On Polynomial-Time Congestion-Free Software-Defined Network Updates

Saeed Akhoondian Amiri (MPI, Saarland, Germany)

Szymon Dudycz (University of Wroclaw, Poland)

Mahmoud Parham and Stefan Schmid (University of Vienna, Austria)

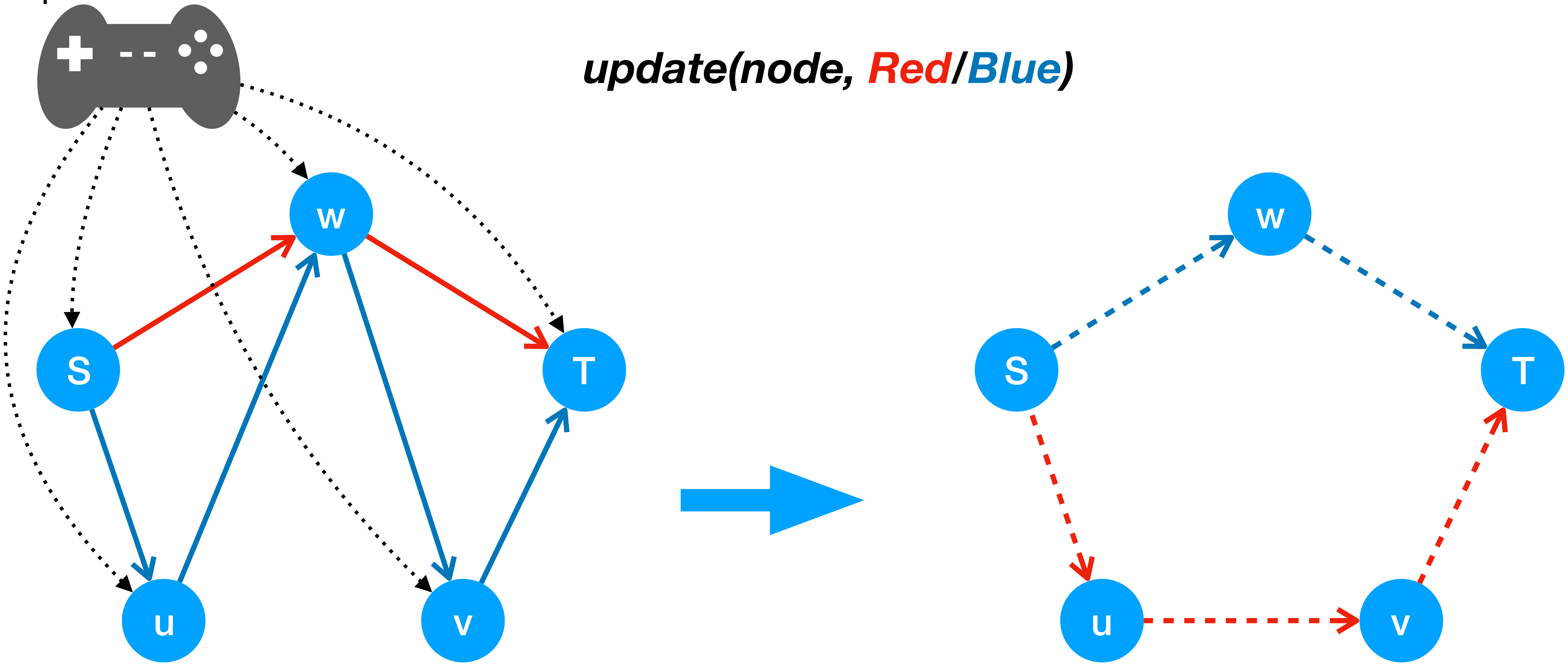
Sebastian Wiederrecht (TU Berlin, Germany)

IFIP NETWORKING 2019, May 21, 2019 – Warsaw, Poland

Why reroute?

- many reasons, including security and policy changes
- traffic engineering, optimizations, demand changes etc.
- rerouting involves distribution of new (forwarding) rules
- while maintaining certain consistency properties
- in a SDN, rules are distributed by a <u>controller</u>

SDN controller issues update commands



old routes

new routes

Task: reroute red and blue flows to their new route

Solid: old route

Dashed: new route

Let’s reroute!

update(u, Blue)

update(S, Blue)

update(w, Red)

update(S, Red)

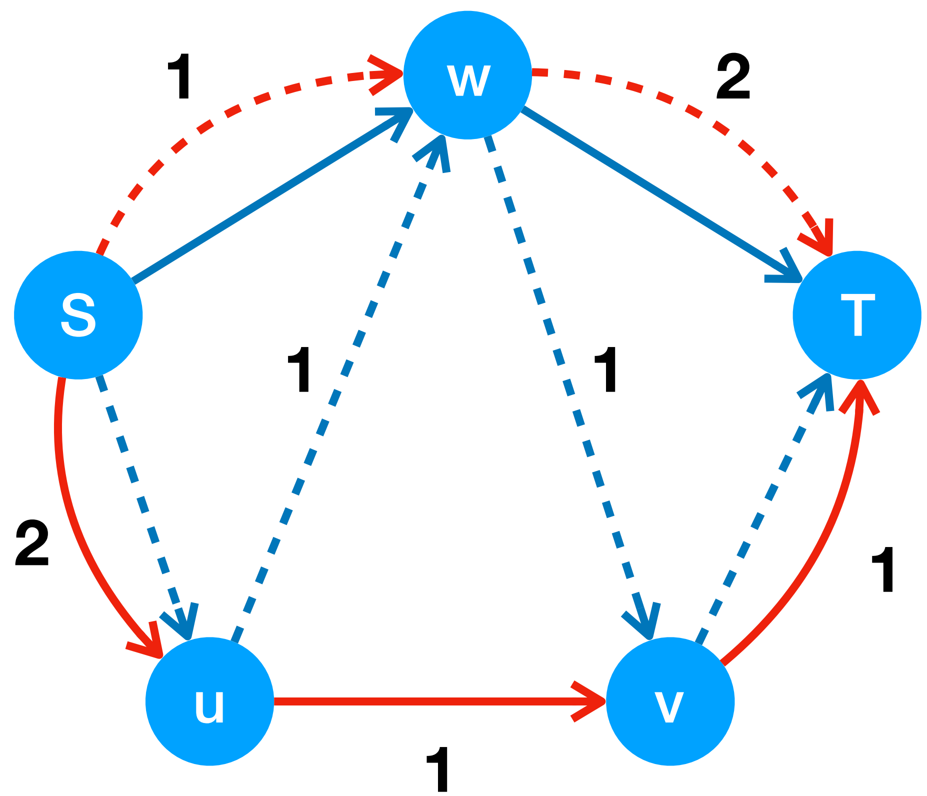
update(u, Red)

update(v, Red)

update(v, Blue)

update(w, Blue)

update(node, Red/Blue)



Task: reroute red and blue flows to their new route

Solid: old route

Dashed: new route

1-update(w, Red)

update(S, Blue)

update(w, Red)

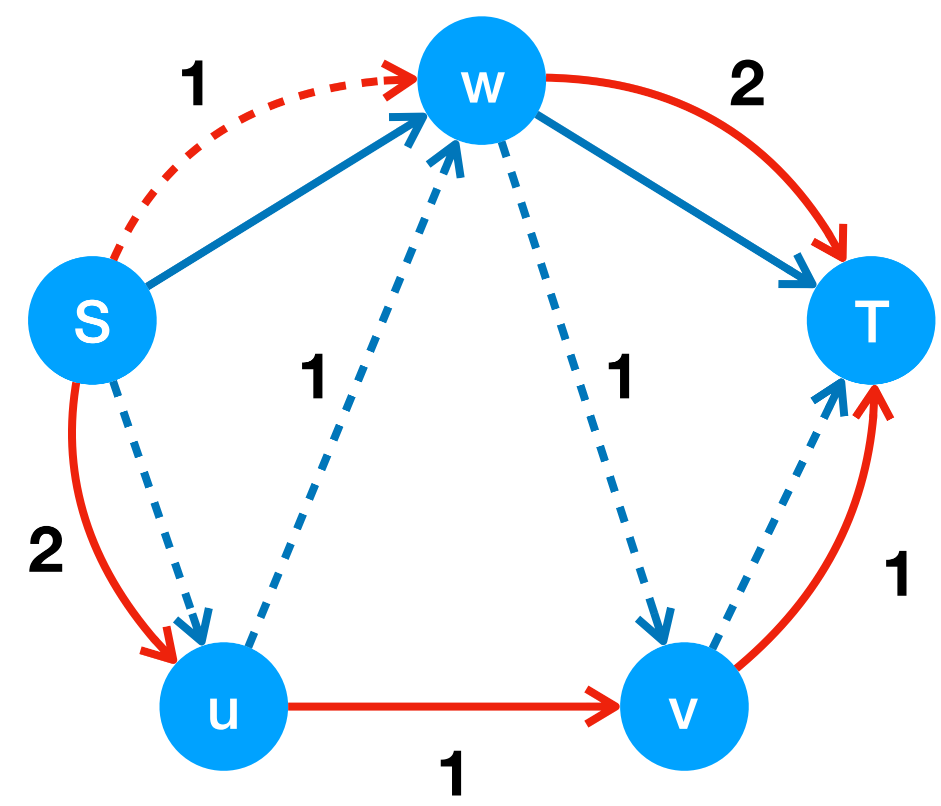
update(S, Red)

update(u, Red)

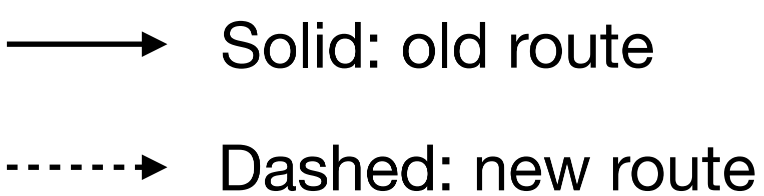
update(v, Red)

update(v, Blue)

update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(w, Red)

2-update(S, Red)

update(w, Red)

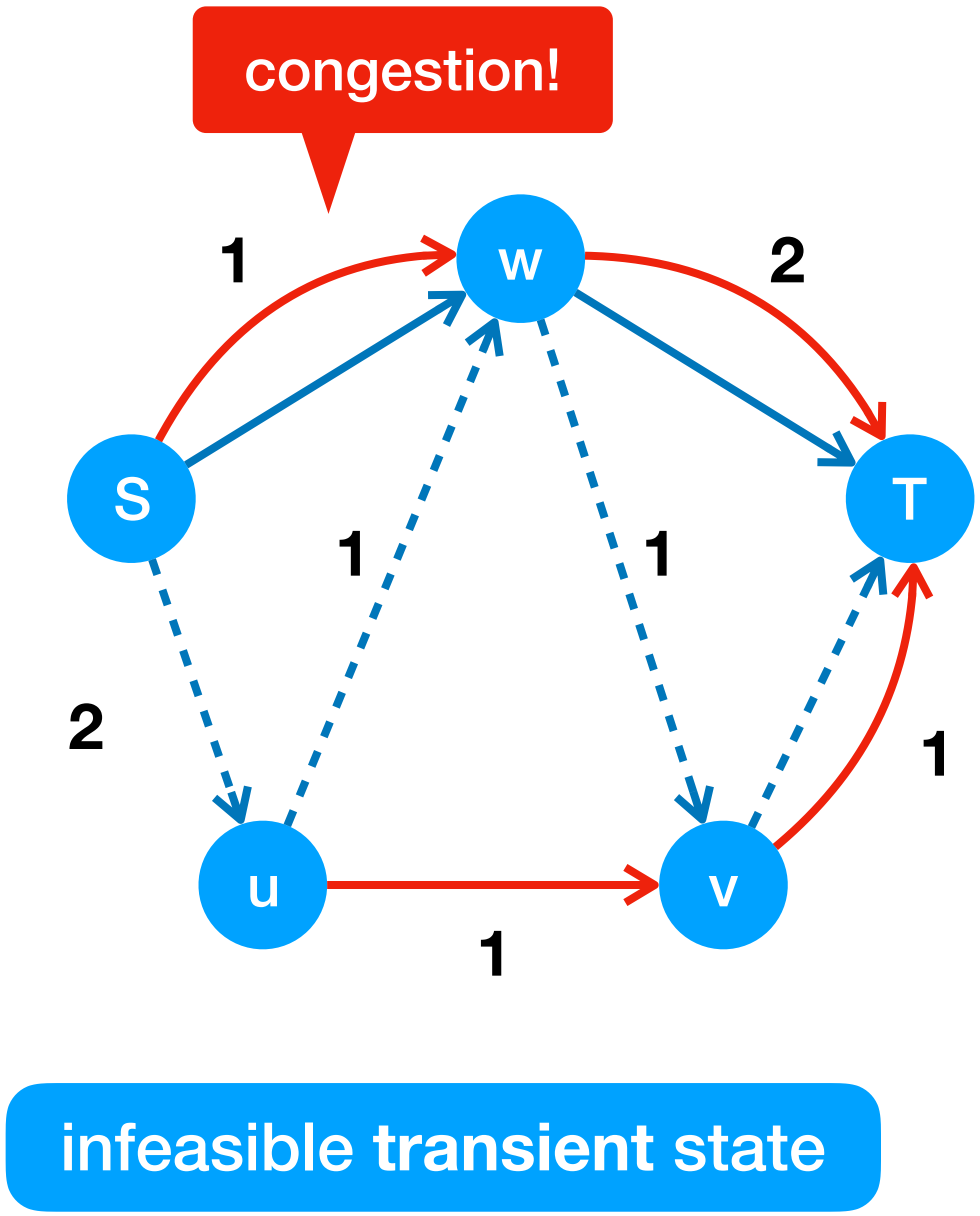
update(S, Red)

update(u, Red)

update(v, Red)

update(v, Blue)

update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!

