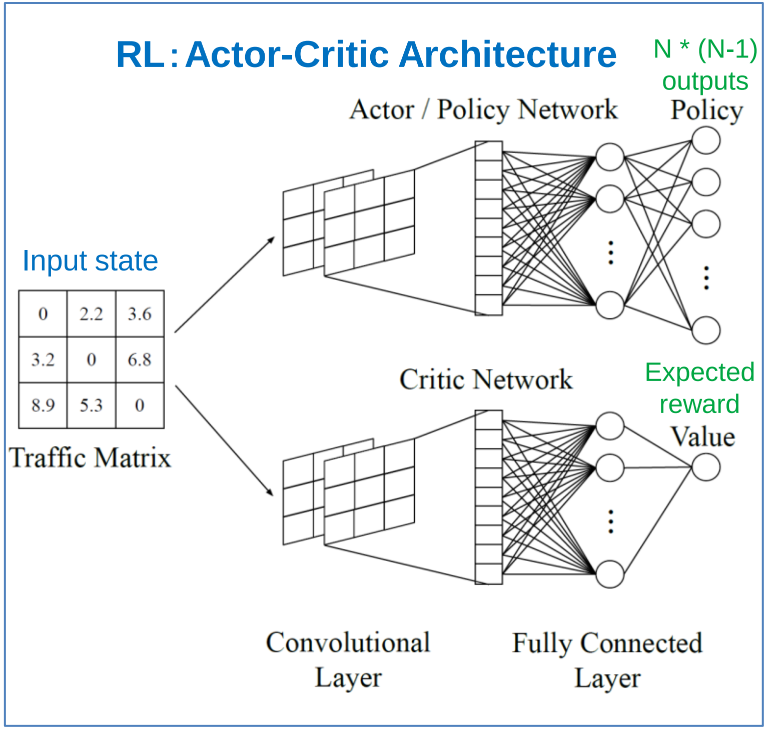
Idea: (1) Using **Reinforcement Learning (RL)** to smartly select critical pairs for routing update

(2) Solve a **Linear Programming (LP)** optimization problem to obtain destination-based routing solution



Why is RL + LP powerful?

- RL can model complex selection policies as neural networks to map "raw" observations to actions
- LP generates reward signal for RL to learn a better combination selection policy (minimize MLU)



Gradient update

Actions

LP

Produce reward signal

$$\text{minimize} \quad U_{max} + \epsilon \cdot \sum_{\langle i,j\rangle \in E} \sum_{d \in V} y^d_{i,j}$$

subject to

$$\sum_{d \in V} y^d_{i,j} = l_{i,j} \qquad i,j : \langle i,j\rangle \in E$$

$$l_{i,j} \le c_{i,j} \cdot U_{max} \qquad i,j : \langle i,j\rangle \in E$$

$$\sum_{k:\langle k,i\rangle \in E} y^d_{k,i} - \sum_{k:\langle i,k\rangle \in E} y^d_{i,k} = -t^{i,d} \qquad i,d : \tau^d_i \in \tau_K$$

$$y^d_{i,k} = \begin{cases} \dfrac{\sum_{n:\langle n,i\rangle \in E} y^d_{n,i} + t^{i,d}}{|\mathrm{ENH}^d_i|} & \text{if } k \in \mathrm{ENH}^d_i \\ 0 & \text{otherwise} \end{cases}$$

$$i,d : \tau^d_i \in \tau_{N*(N-1)-K},\ k : \langle i,k\rangle \in E$$

$$\sum_{k:\langle k,d\rangle \in E} y^d_{k,d} - \sum_{k:\langle d,k\rangle \in E} y^d_{d,k} = \sum_{s \in V, s \neq d} t^{s,d} \qquad d \in V$$

$$y^d_{i,j} \ge 0 \qquad d \in V,\ i,j : \langle i,j\rangle \in E$$

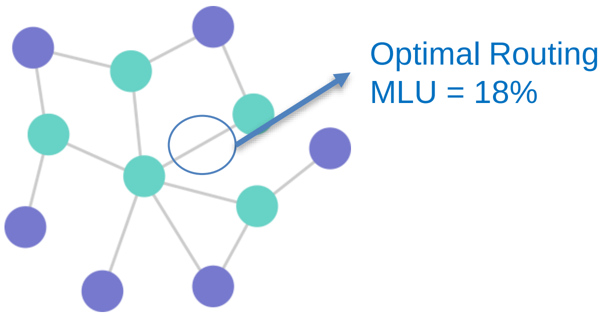
Experiment setup

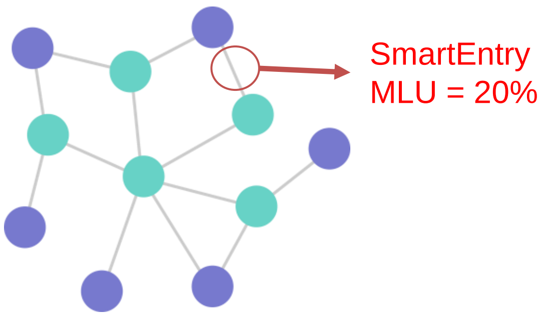
- We use 4 real networks to evaluate SmartEntry

