Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates

Klaus-T. Foerster

Brief Preamble

- Focus on algorithmic/complexity issues in consistent updates in Software Defined Networks (SDNs)
  - Not so much on system etc. issues respectively SDNs themselves

- Two “bigger” connections to classic distributed computing halfway-in
  - Proof Labeling Schemes
  - Distributed Control Plane

universität wien

Network Updates

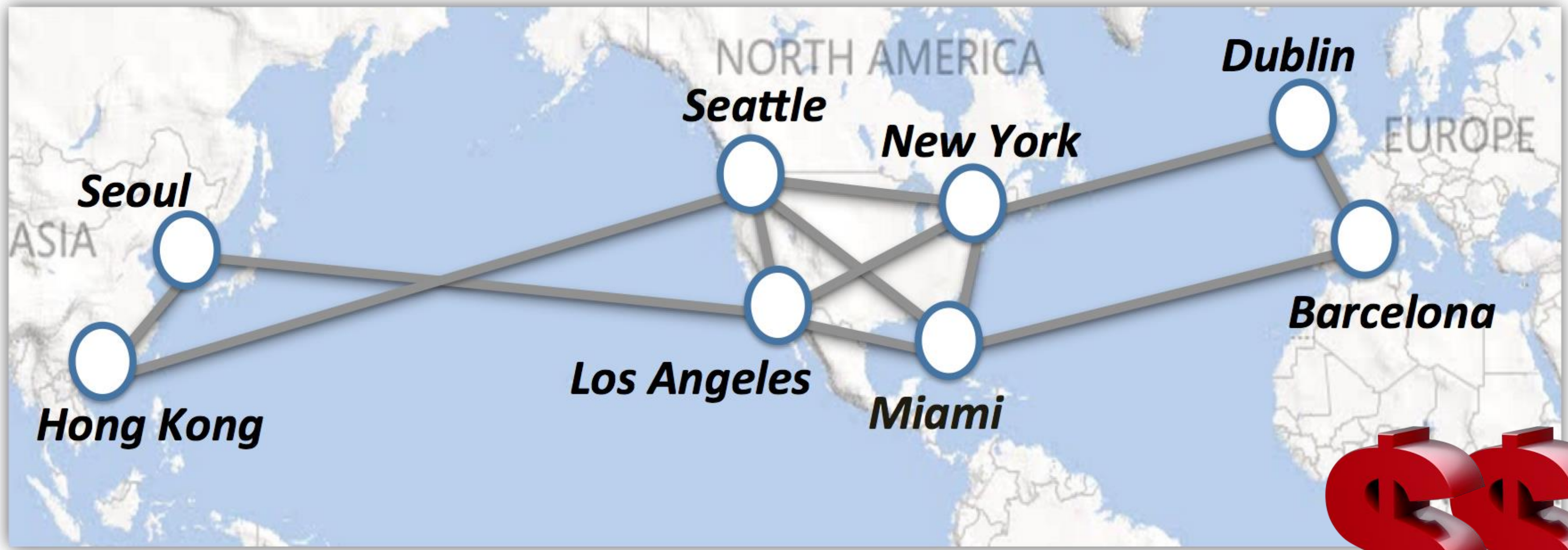
- The Internet: Designed for selfish participants
  - Often inefficient (low utilization of links), but robust

- But what if eg the Wide-Area Network is controlled by a single entity?
  - Examples: Microsoft & Amazon & Google …
  - They spend hundreds of millions of dollars per year

Also relevant in eg Data Center Networks, for ISPs etc

Network Updates

Eg update link capacity at runtime?*



Think: Google, Amazon, Microsoft

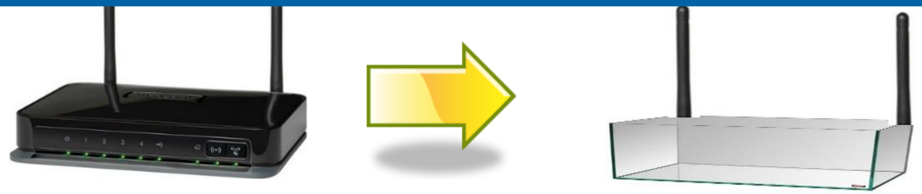
*:RADWAN: Rate Adaptive Wide Area Network. R. Singh, M. Ghobadi, K.-T. Foerster, M. Filer, P. Gill. ACM SIGCOMM 2018

Note: There is also a lot of (prior) research on consistency before SDNs – can't cover everything in this tutorial

See history section in:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019

Software-Defined Networking

- Possible solution:
  - **S**oftware-**D**efined **N**etworking (**SDN**s)



- General Idea: Separate data & control plane in a network
- Centralized controller updates networks rules for optimization
  - Controller (*control plane*) updates the switches/routers (*data plane*)

Virtual Services ⟷ Controller ⟷ Physical Network

- Logically centralized controller (eg implemented with replication)

old network
rules

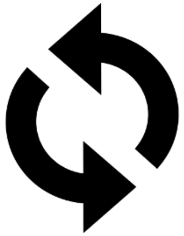
network updates

new network
rules

old network
rules

network updates

new network
rules



old network
rules

network updates

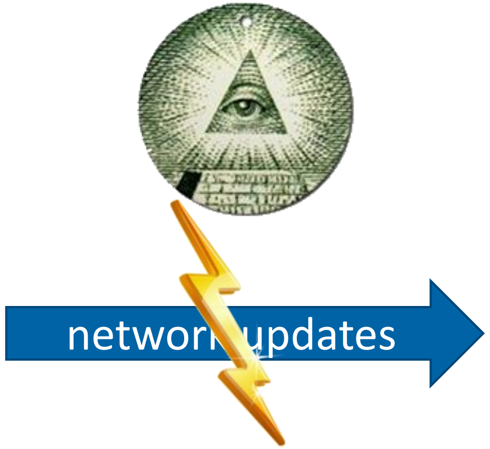
new network
rules

possible solution: be fast!

But they deviated from that a bit in the B4 2018 version…

e.g., B4 (Google, 2013)