Solid: old route

Dashed: new route

Let's try again!

1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

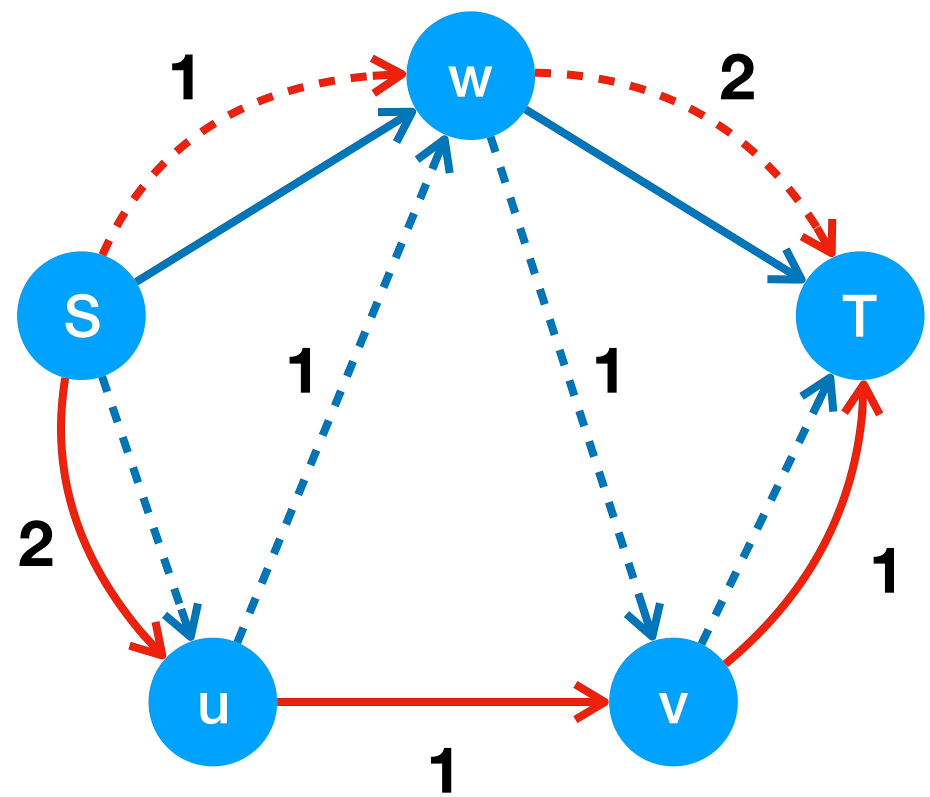
4-update(S, Red)

5-update(u, Red)

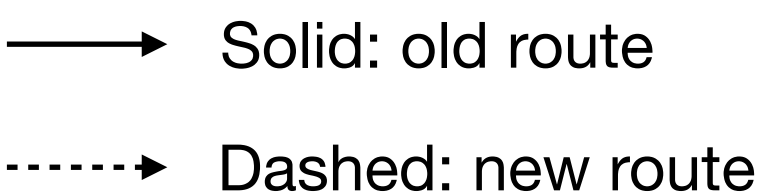
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



one update per round

1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

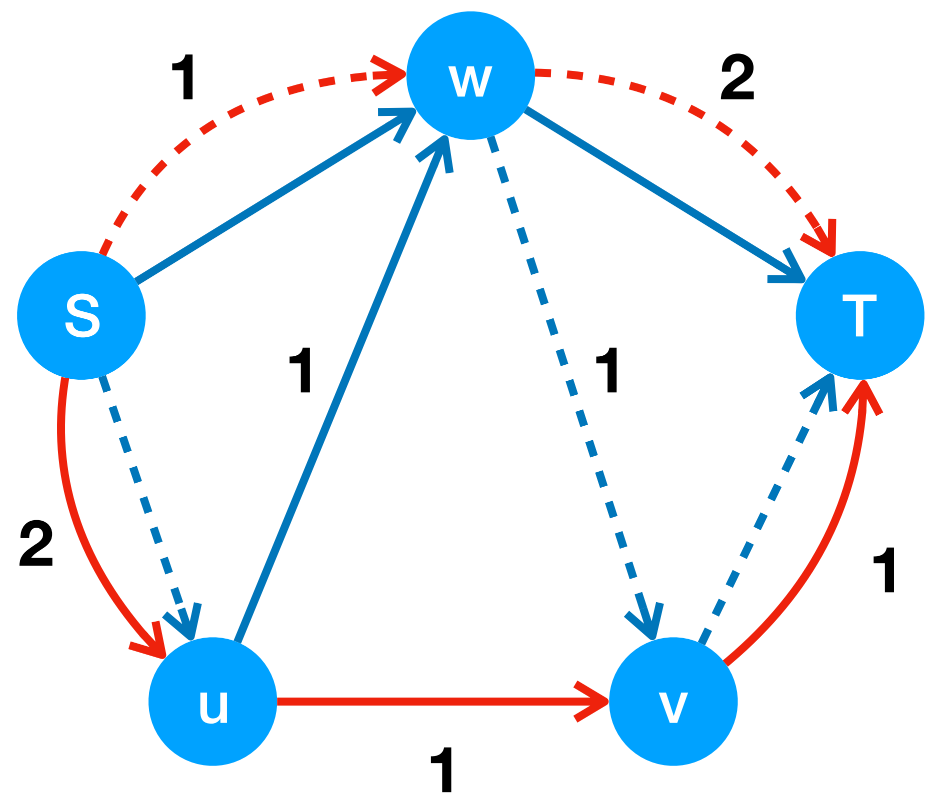
4-update(S, Red)

5-update(u, Red)

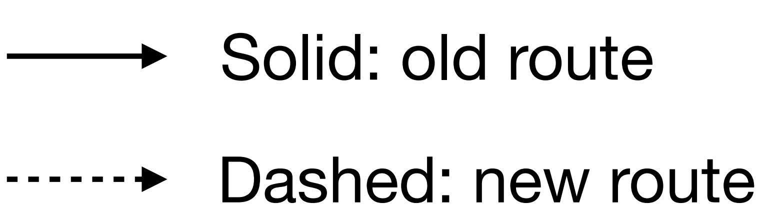
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

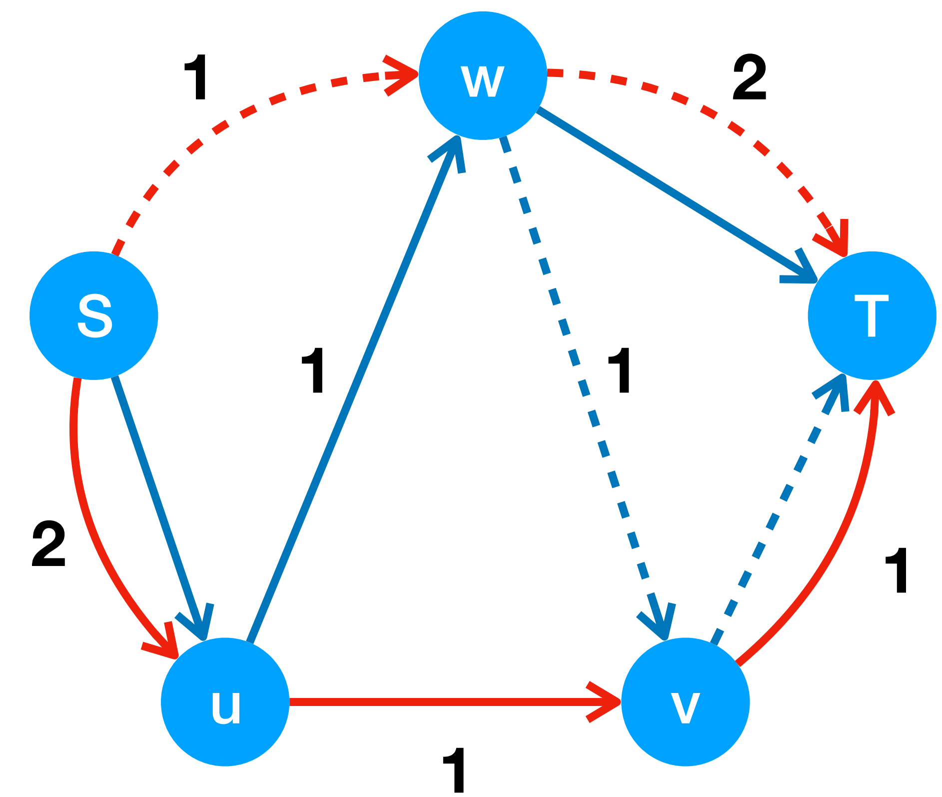
4-update(S, Red)

5-update(u, Red)

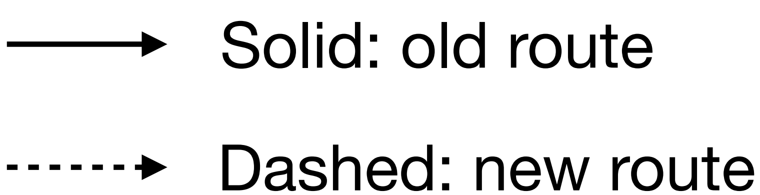
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

