Link-Based Fine Granularity Flow Migration in SDNs to Reduce Packet Loss

Yang Chen and Jie Wu
Center for Networked Computing
Temple University, USA

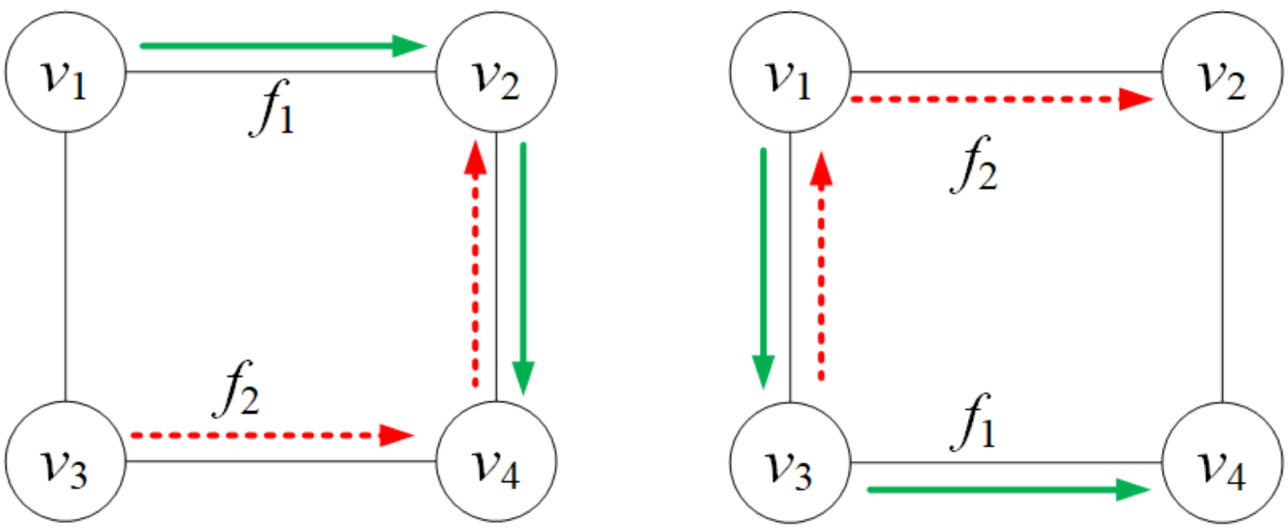
Road Map

- Introduction
- Model
- Link-based Flow Migration
- Simulation
- Conclusion

1. Introduction

- Flow migration in SDN: Upon traffic changes
- Challenges: Asynchronous rule updates -> congestion -> deadlocks
- Current update methods: path-based

Initial State → Final State



Unit link capacity
Unit flow demand
Flows are unsplittable

The initial path of f_1 overlaps the final path of f_2, and vice versa

- In this paper, we migrate flows in a finer granularity of links.

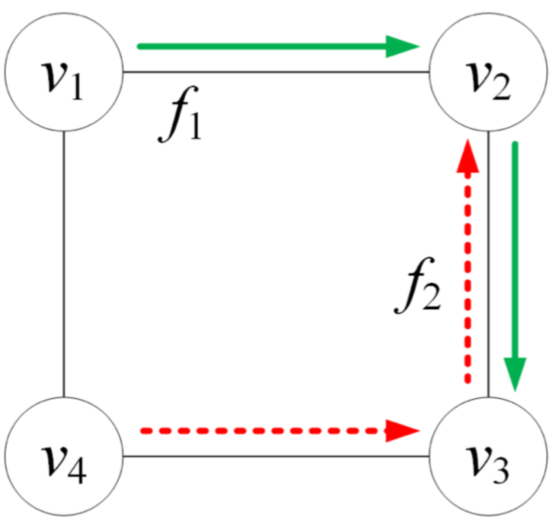
Example (Cont'd)

- Link-based update scheme:

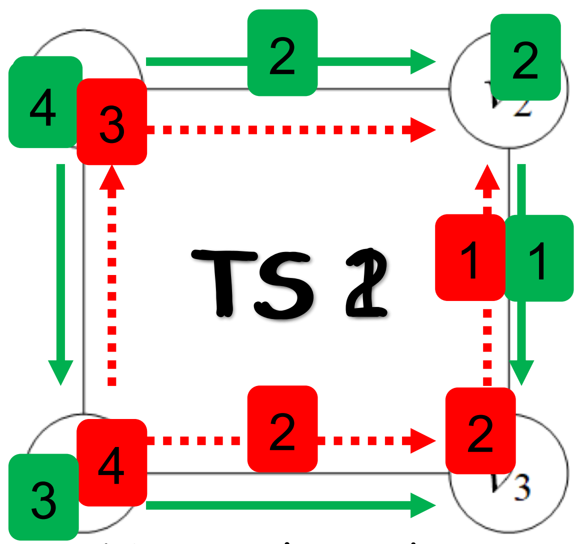
Time Step (TS): the time to assign and release one link resource

- Example:

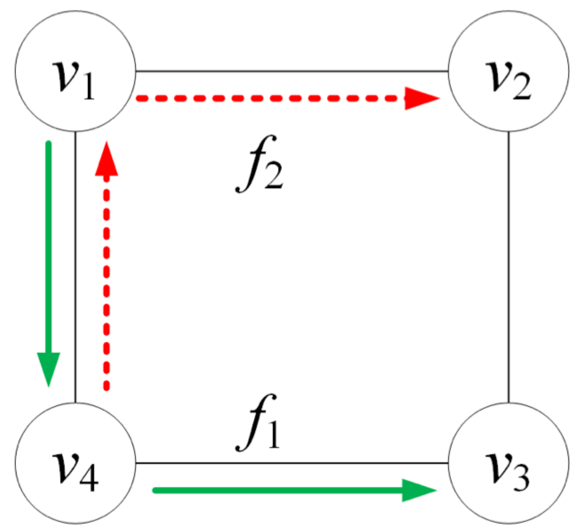
1. TS1: f_1 frees e_{12} and occupies e_{14} ; f_2 frees e_{43} and occupies e_{41}
2. TS2: f_1 frees e_{23} and occupies e_{43} ; f_2 frees e_{32} and occupies e_{12}



Initial State



Network Update



Final State

2. Model

- Model

  A network with capacitated links and a set of flows with demands

- Objective

  Migrate flows from initial to final paths consistently

- Migration constraint:

  Consistent: no congestion and packet loss

- Update network in the granularity of link:

  Single link request and assignment in each time step

- Key observation

  Link-based scheduling causes less deadlocks

Complexity of the problem

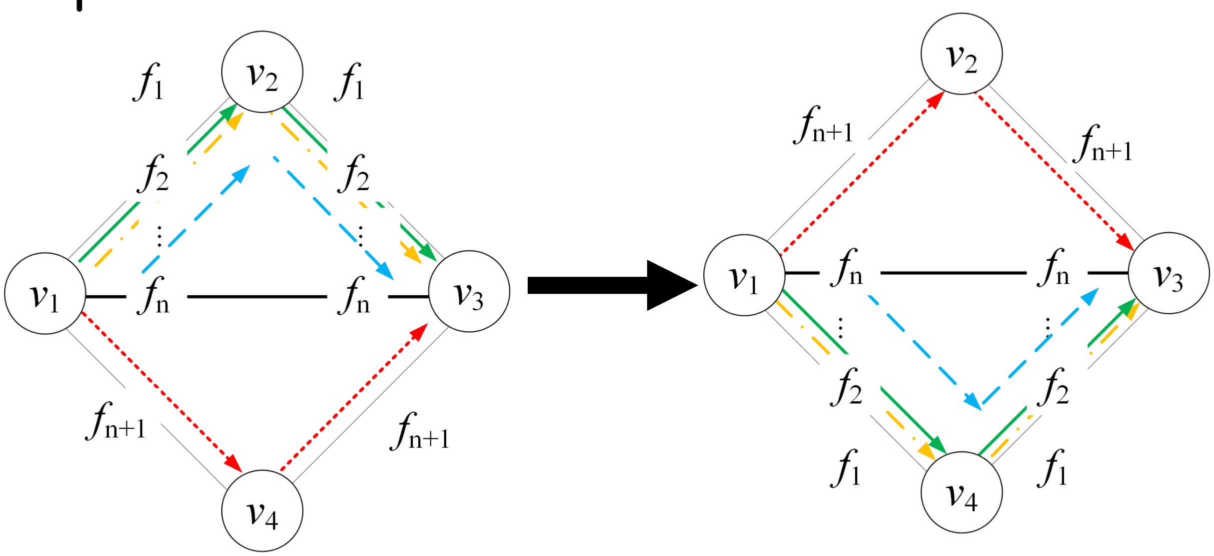
Theorem 1: Checking feasibility of a consistent migration is NP-hard.