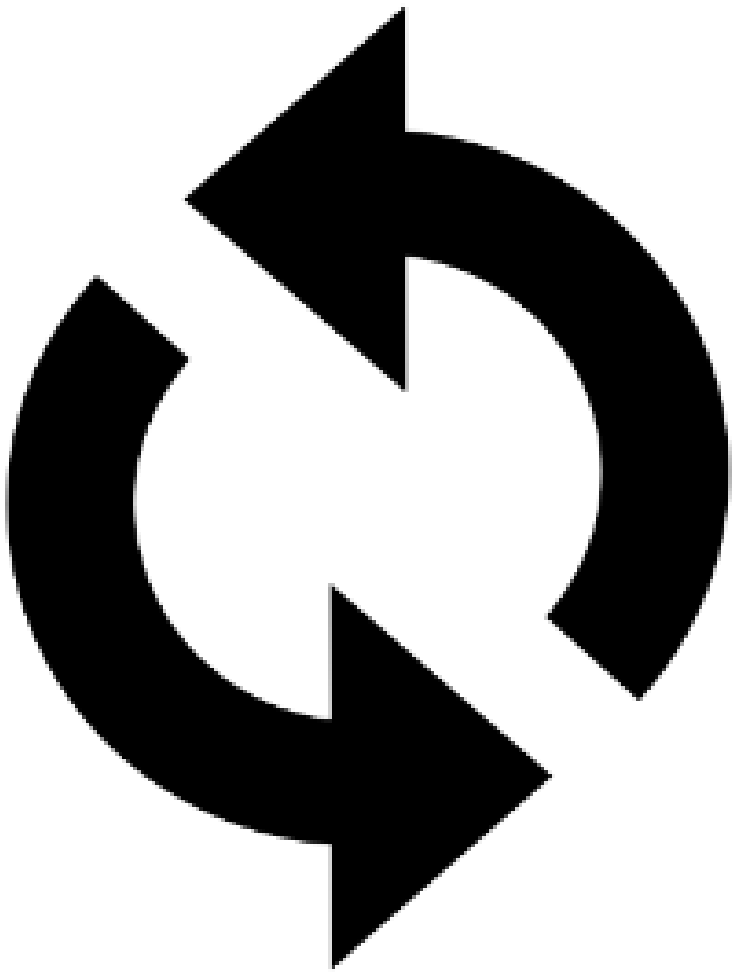
old network
rules



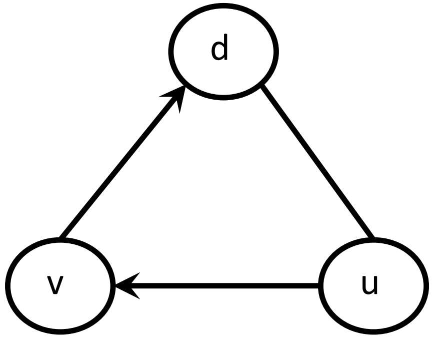
new network
rules

Alternative: Be consistent!

- Algorithms with guarantees

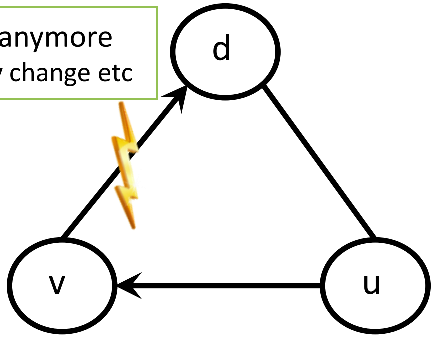


Toy Example

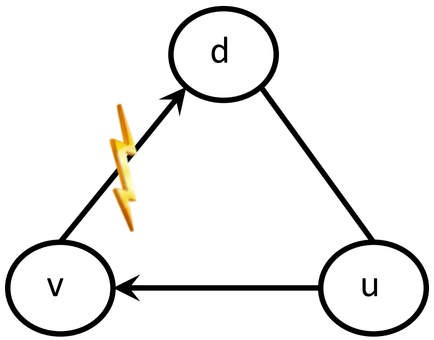


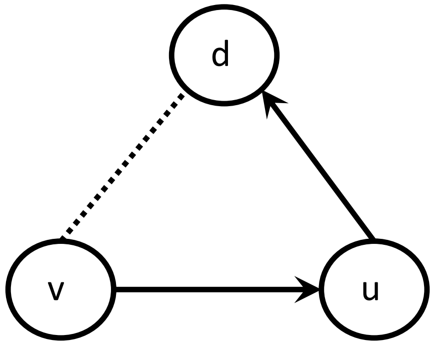
Toy Example

Link should not be used anymore
eg repair, congestion, policy change etc



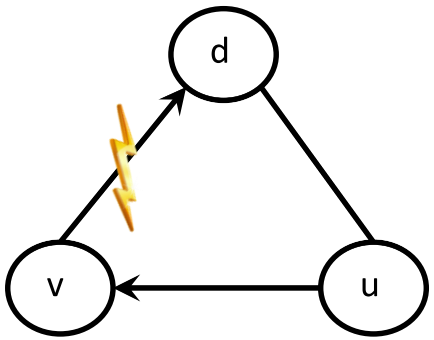
Toy Example

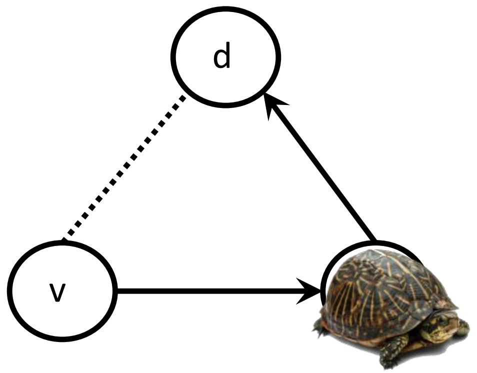




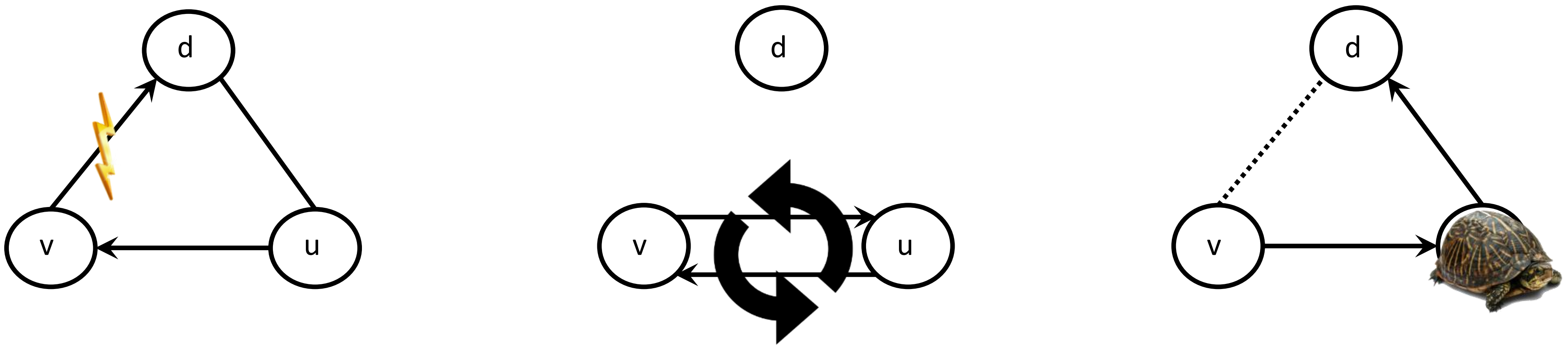
Toy Example

Toy Example

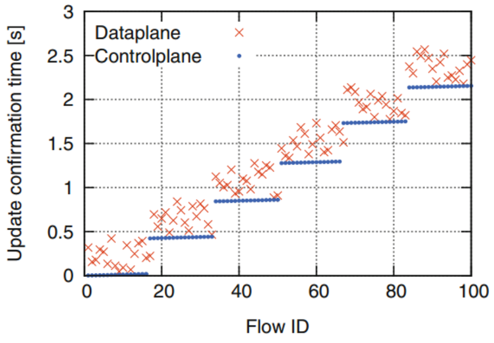




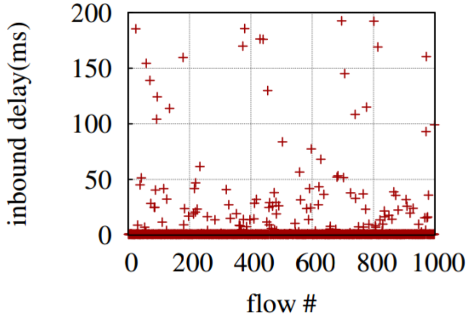
Toy Example



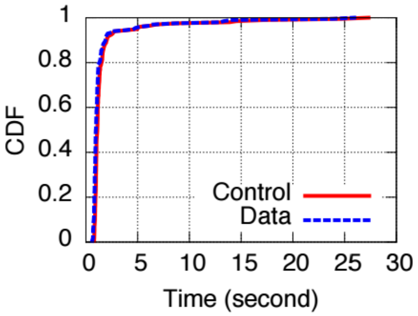
Appears in Practice



"Data plane ***updates may fall behind*** *the control plane acknowledgments and may be even* ***reordered****."*
Kuzniar et al., PAM 2015