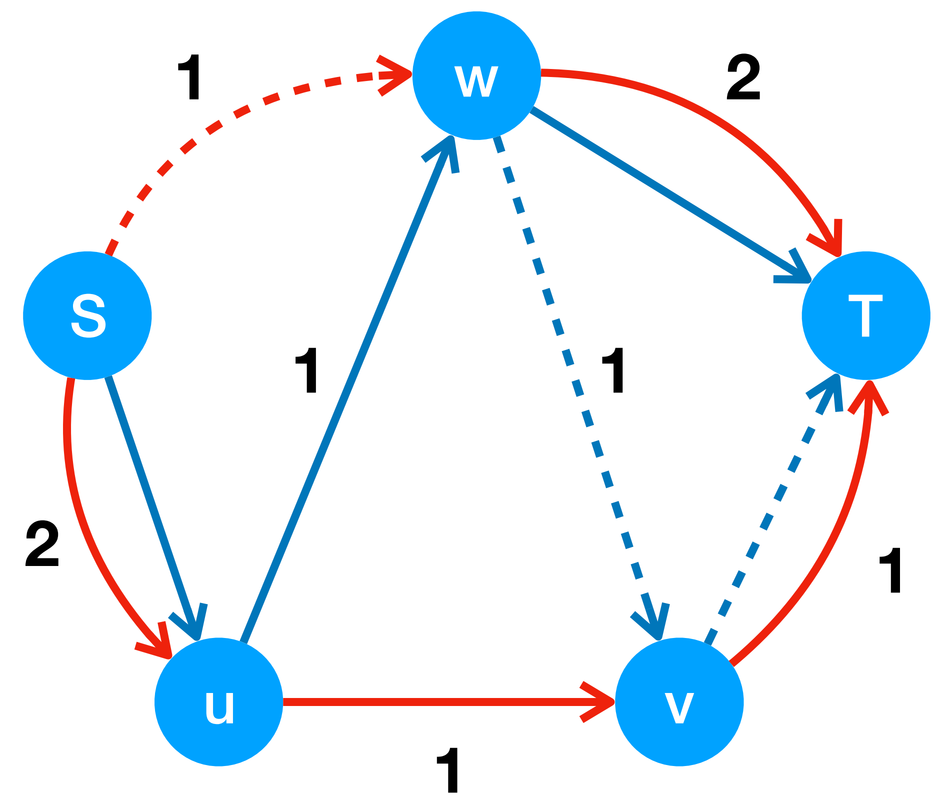
4-update(S, Red)

5-update(u, Red)

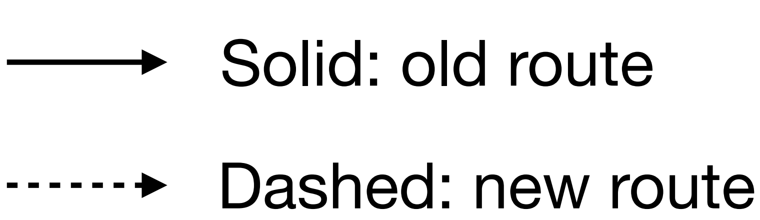
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

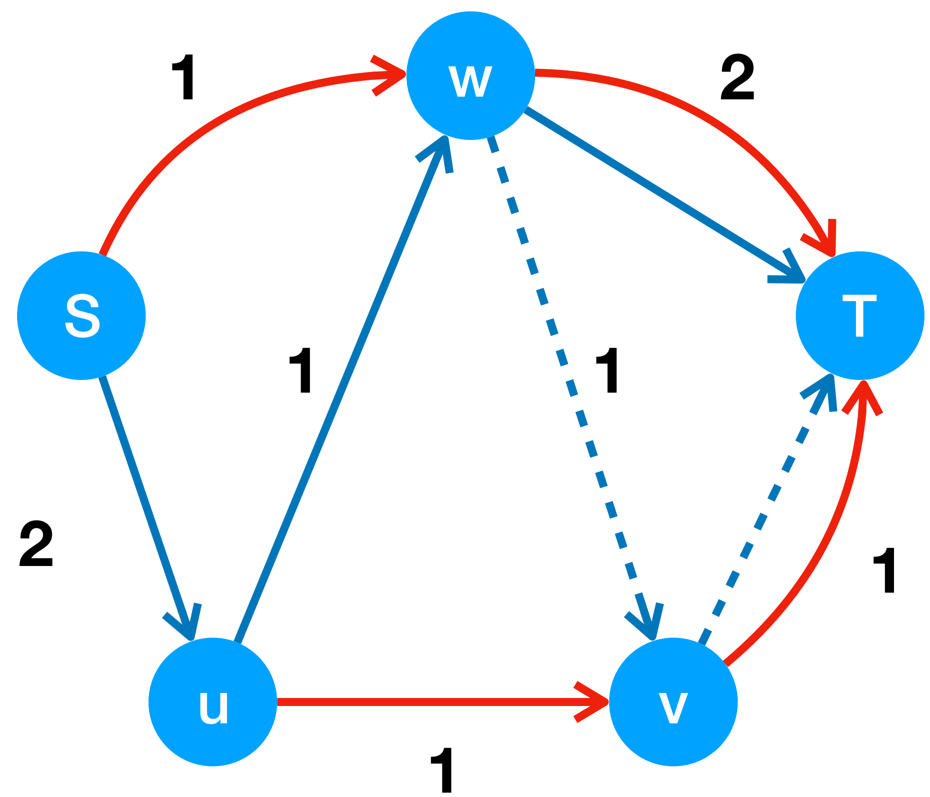
4-update(S, Red)

5-update(u, Red)

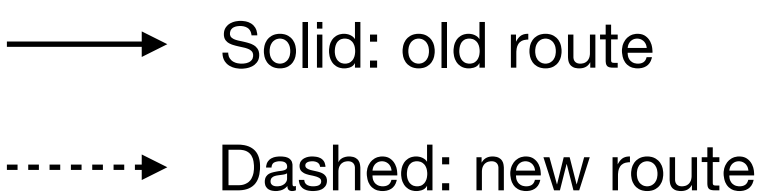
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

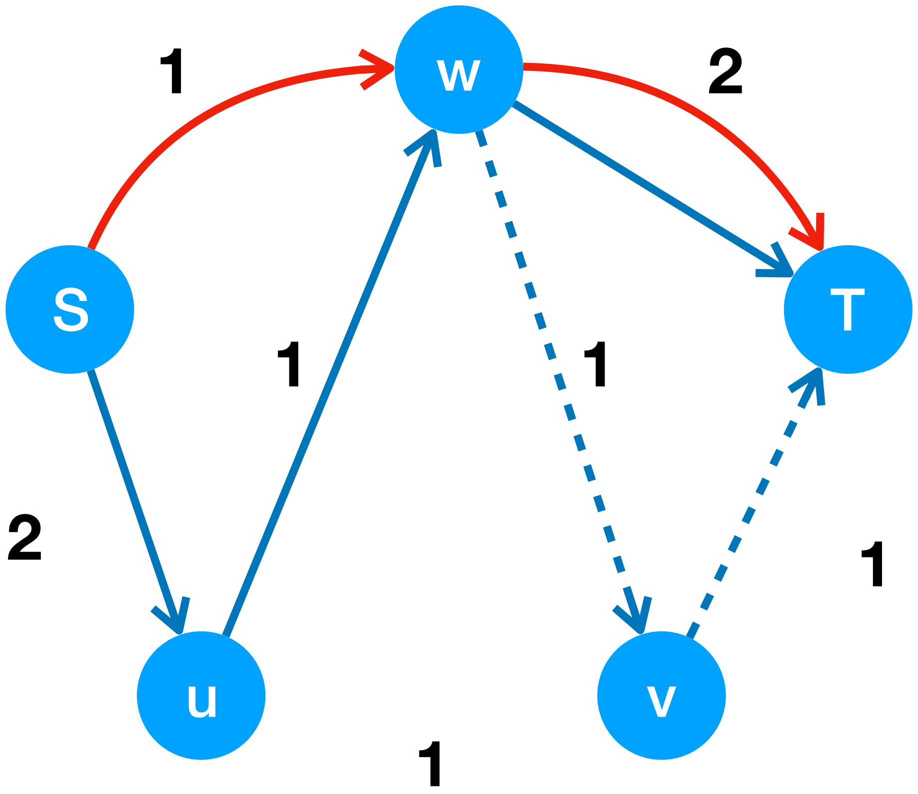
4-update(S, Red)

5-update(u, Red)

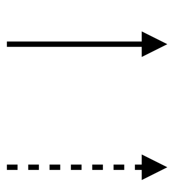
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

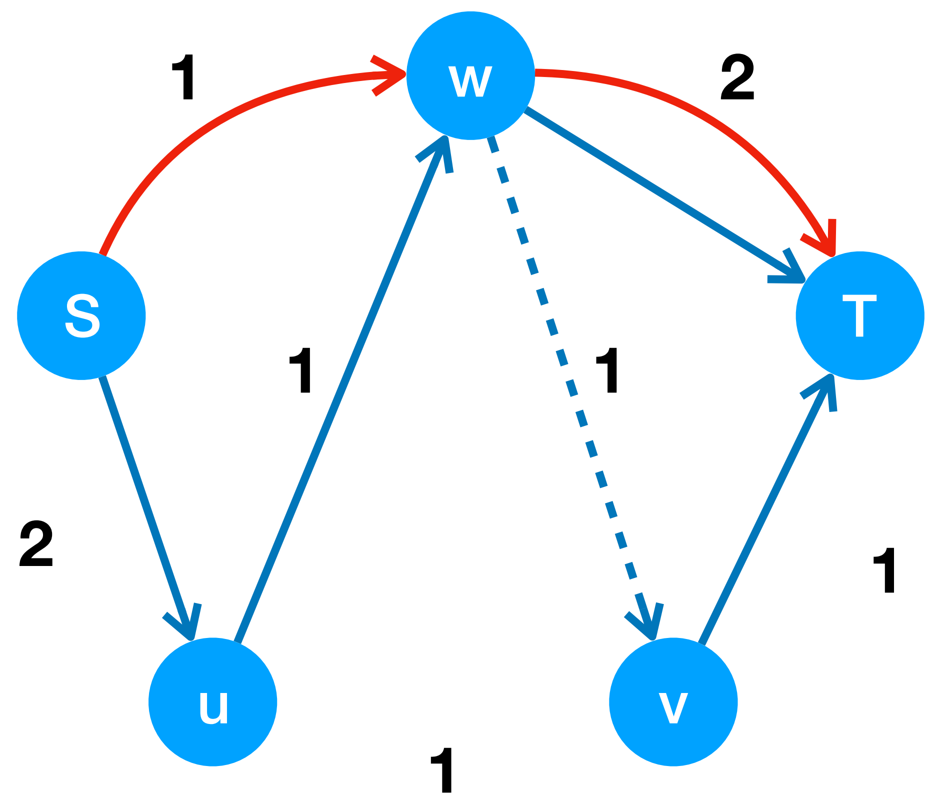
4-update(S, Red)

5-update(u, Red)

6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!

Solid: old route

Dashed: new route

1-update(u, Blue)

2-update(S, Blue)

3-update(w, Red)

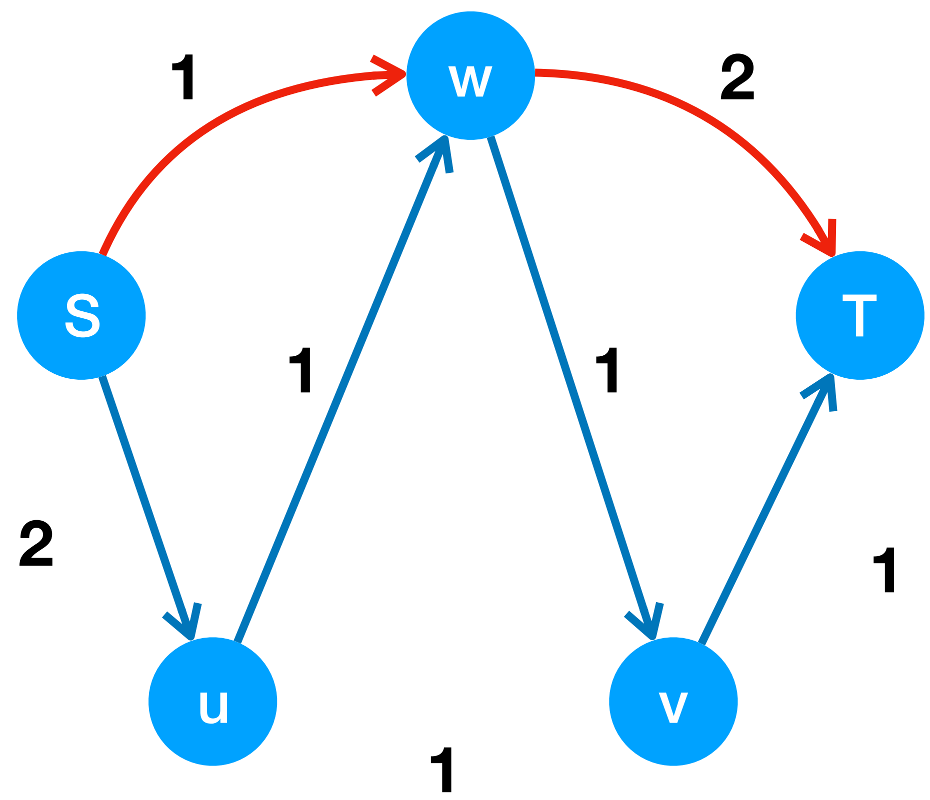
4-update(S, Red)

5-update(u, Red)

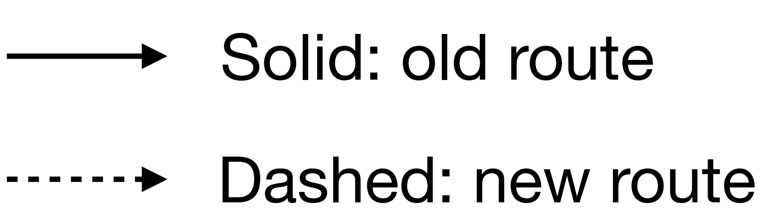
6-update(v, Red)

7-update(v, Blue)

8-update(w, Blue)



Rerouting red and blue flows to their new route, **congestion-free**!