Proof ideas: using a special update case

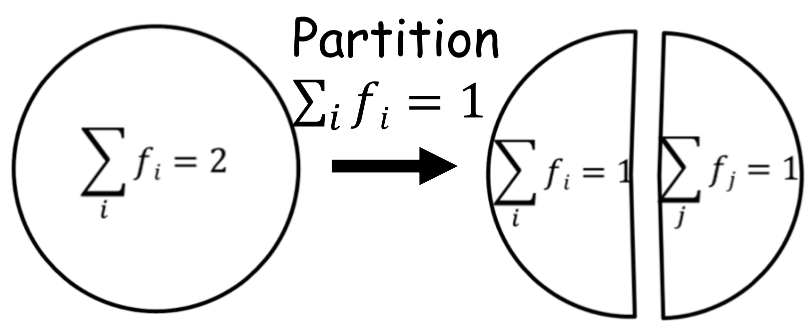
Link's capacity: 2

Flows' demands: $f_1 + f_2 + \ldots + f_n = 2$; $f_{n+1} = 1$

Reduction from the partition problem: whether $f_1, f_2, \ldots, f_n$ can be partitioned into two sets with the same sum of demands.



Initial State   Final State



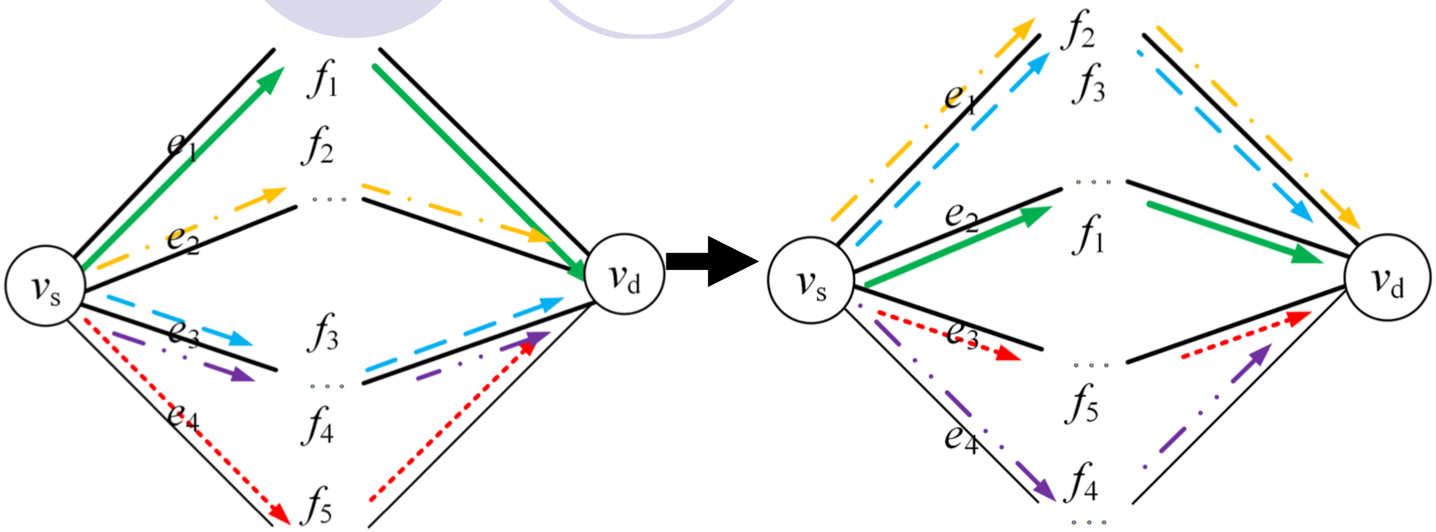
1. Move one set down
2. Move f_{n+1} up
3. Move another set down