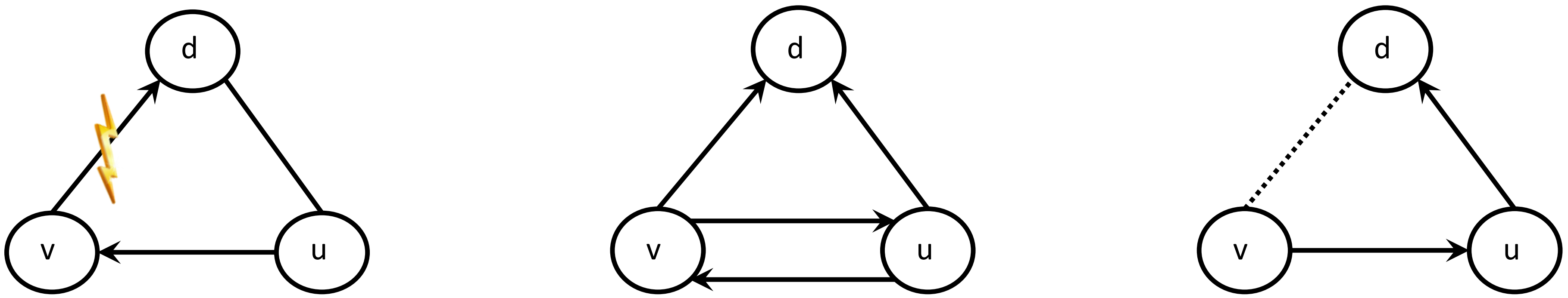


"…the inbound latency is ***quite variable*** *with a […] standard deviation of 31.34ms…"*
He et al., SOSR 2015



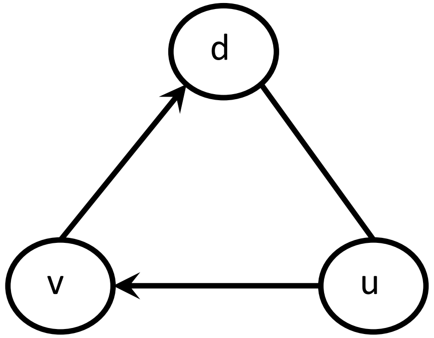
*"some switches can '****straggle****,' taking substantially* ***more time*** *than average (e.g., 10-****100x****) to apply an update"*
Jin et al., SIGCOMM 2014

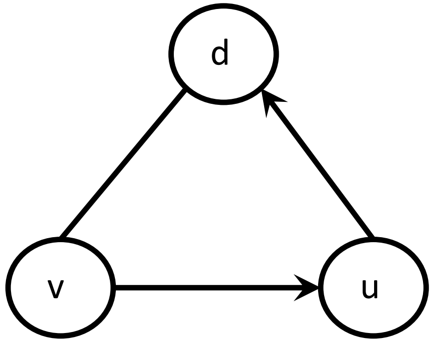
Toy Example



Old and new states exist simultaneously in a limbo state

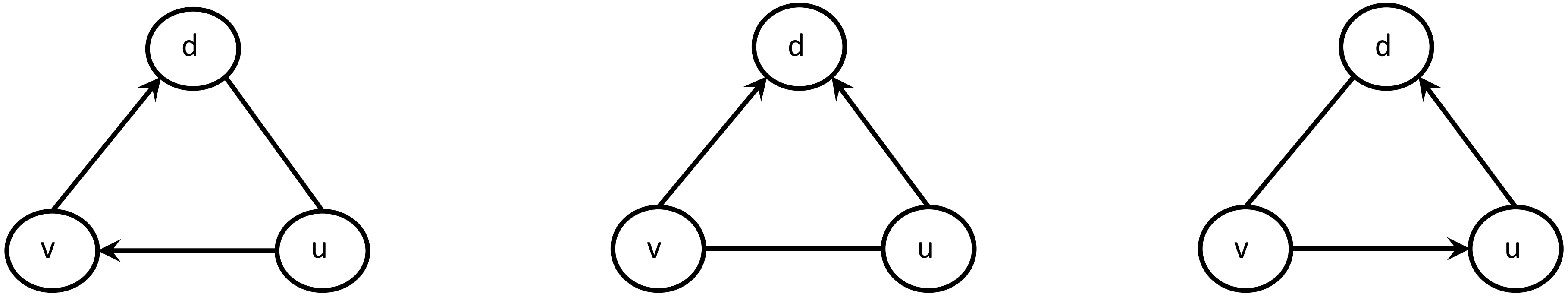
Ordering Solution: Go backwards through the new routing tree