Observations

- 8 updates, 8 rounds in this example
- Multiple updates could be scheduled for a same round
- Updates in a same round finish in arbitrary order ⇒ **transient states**
- All transient states must respect capacity constraints
- The example admits a feasible schedule in only **4** rounds

Feasible Transient States

1-update(u, Blue)

2-update(S, Blue)

2-update(v, Blue)

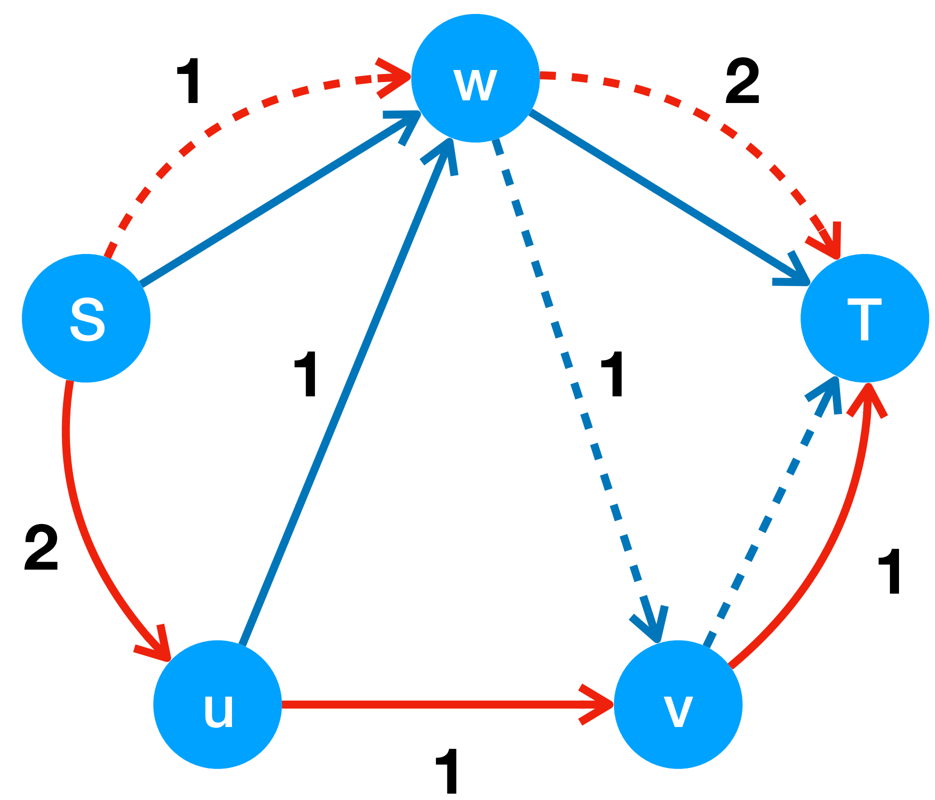
2-update(w, Red)

3-update(S, Red)

4-update(u, Red)

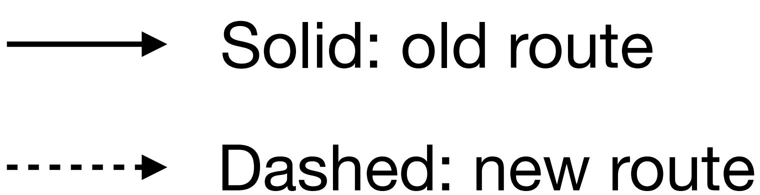
4-update(v, Red)

4-update(w, Blue)



2^3=8 possible transient states for round 2

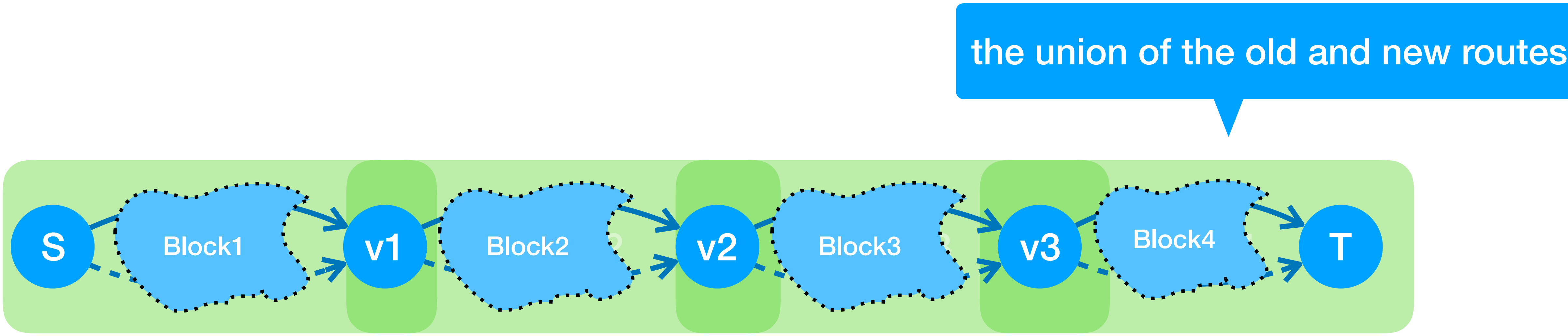
Rerouting red and blue flows to their new route, **congestion-free**!



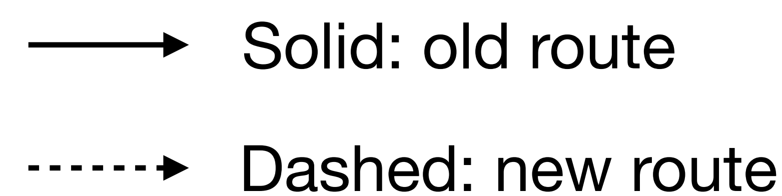
Problem Definition

- **Input**: two flow pairs *(old,new),* unit demands, unsplittable.
- **Task**: reroute each flow, from its **old** route to its **new** route, **consistently**.
- Consistent: Loop-free + Congestion-free.
- For every pair, *(old* ∪ *new)* is a **DAG** ⇒ Loop-freedom for free!
- **Feasibility**: is there a sequence of congestion-free updates?
- **Optimality**: how to minimize the number of rounds efficiently?

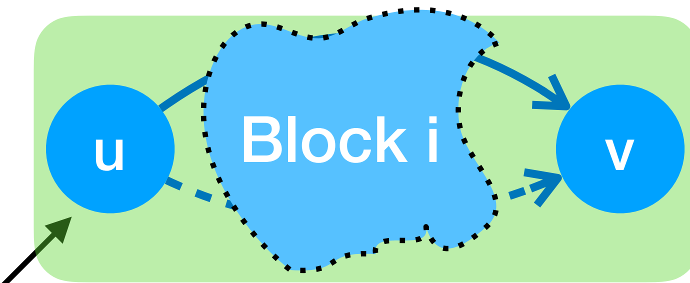
Block Decomposition



1) Take the union of old and new routes => a DAG
2) Obtain its nodes in a topologically sorted oder
3) Intersections consecutive in that order are **block**'s endpoints



Block Update



start of block

block update in 3 rounds

(1) activate all new route links not incident to **u**

(2) *update(**u**, **blue**)*

(3) deactivate links on the old route

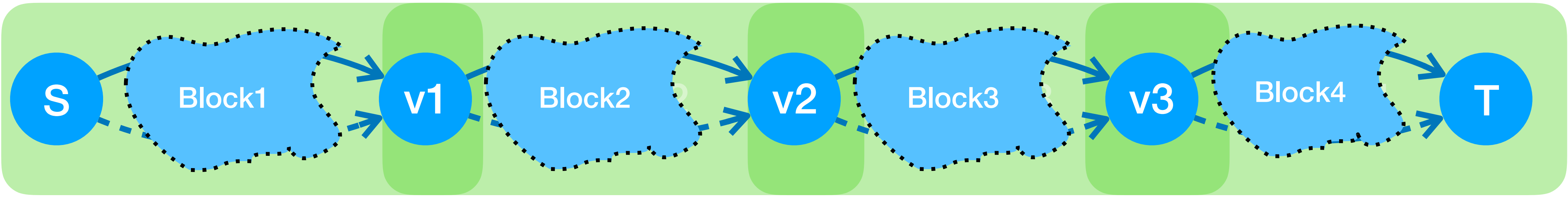
Lemma 2. *Given any feasible (not necessarily shortest) update sequence $\mathfrak{R}$, there is a feasible update sequence $\mathfrak{R}'$ which updates every block in at most 3 consecutive rounds.*

Solid: old route

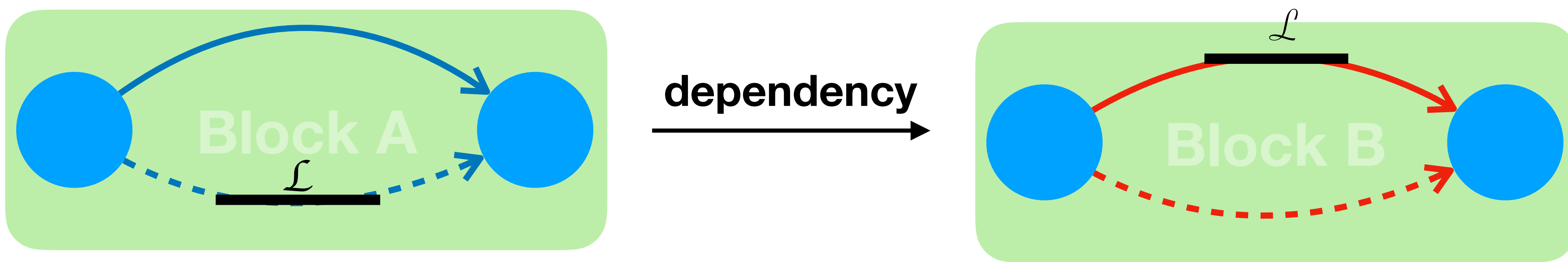
Dashed: new route

Towards an Efficient Algorithm

- Update the whole flow route block by block
- A block takes up to **3** rounds to update
- Blocks of the same flow can be updated independently
- Blocks of different flows may have dependencies



Towards an Efficient Algorithm for 2 Flows



- Link $\mathcal{L}$ has capacity 1
- Blue cannot reroute before Red reroutes away from $\mathcal{L}$
- Block A **depends** on Block B
- Any feasible schedule updates B before A

Lemma 3. *If there is a cycle in the dependency graph, then there is no feasible update sequence.*

Solid: old route

Dashed: new route

Overview of the Algorithm

1) Compute the block composition for the two flows.
2) Compute the dependency graph D of G.
3) If there is a cycle in D then terminate.
4) While D is not empty, repeat:
    1) Update all blocks that correspond to the **sink** vertices of D.
    2) Remove all the sink vertices from D.

of rounds

- All sink blocks are updated together, in 3 rounds.
- 3 rounds per iteration is not always necessary.
- Allocating the necessary rounds yields the optimal schedule

Theorem 1. *An optimal (feasible) update sequence on acyclic update flow networks with exactly 2 update flow pairs can be found in linear time.*

Hardness

Theorem 2. *Deciding whether a feasible network update schedule exists for a given update flow network in which each flow pair forms a DAG is* NP*-hard for six flows.*

The complexity for 3 to 5 flows remains open.