Concepts

- Resource Dependency Graph (RDG)
  1. flows & links -> nodes
  2. link' requests & assignments -> directed edges
- Deadlock: all links impossible to satisfy any request inside it
- Stuck State: remaining capacities unable to satisfy any request
- Knot: a set where each node only can reach all nodes in the set

Flow-link relationship scheme

An illustrating example



Initial State

Final State

Demand:

f_1=2, others=1

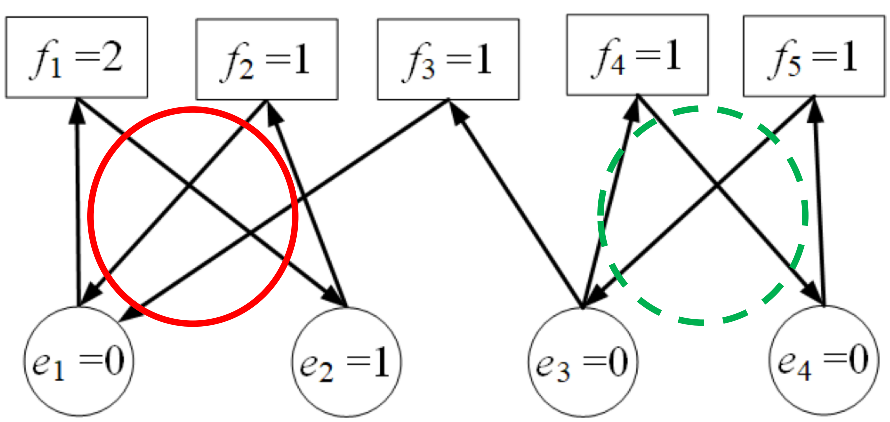
Capacity:

e_4 =1,others=2

Two deadlocks:

1. {e_1, f_1, e_2, f_2}
2. {e_3, f_4, e_4, f_5}

A knot: {e_1, f_1, e_2, f_2}



Stuck RDG

3. LInk-based Flow MIGration (LifMig)

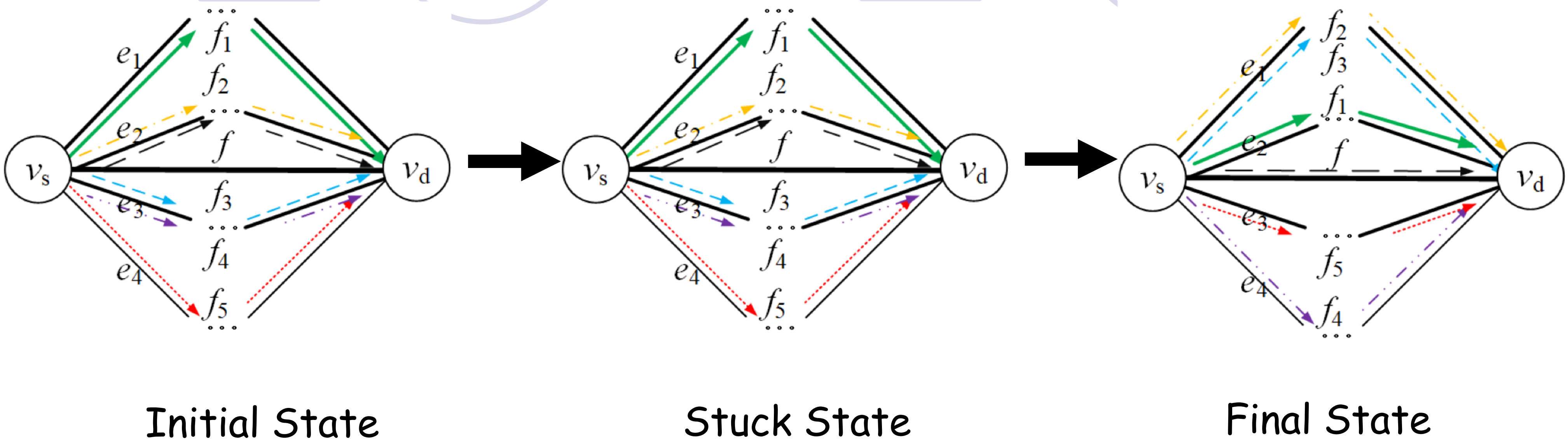
- Algorithm 1: LifMig
- While migration not finished:

1. Construct RDG in the current time step;
2. Remaining resource allocation using Algorithm 2;
3. Deadlock detection;
4. Detected-> resolve by spare paths (ISPA'17) ;
5. Still stuck-> rate limiting flows;

- Algorithm 2: Remaining Resource Allocation

1. For each link with remaining capacity:
2. Find flows with demand less than the remaining capacity;
3. Assign to flows in order of benefit (demand × #link's waiting requests);
4. Update RDG;

An illustrating example



Demand: f_1=2, others=1

Capacity: e_4=1, e_{sd} =3, others=2

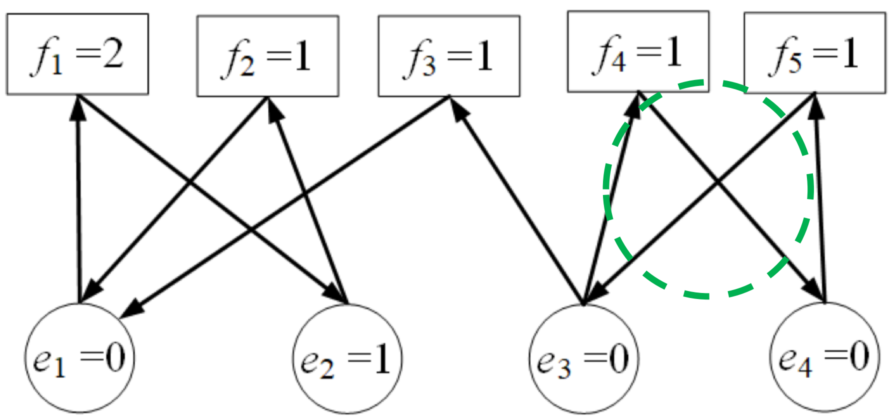
1. Move f -> stuck state
2. Move f_1 to e_{sd} (spare path)
3. Move f_2,f_3,f_4,f_5
4. Move f_1

Deadlock Detection in RDG

Theorem 2: A cycle in the RDG is a necessary condition for deadlocks.

Observation:

RDG with no cycles -> use the topological order to update flows



A cycle is necessarily a deadlock.

Deadlock Detection in RDG

Theorem 3: In a stuck RDG, a knot is a sufficient condition for the existence of a deadlock.

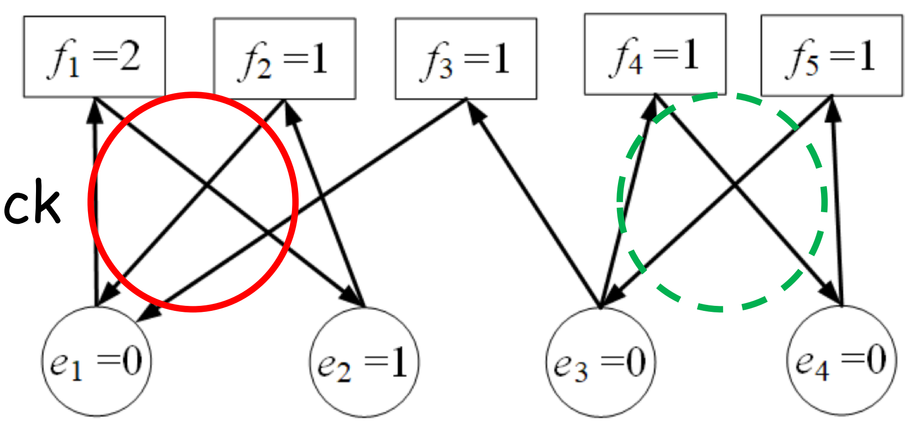
Proof:

no assignment to out-knot nodes in stuck RDG

-> release resources only by intra-knot flows

-> intra-knot flows also wait intra-knot link resources

knot & deadlock



deadlock & not knot

A knot is sufficiently a deadlock.