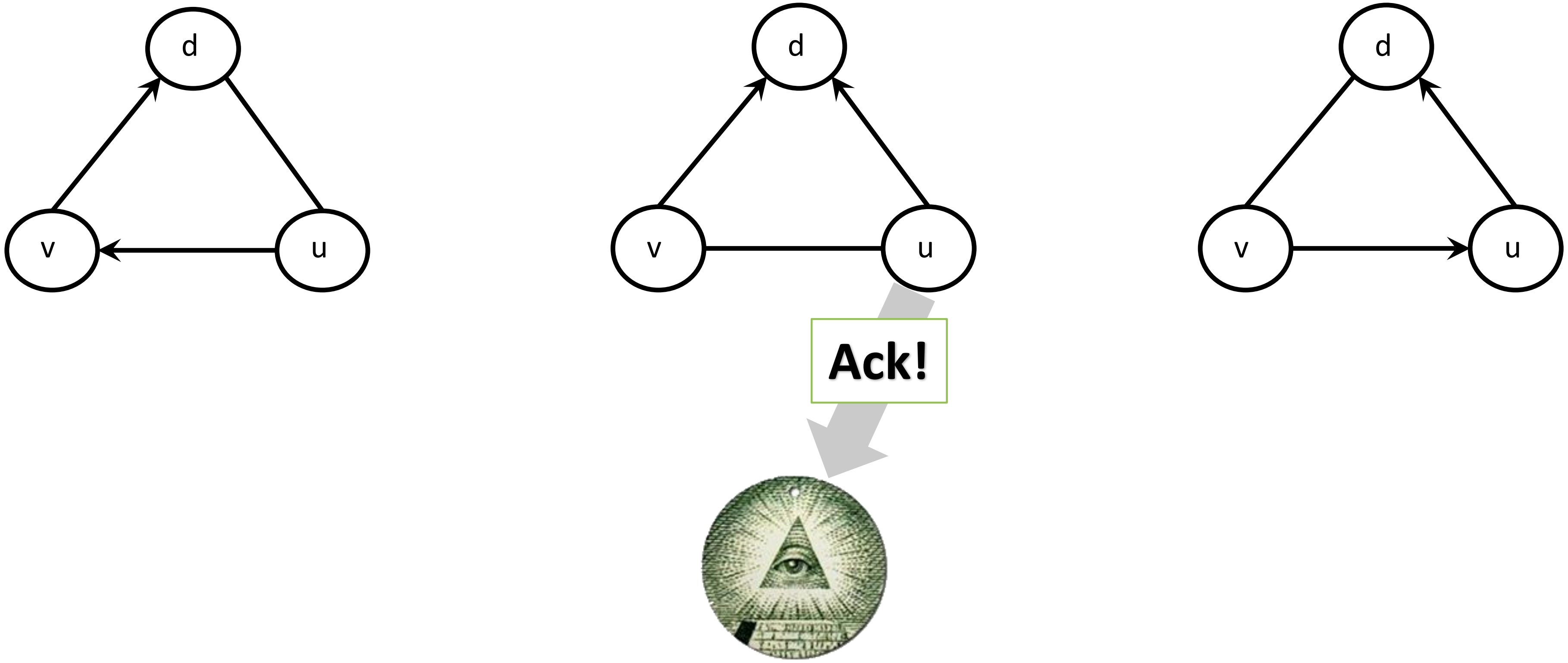


Ordering Solution: Go backwards through the new routing tree

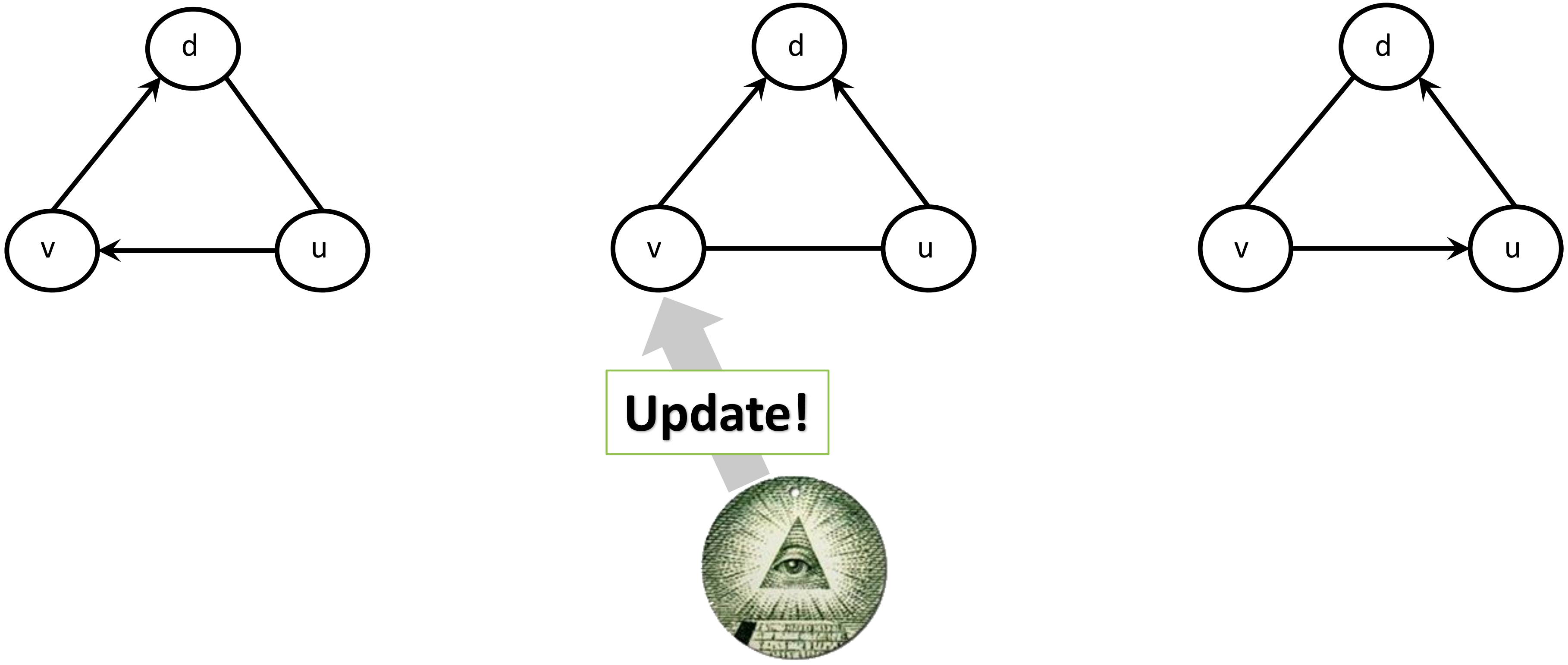
Ordering Solution: Go backwards through the new routing tree



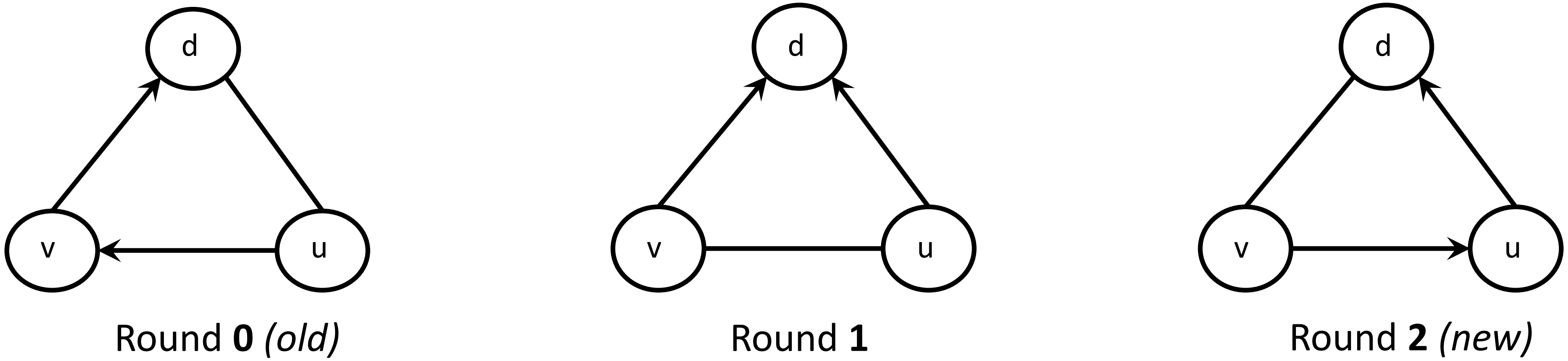
Ordering Solution: Go backwards through the new routing tree