Summary

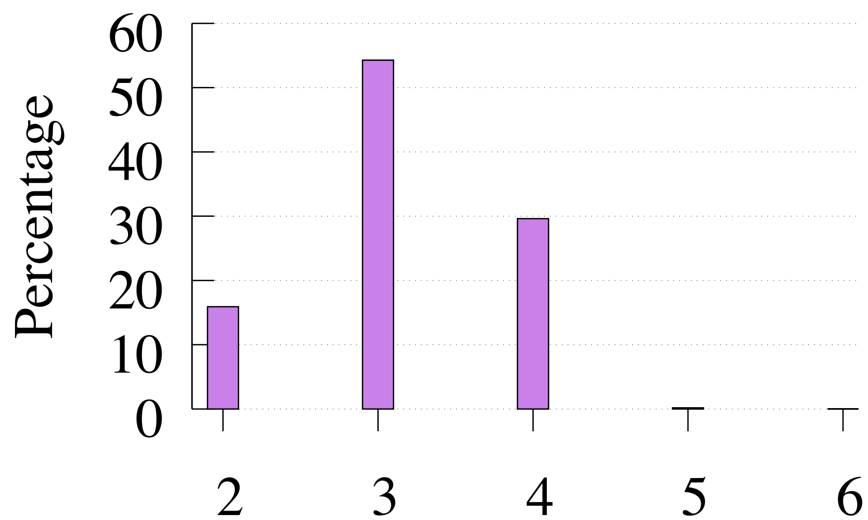
- NP-hard from 6 flows
- special case: 2 flows, every route pair forms a **DAG**
- optimal schedule for **2 flows**, given **acyclic** pairs
- congestion-free schedule for $3 \leq k \leq 5$ flows? **open**

Summary

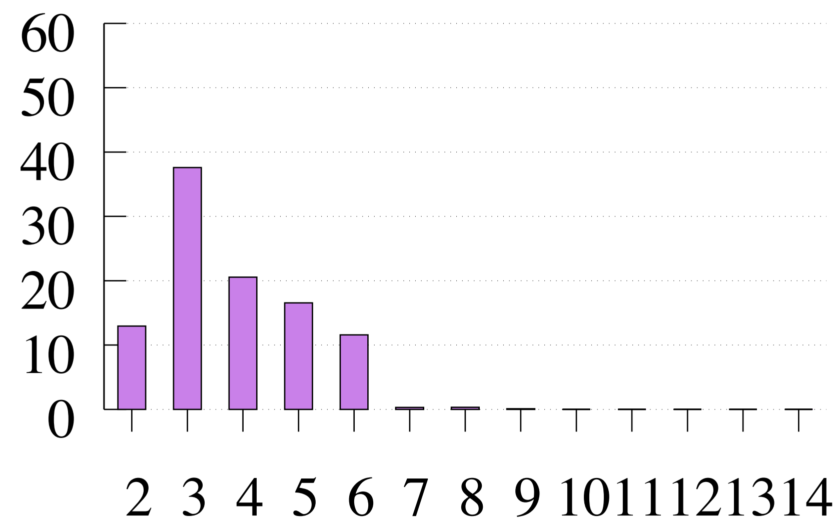
- NP-hard from 6 flows
- special case: 2 flows, every route pair forms a **DAG**
- optimal schedule for **2 flows**, given **acyclic** pairs
- congestion-free schedule for $3 \leq k \leq 5$ flows? **open**

Thank you!

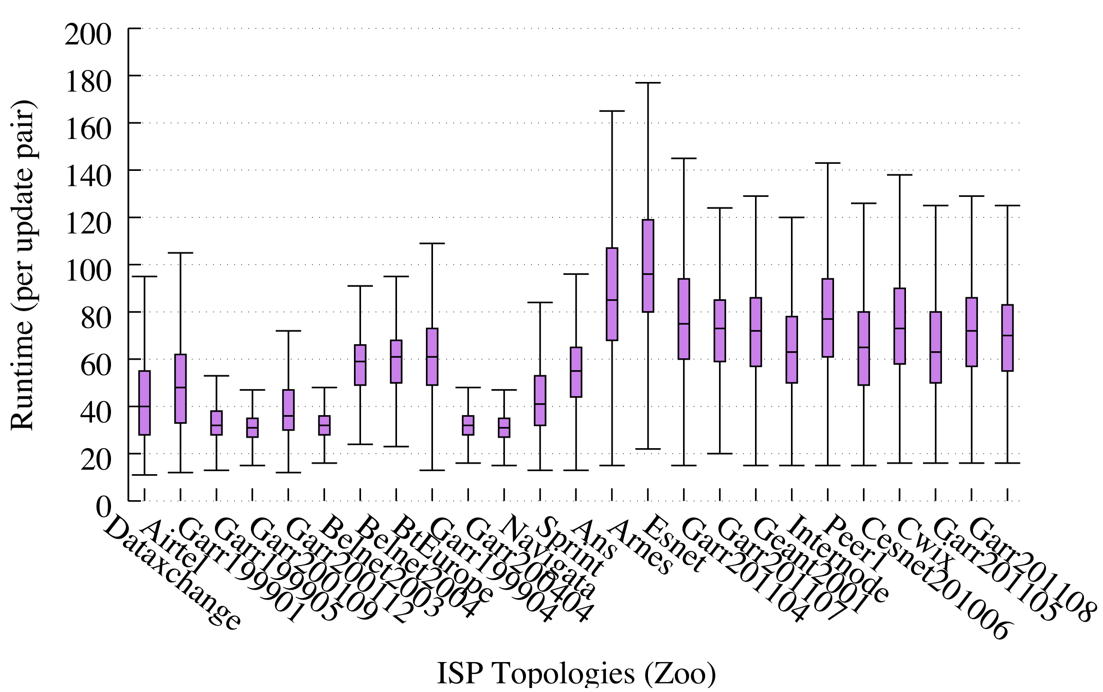
Backup slides



The frequency of each possible number of rounds (in %) in optimal schedules from Algorithm 2

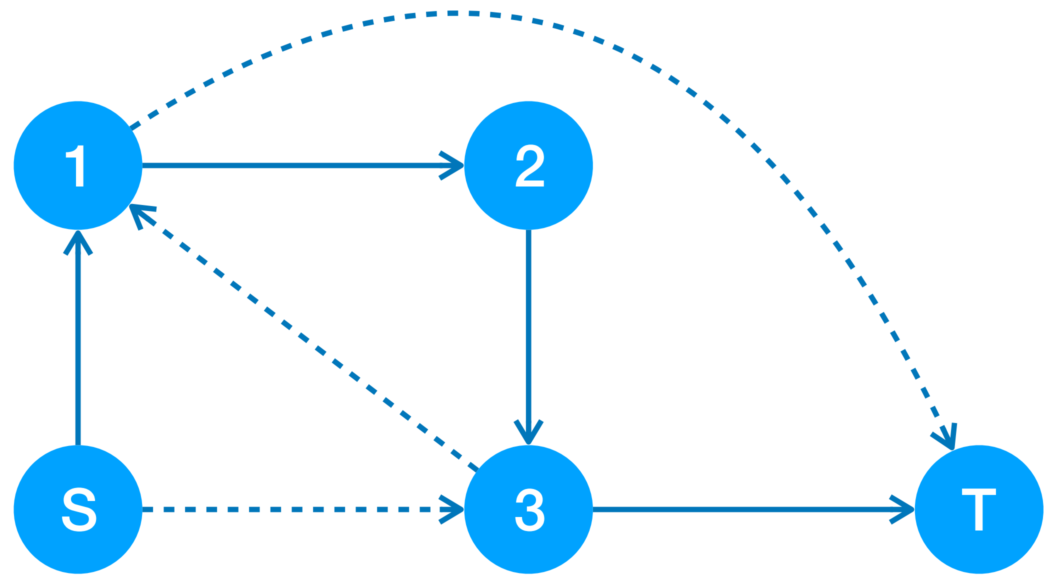


arbitrary feasible schedules from Algorithm 1



Distribution of runtime (in microsecond) over all the problem instances from some of the evaluated graphs

Loop-free Rerouting



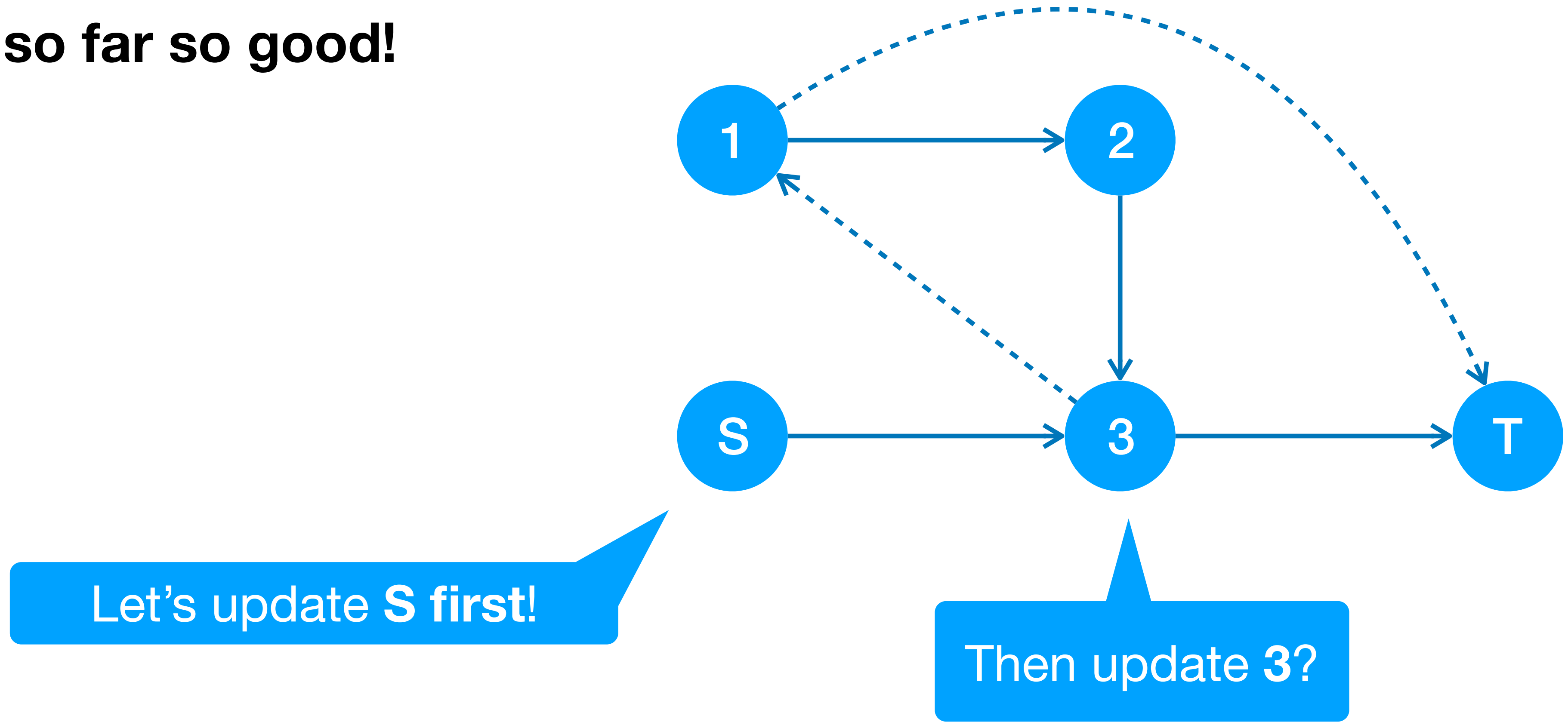
Let’s update **S first**!