A knot is not necessarily a deadlock.

Deadlock Detection in RDG

Theorem 4: In a stuck RDG with unit demands for all flows, a knot is a necessary and sufficient condition for the existence of a deadlock.

Proof ideas:

1. Sufficiency by Theorem 3
2. Necessity: using contradiction

   If no knots,

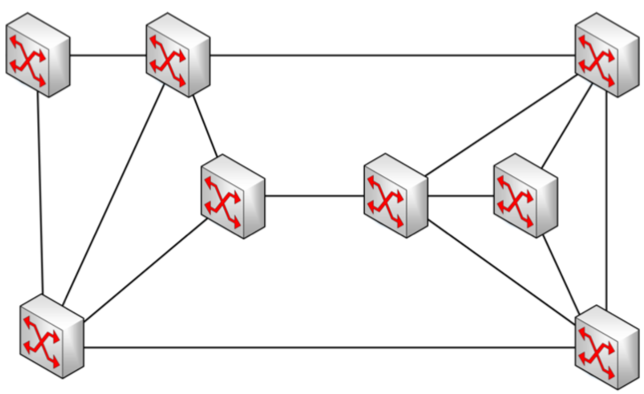
   -> a path from any requesting flow to the occupying flow exists

   -> not stuck

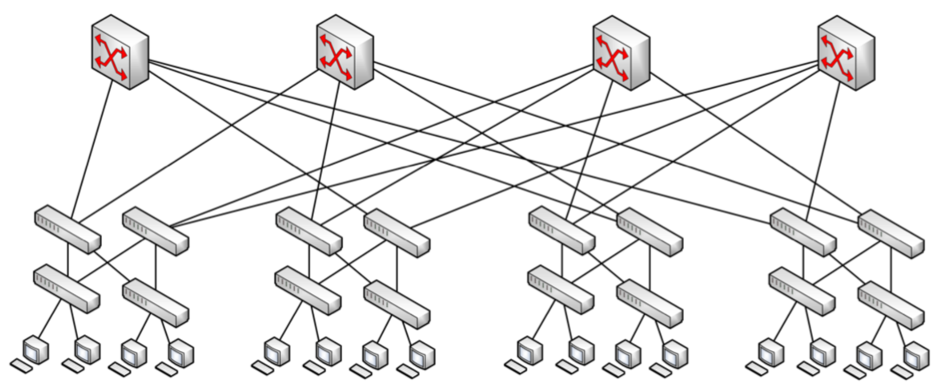
   -> violate assumption

4. Simulation

- Two comparison algorithms:
  1. **Dionysus**: migrate flows in a topological order and opportunistically rates limit flows as zero for resolving deadlocks (SIGCOMM'14)
  2. **NUSL**: a path-based consistent update strategy and solve deadlocks by spare paths (ISPA'17)
- Network topologies



WAN network



Fat-tree network

Settings and Measurements

- Settings
  1. WAN topology (link capacity: 1 Gbps)