Topology	Nodes	Directed Links
Abilene	12	30
GÉANT	23	74
EBONE	23	76
Tiscali	49	172

- Baseline Methods
  - ECMP: Distributes traffic evenly among available next hops along the shortest paths
  - Weighted ECMP: extends ECMP to allow weighted traffic splitting with shortest paths

- Evaluation Metric: **Performance Ratio (PR)**
  - Compare against optimal flow-based routing in terms of MLU
  - $PR = MLU_{optimal} / MLU_{SmartEntry}$

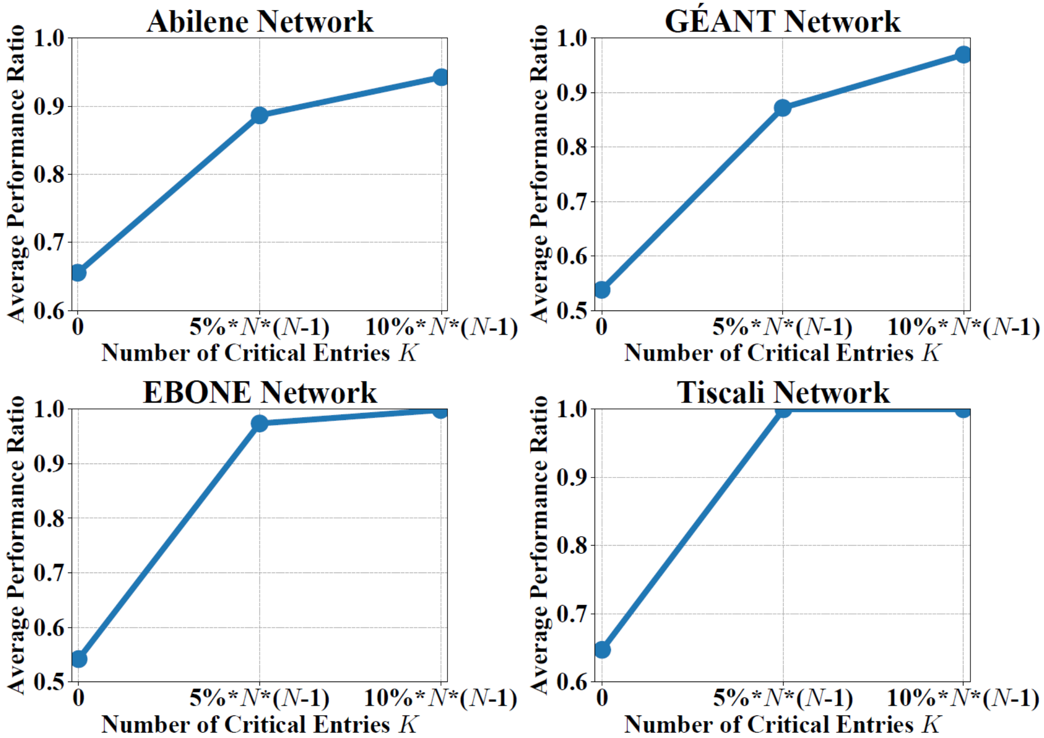




$$PR = \frac{18\%}{20\%} = 0.9$$

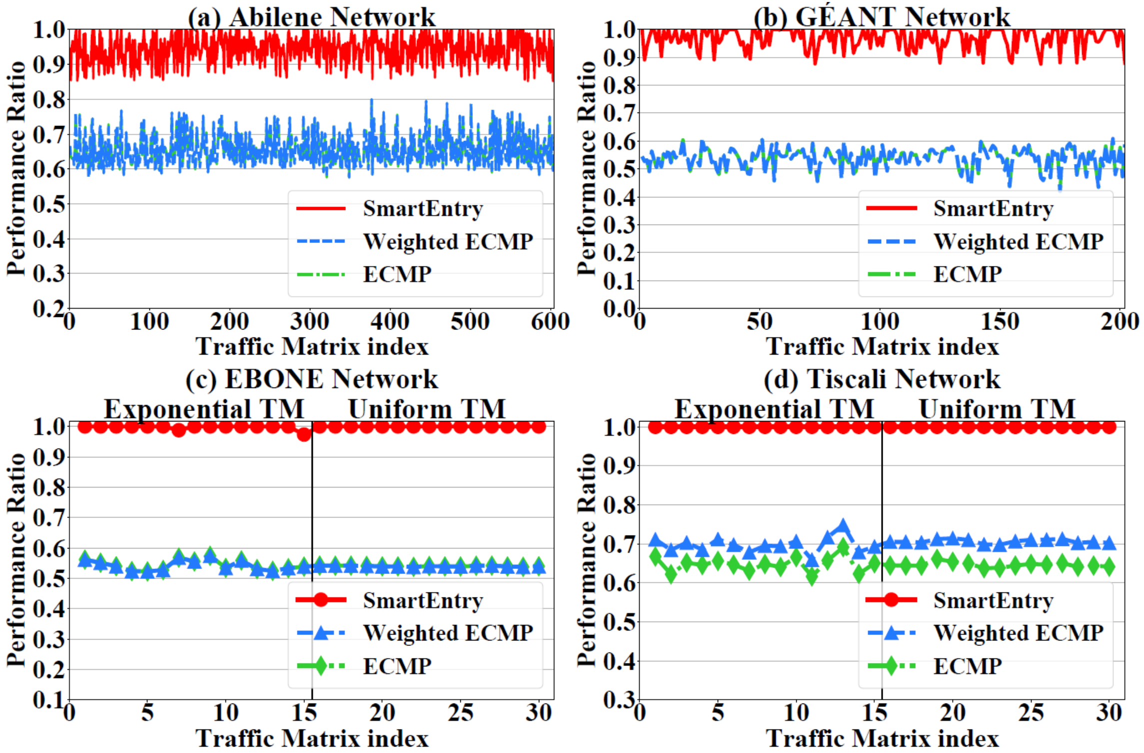
Number of critical entries

- SmartEntry achieves near-optimal performance with only 10% entries updated



Comparison in different networks

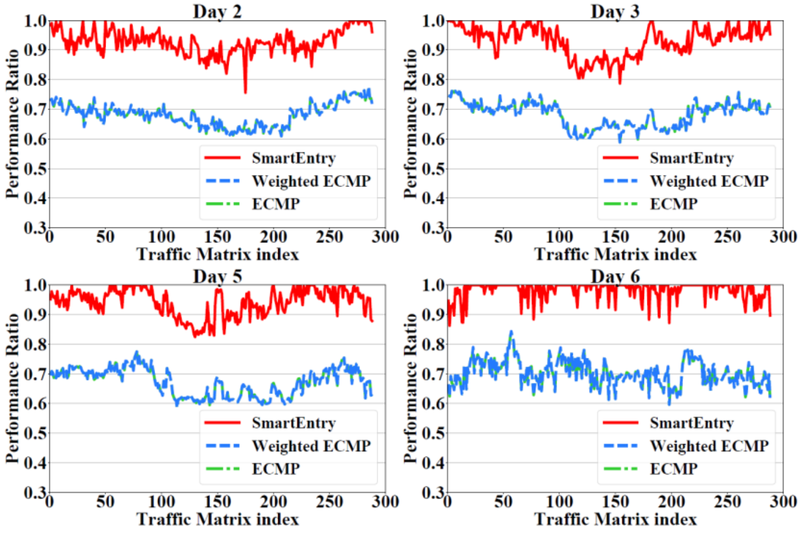
➢ SmartEntry performs consistently well on real and synthesized traffic matrices



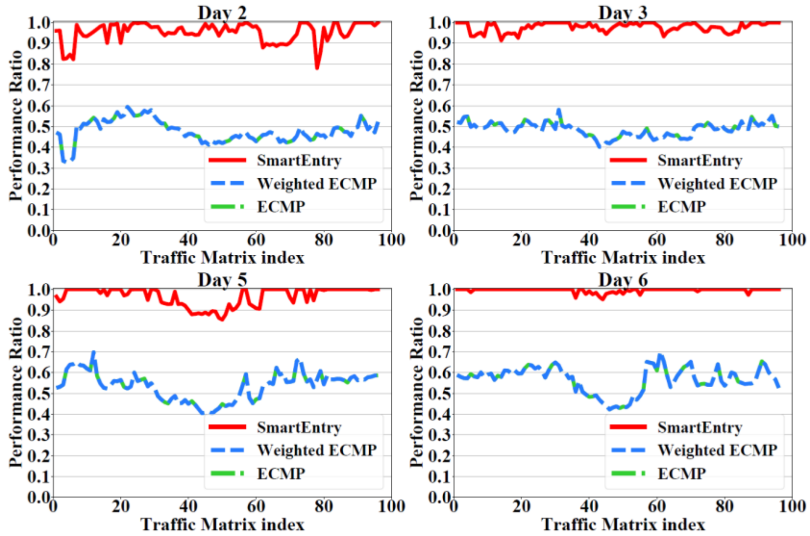
Generalization test

- ➢ Training on week 1, test on week 2
- ➢ SmartEntry generalizes well to unseen traffics

Abilene Network



GÉANT Network



Conclusion

- With an objective of minimizing maximum link utilization in a network and mitigating routing update overhead, we proposed SmartEntry, a scheme that learns a combination selection policy automatically using reinforcement learning, without any domain specific rule-based heuristic.

- SmartEntry smartly selects a combination of K node-destination pairs for each given traffic matrix and reroutes the selected traffic to achieve load balancing of the network by solving a rerouting optimization problem.

- Extensive evaluations show that SmartEntry achieves near-optimal performance and generalizes well to traffic matrices for which it was not explicitly trained.