Solid: old route

Dashed: new route

Loop-free Rerouting

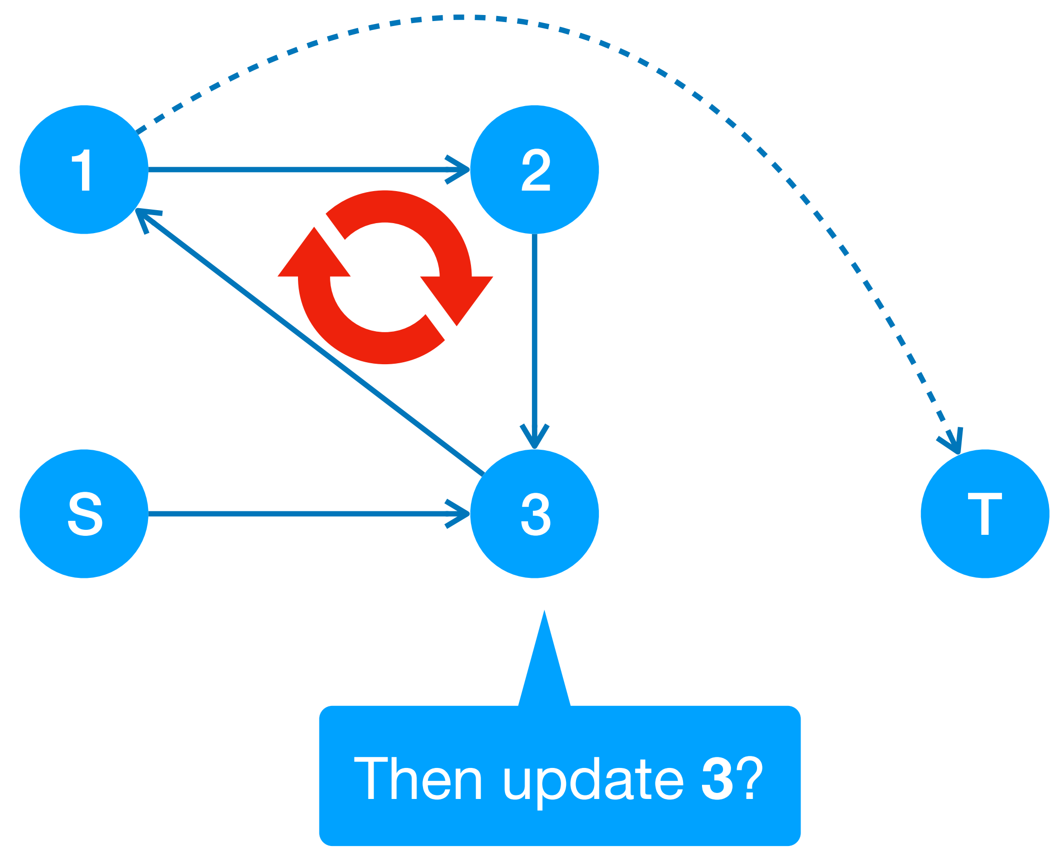
so far so good!



Solid: old route

Dashed: new route

Loop-free Rerouting

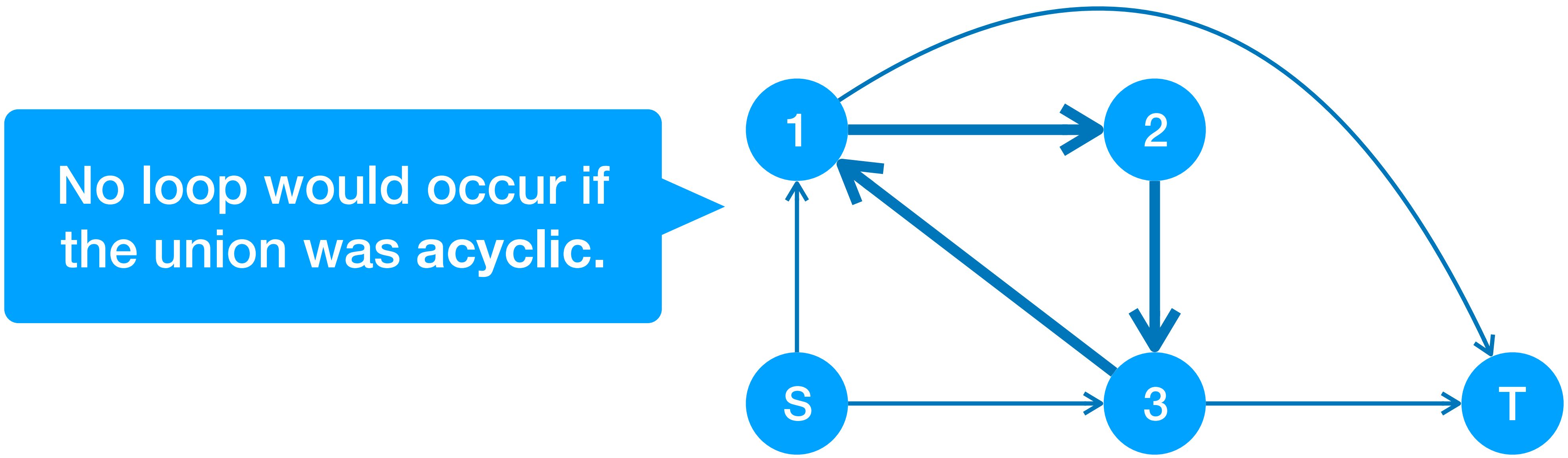


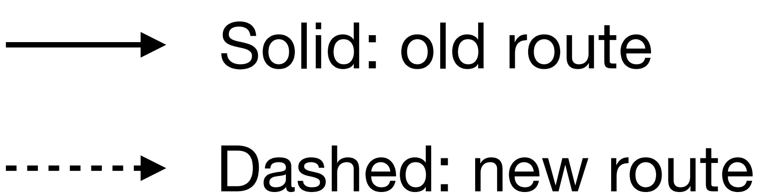
Not consistent!