Traffic load	0.3	0.4	0.5	0.6	0.7	0.8
Flow number	1023	1548	1899	2302	2637	3110

  2. Fat-tree topology (link capacity: 1 Gbps)

Traffic load	0.3	0.4	0.5	0.6	0.7	0.8
Flow number	3608	4139	5302	6327	7122	8423

- Measurement
  1. Traffic loss ratio

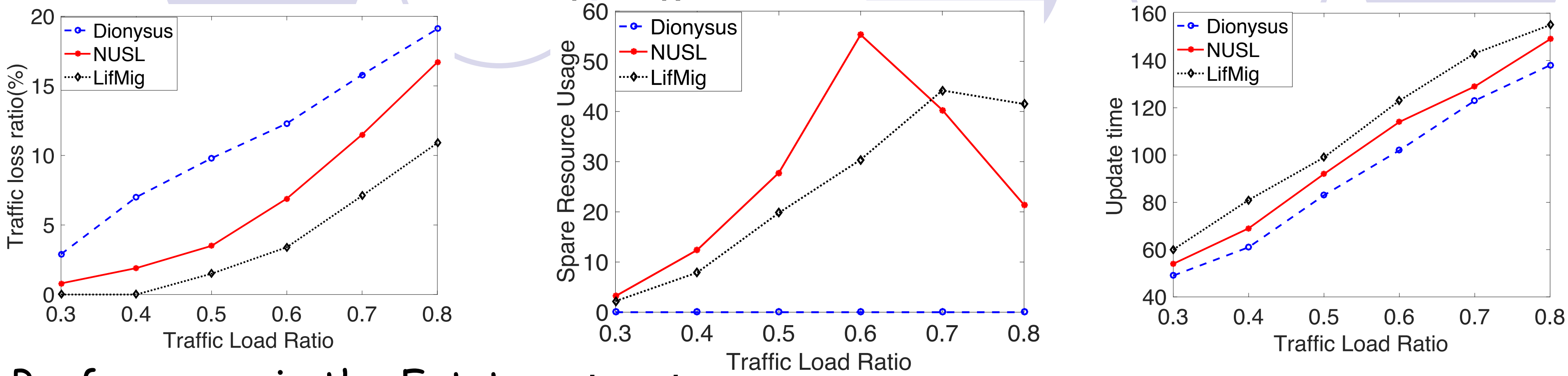
     the ratio of lost packets against all packets
  2. Spare resource usage

     bandwidth resource as spare paths
  3. Update time

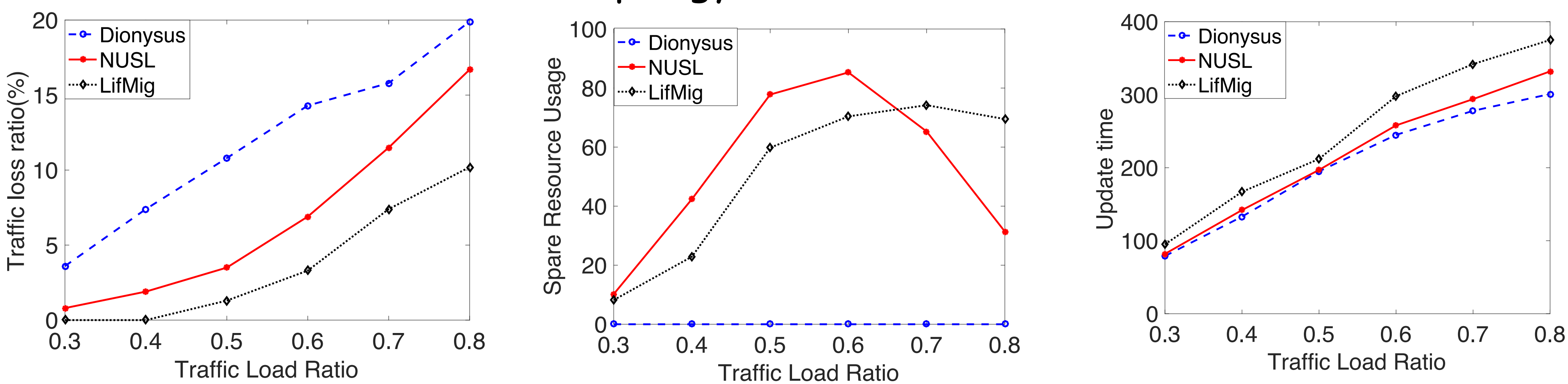
     the number of time steps during the update

Simulation Results

Performance in the WAN topology



Performance in the Fat-tree topology



- LifMig always has the least traffic loss
- LifMig uses fewer spare resources than NUSL
- LifMig takes about 17% (WAN) and 25% (Fat-tree) more steps than NUSL

5. Conclusion:

- A finer network update granularity: links
- Key observation:
  - Link-based scheduling causes less deadlocks
- NP-hardness:
  - Check the update feasibility
- Efficient network update scheme
- Deadlock existence conditions

Thank you!

Q & A