Ordering Solution: Go backwards through the new routing tree



Ordering Solution: Go backwards through the new routing tree

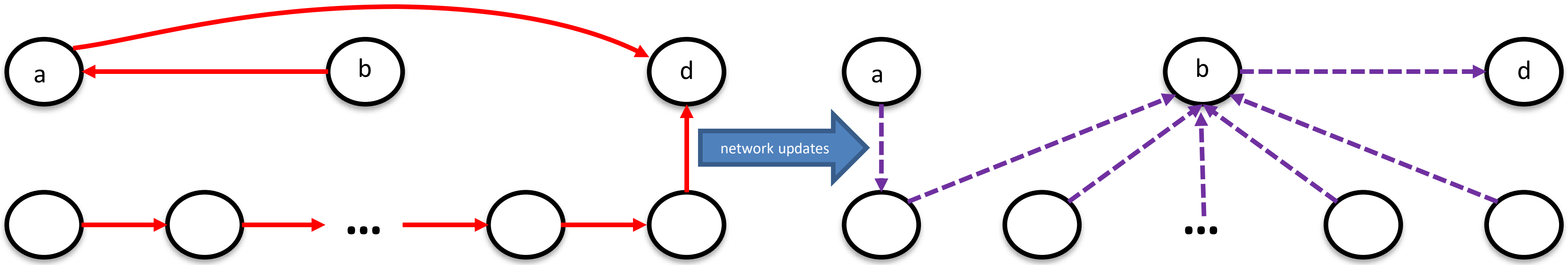


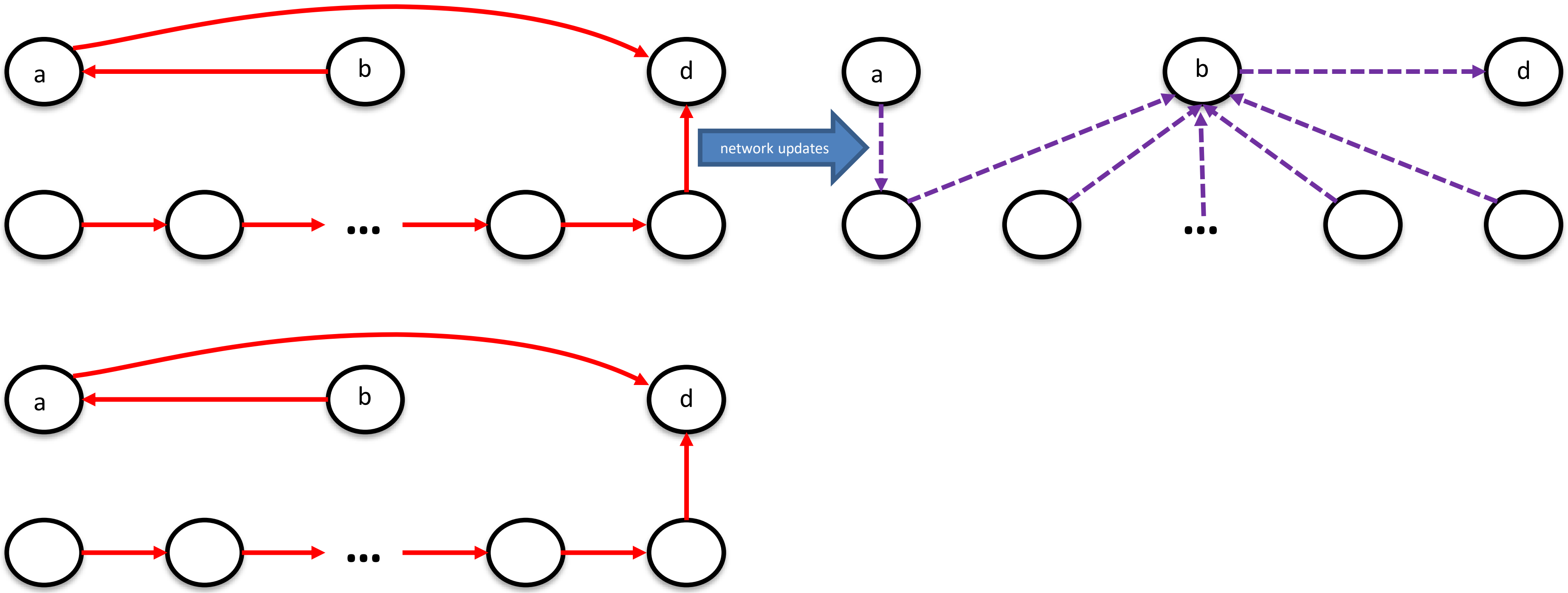
- Always works for single-destination rules
  - Also for multi-destination with sufficient memory (“split“)
- Schedule length: tree depth (up to Ω(n))
  - Optimal scheduling algorithms?

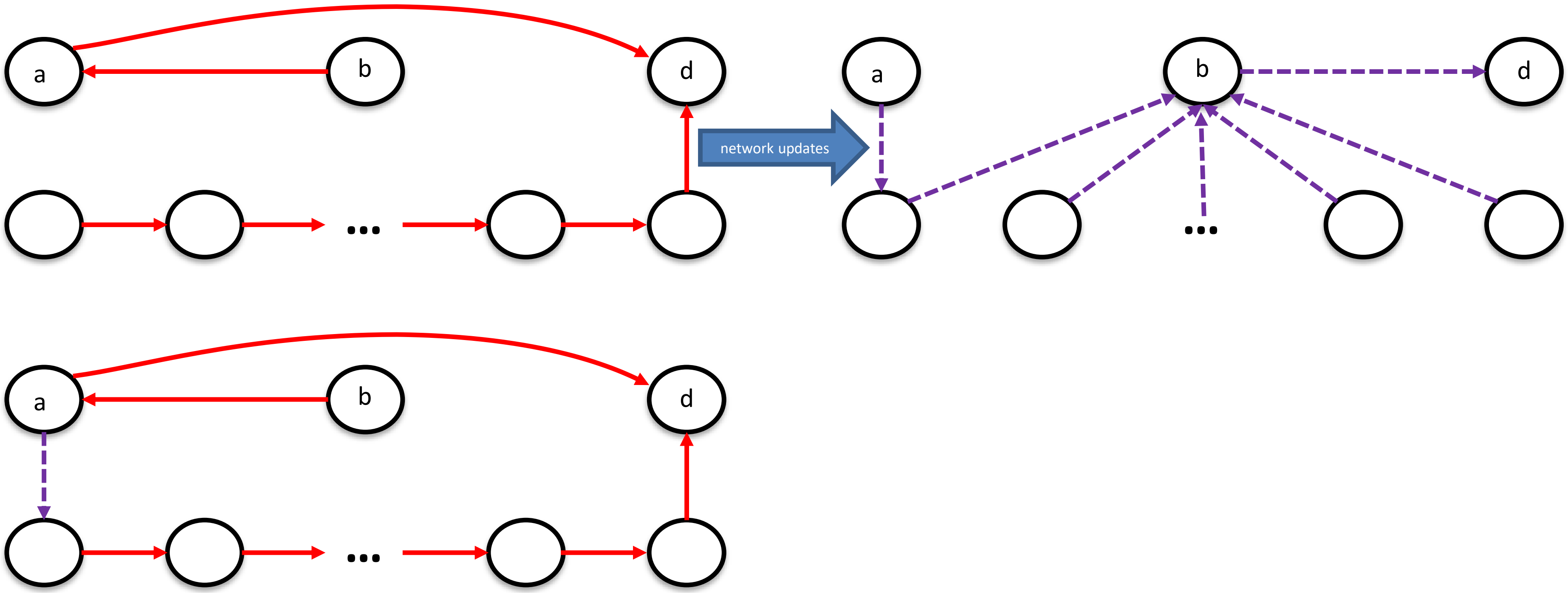
More on scheduling multiple policies:
Basta et al: Efficient Loop-Free Rerouting of Multiple SDN Flows. ToN 2018