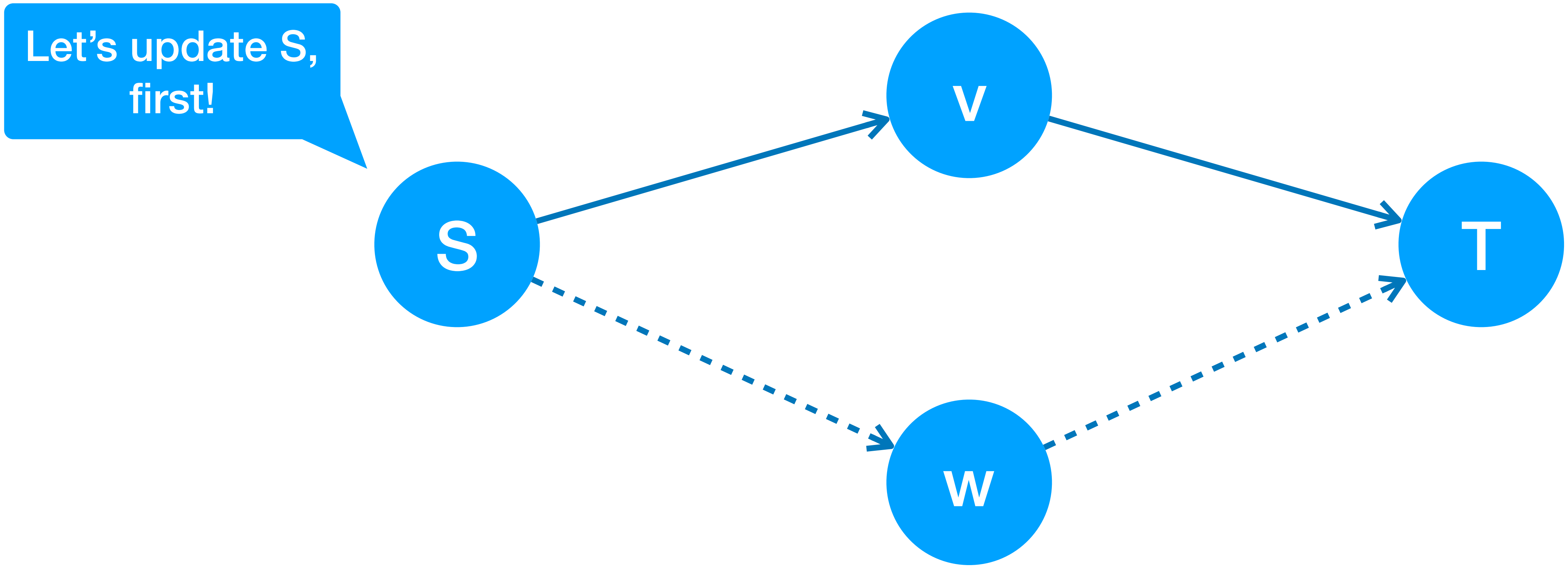
Solid: old route

Dashed: new route

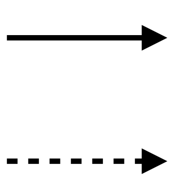
Loop-free Rerouting

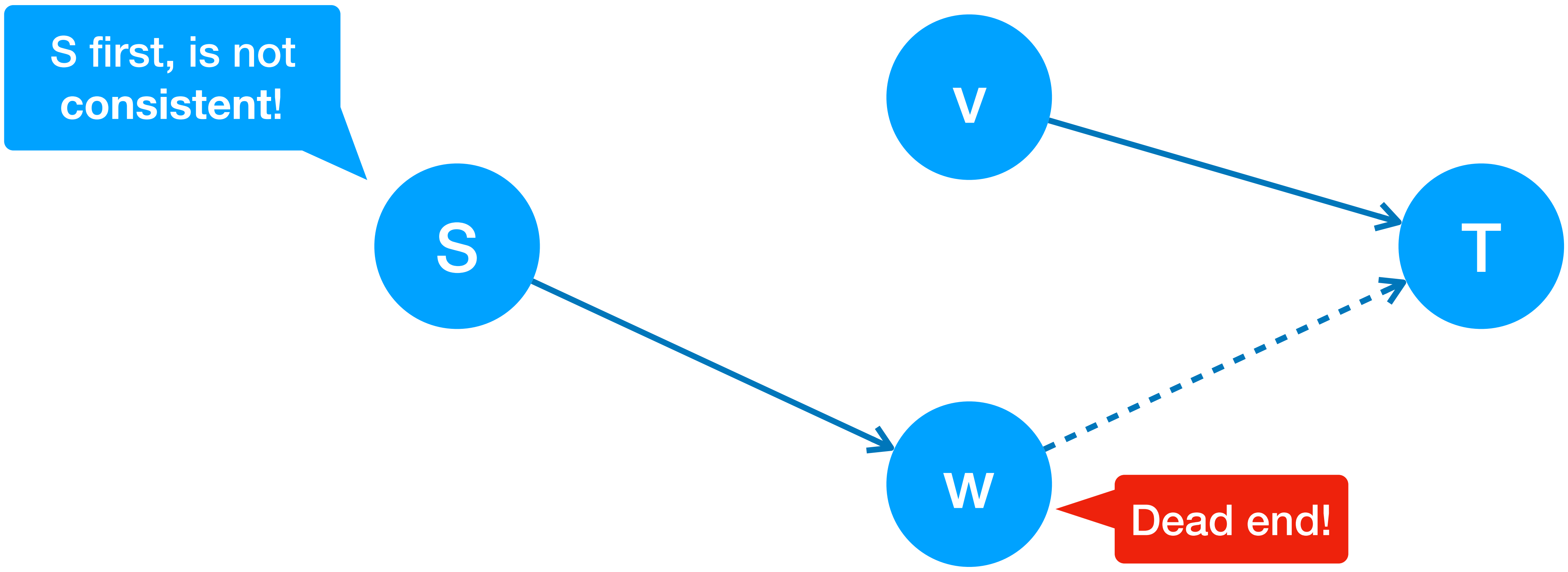




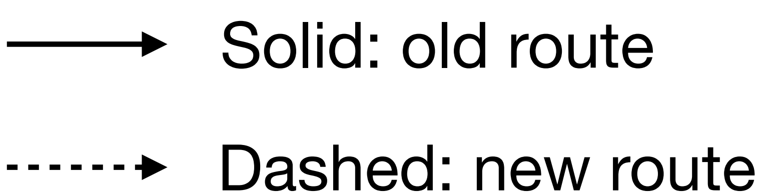


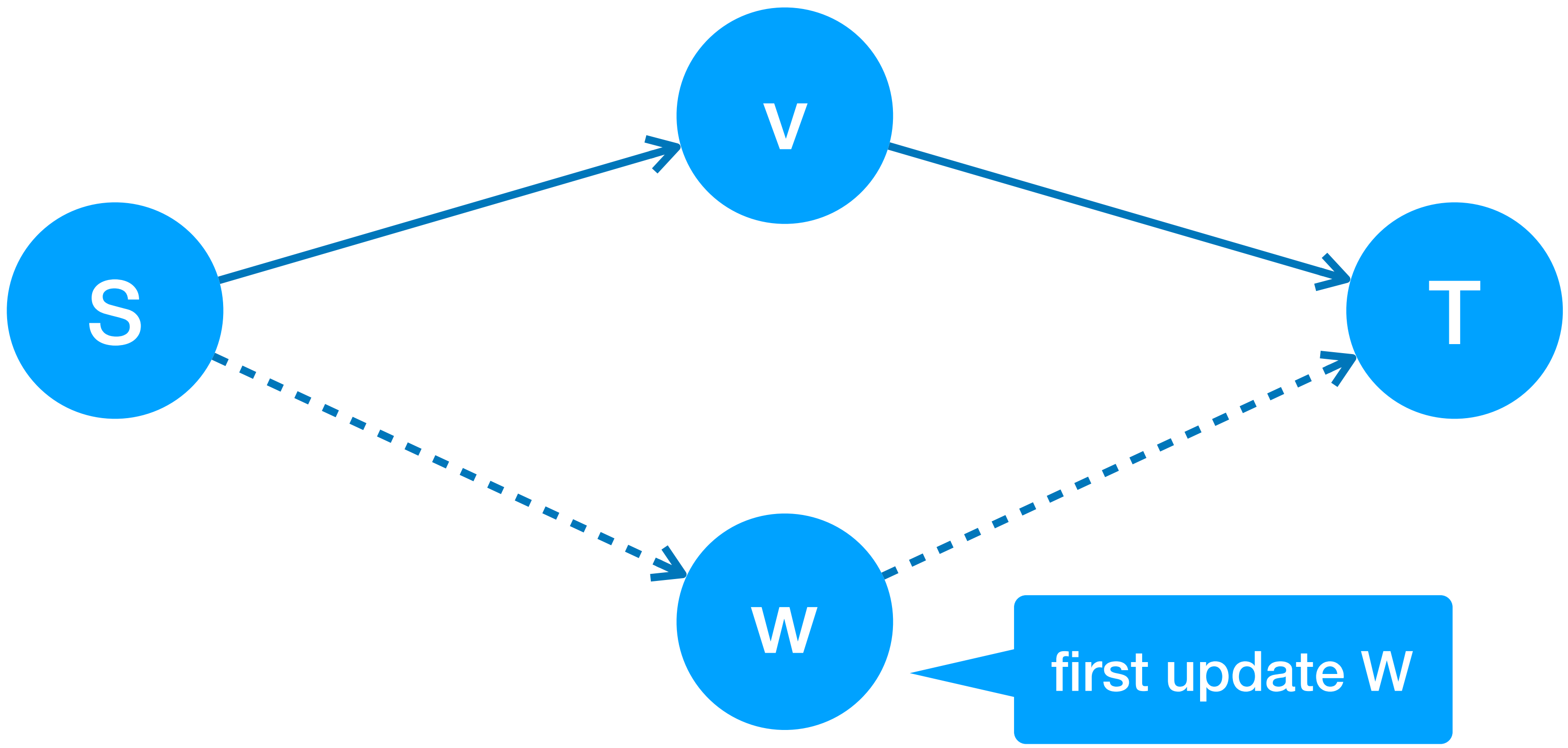
rerouting the old route to the new route





rerouting the old route to the new route

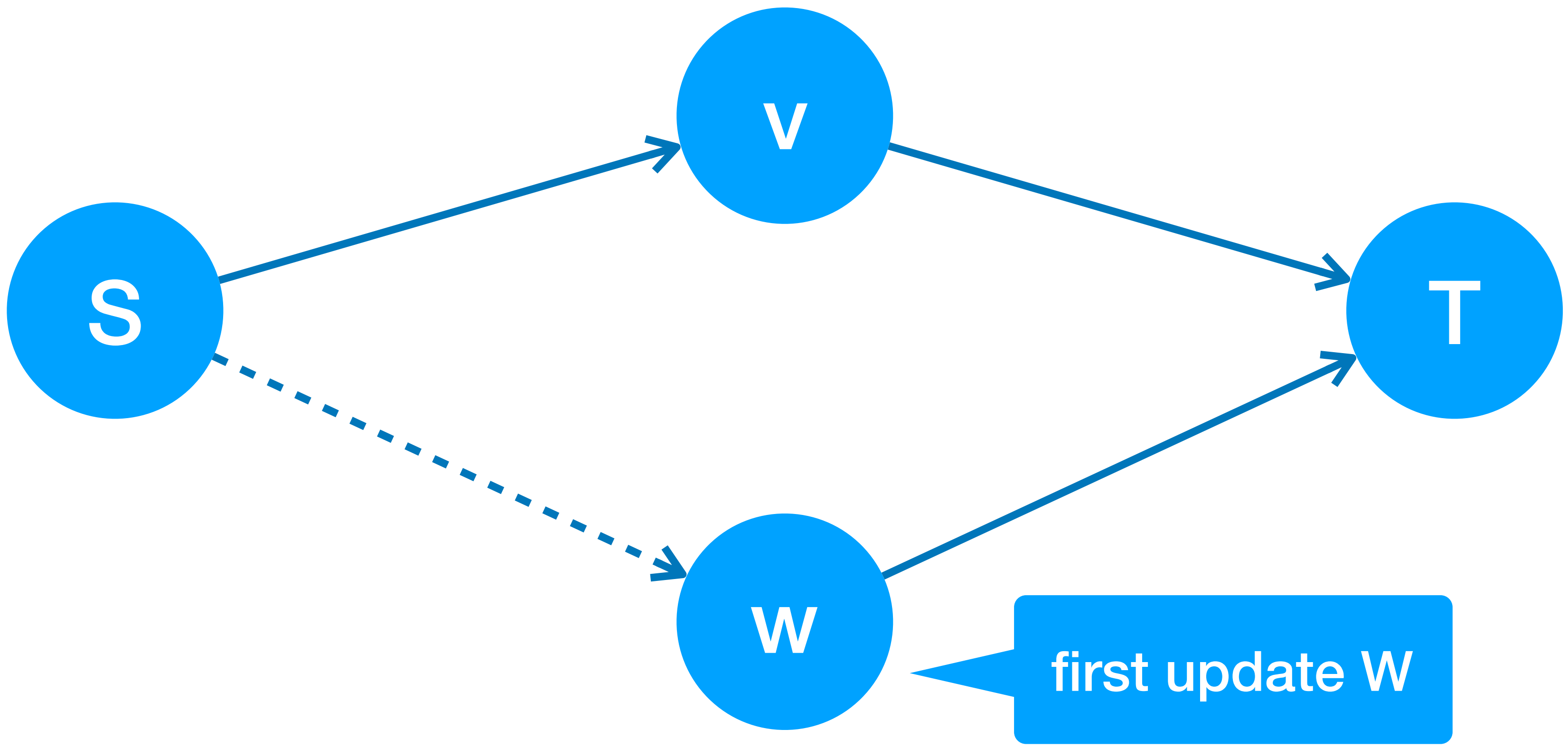




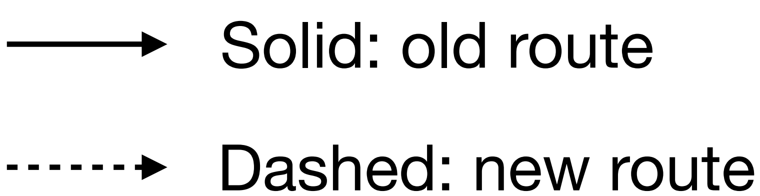
rerouting the old route to the new route