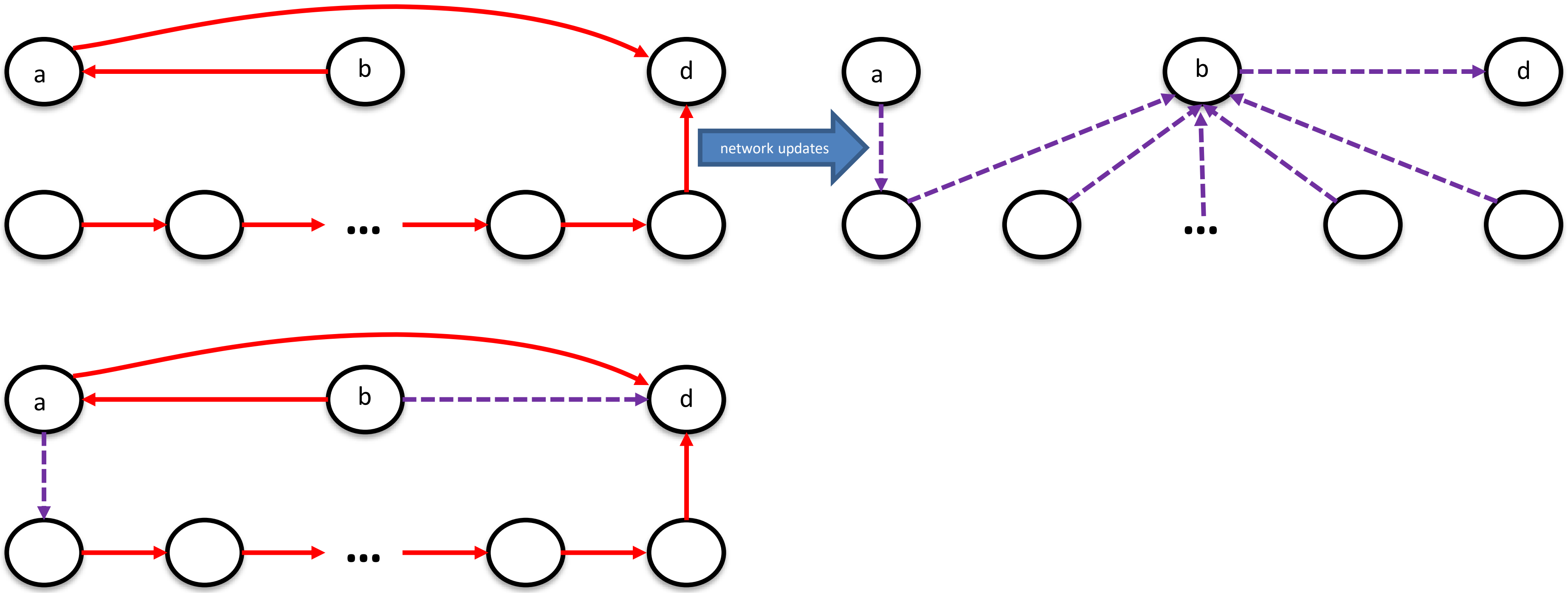
Greedy? Update as many as possible per round