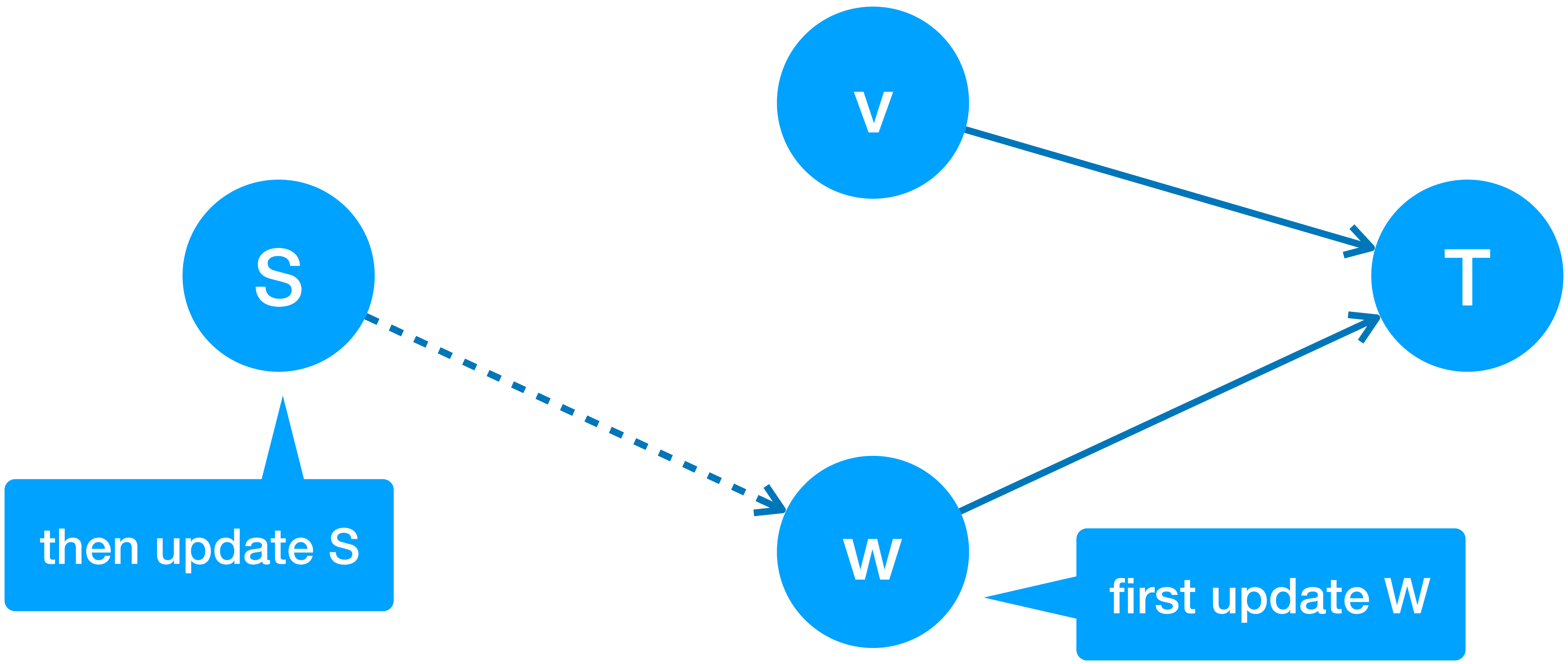
Solid: old route

Dashed: new route

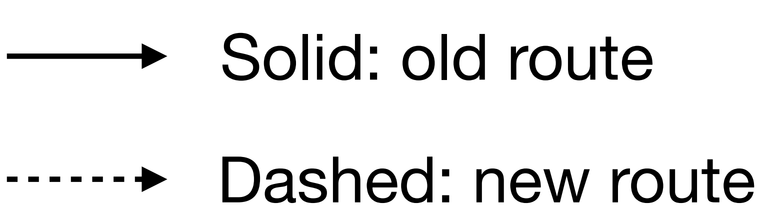


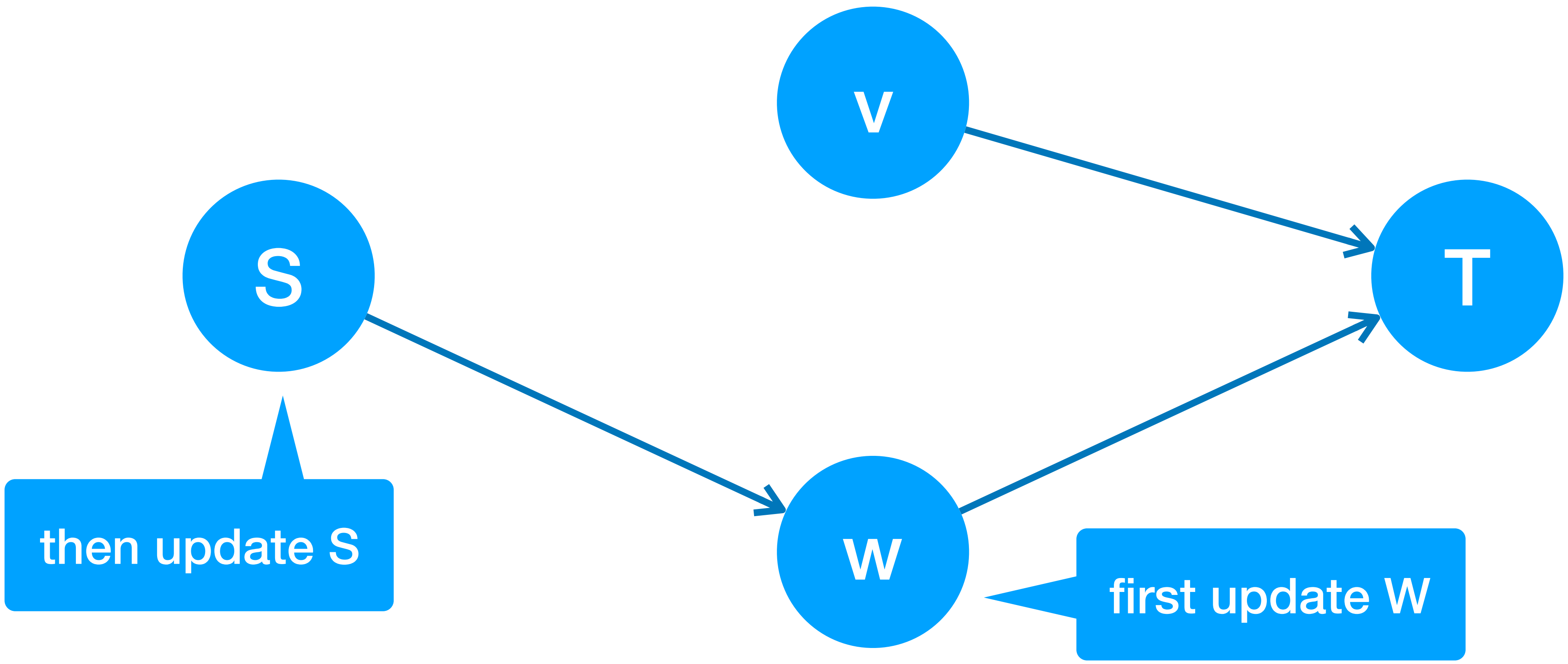
rerouting the old route to the new route



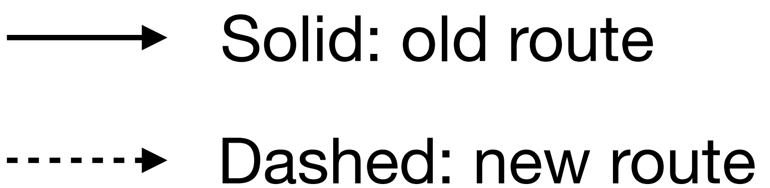


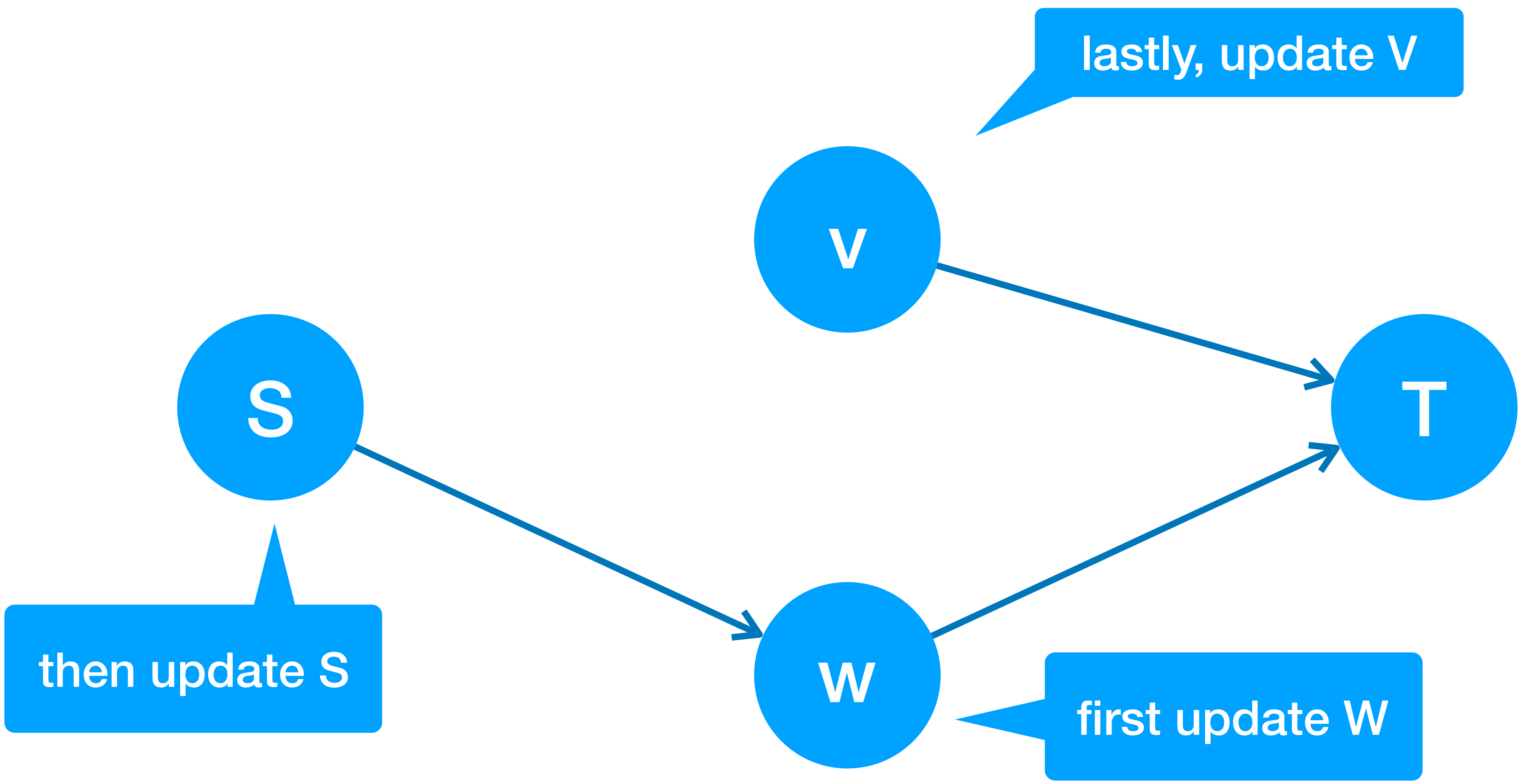
rerouting the old route to the new route



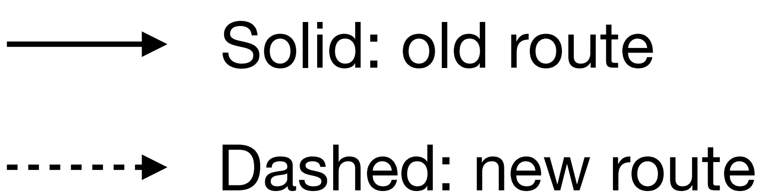


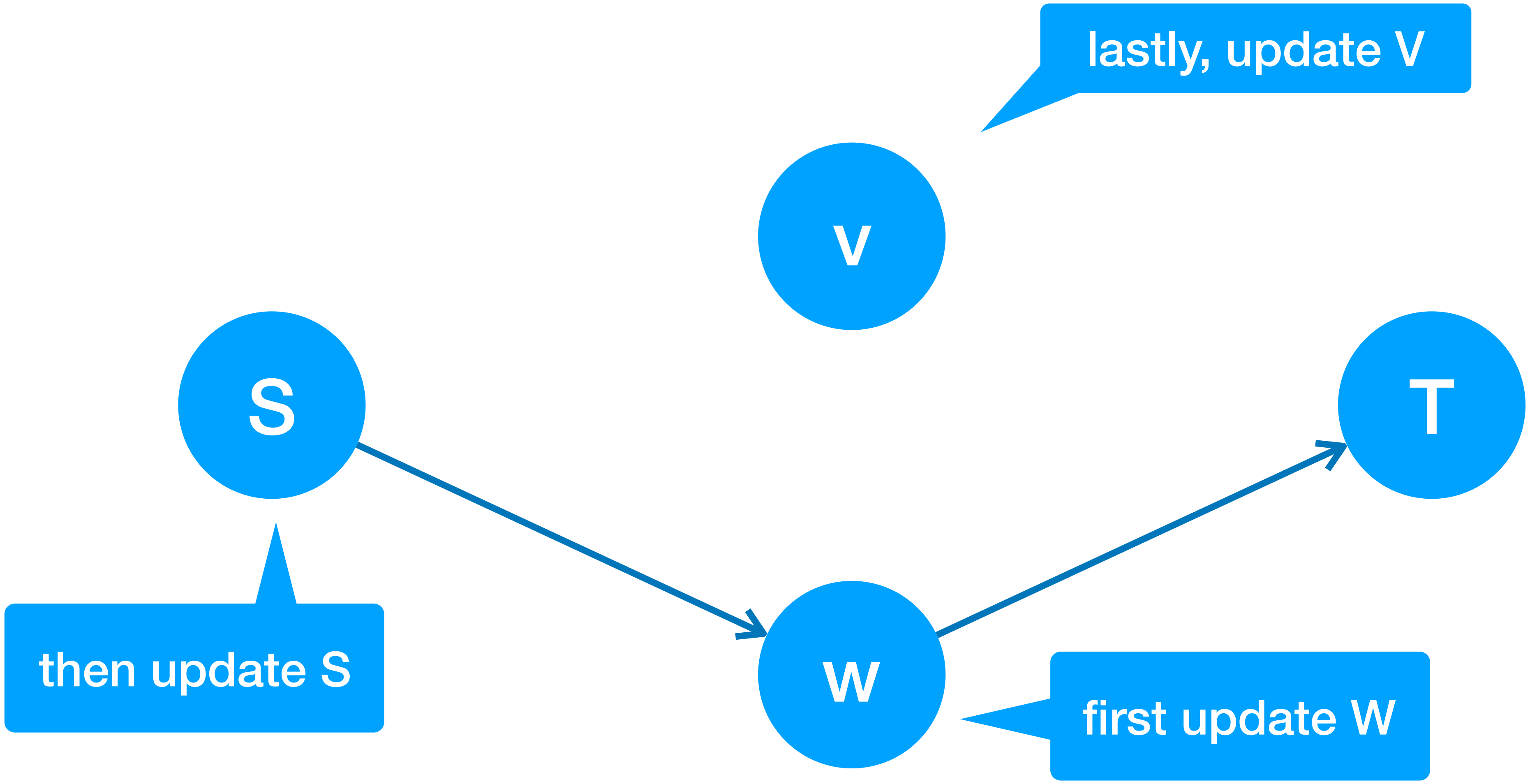
rerouting the old route to the new route





rerouting the old route to the new route





rerouting the old route to the new route

Solid: old route

Dashed: new route