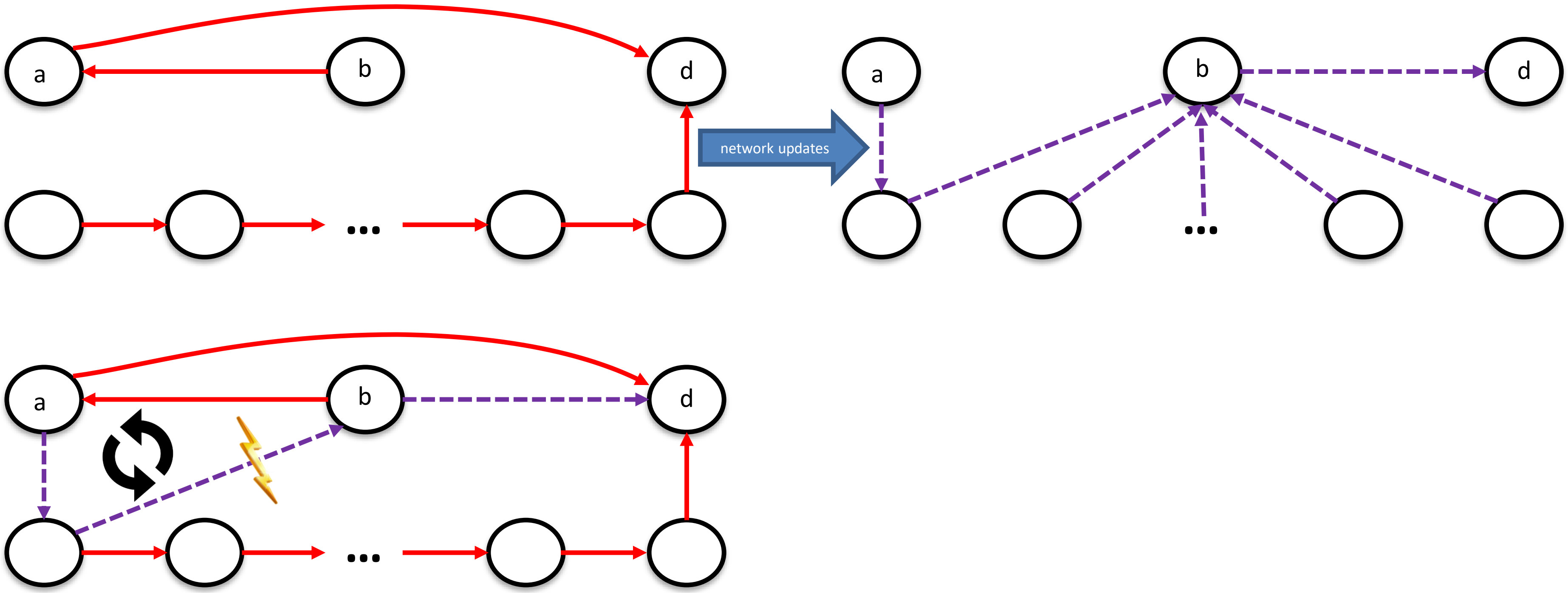
- Always works ☺

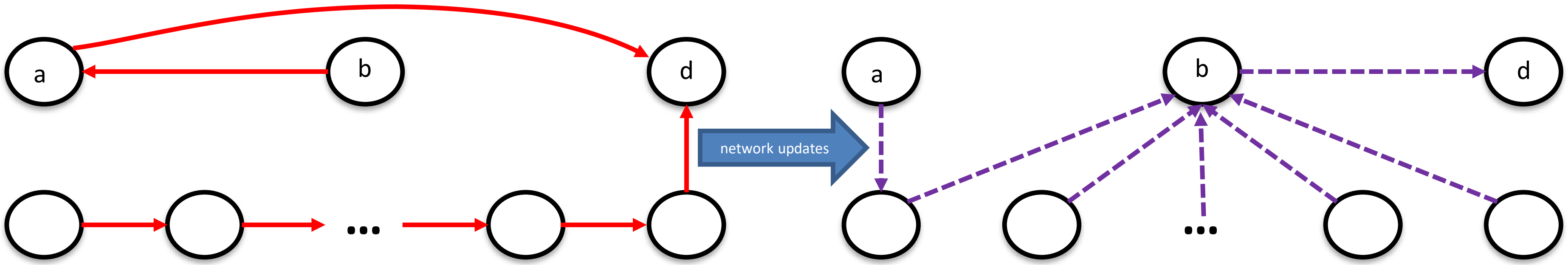

a
b
d
…
network updates
a
b
d
…

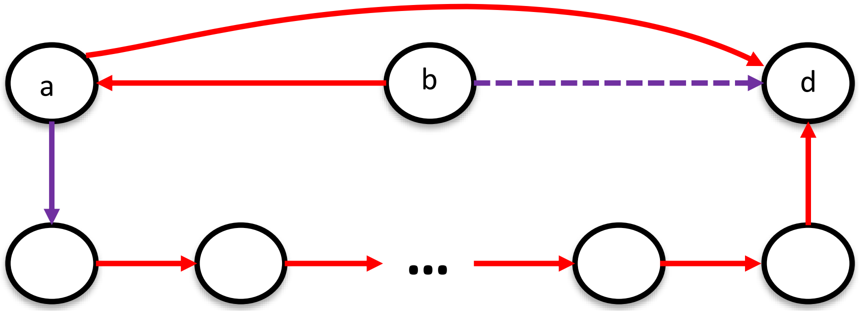

a
b
d
network updates
a
b
d
a
b
d


a
b
d
...
network updates
a
b
d
...
a
b
d
...

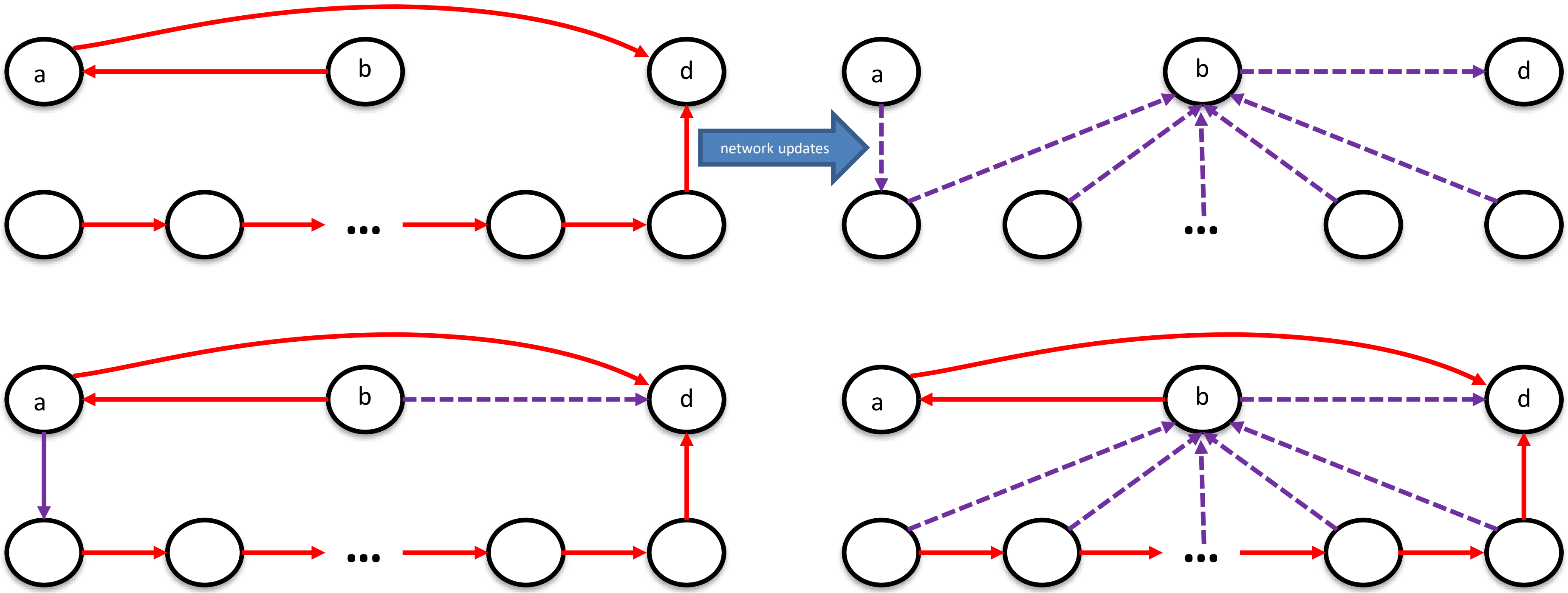

a
b
d
network updates
a
b
d
a
b
d


a
b
d
network updates
a
b
d
a
b
d





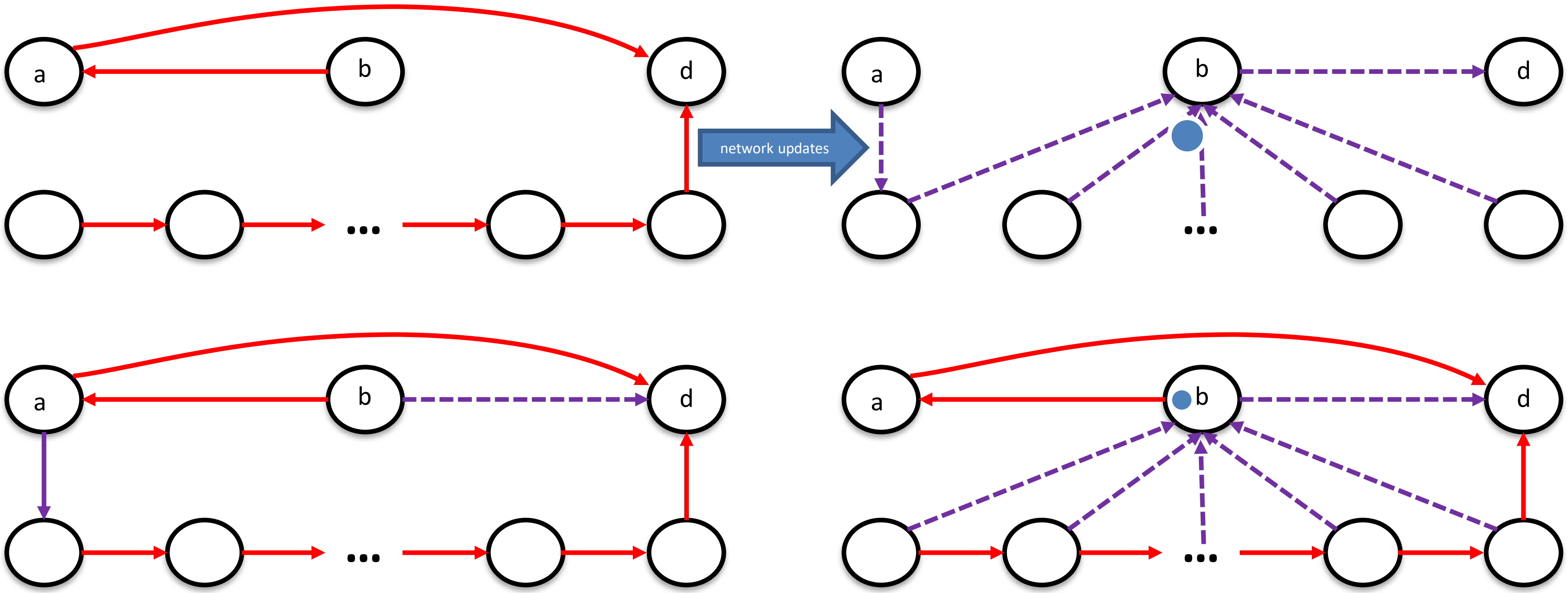
greedy **maximal** update
a & b update → all others wait
2 nodes update



greedy **maximal** update
a & b update → all others wait
2 nodes update

maximum update
a waits→ all others update
all but 1 update

How hard?

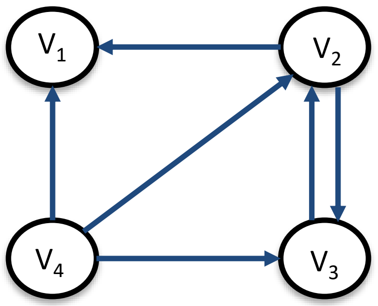


greedy **maximal** update
a & b update → all others wait
2 nodes update

maximum update
a waits→ all others update
all but 1 update

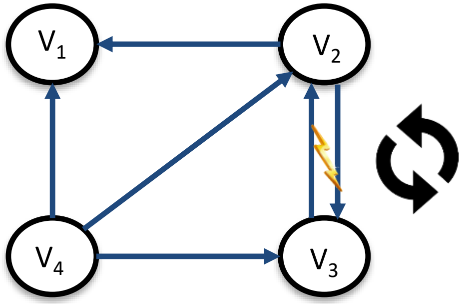
Find maximum update?

- Let’s go more general
- Delete all cycles in a graph



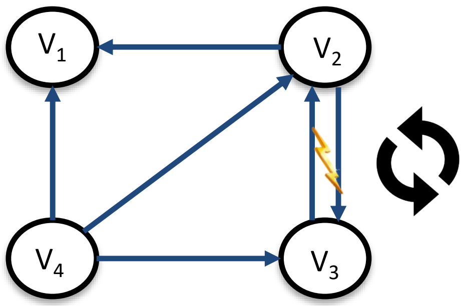
Find maximum update?

- Let's go more general
- Delete all cycles in a graph



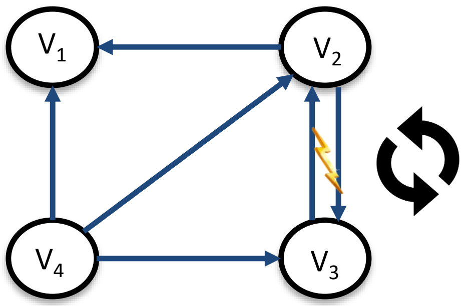
Find maximum update?

- Let's go more general
- Delete all cycles in a graph
- **NP-hard** to approximate
  – *Feedback Arc Set*

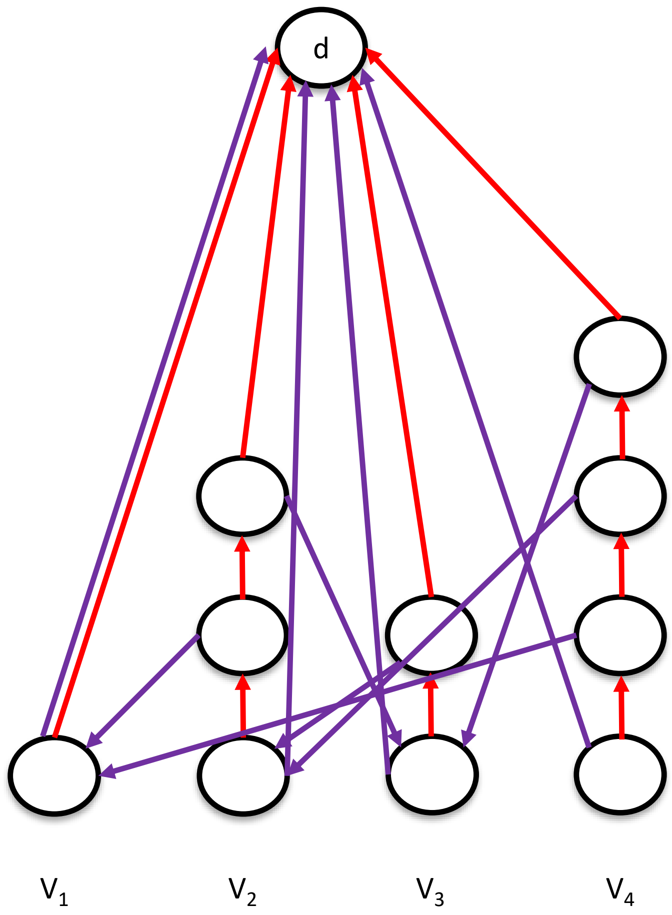


Find maximum update?

- Let’s go more general
- Delete all cycles in a graph
- **NP-hard** to approximate
  - *Feedback Arc Set*
- And it’s (essentially) equivalent ☹

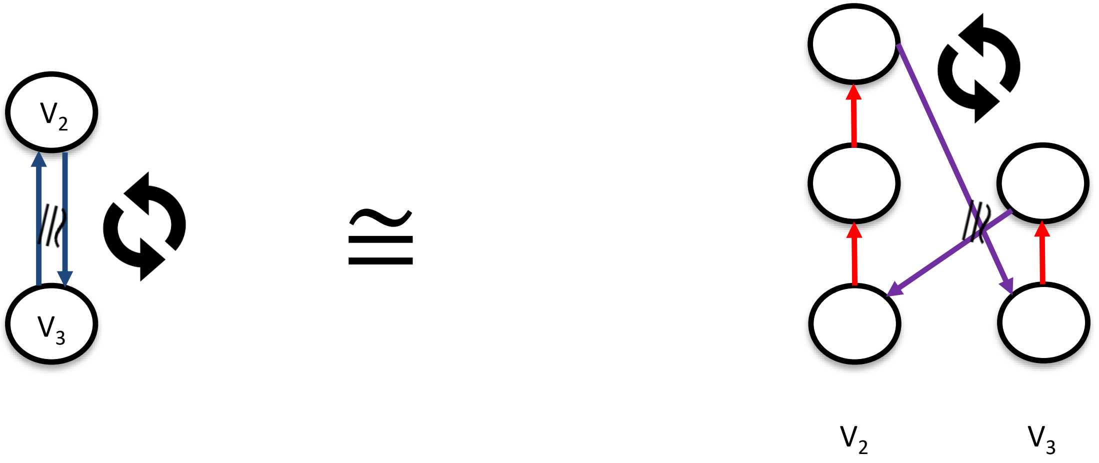


≅



Find maximum update?

- Let's go more general
- Delete all cycles in a graph
- **NP-hard** to approximate
  - *Feedback Arc Set*
- And it's (essentially) equivalent ☹



Also NP-hard for any o(n) for 2-destination policies:
F., Wattenhofer, ICCCN 2016

Greedy? Update as many as possible per round

- Always works ☺
- Maximizing is NP-hard ☹
  - *Transiently Consistent SDN Updates: Being Greedy is Hard*. S. Akhoondian Amiri, A. Ludwig, J. Marcinkowski, S. Schmid. In: SIROCCO 2016
  - *The Power of Two in Consistent Network Updates: Hard Loop Freedom, Easy Flow Migration*. K.-T. Foerster, R. Wattenhofer. In: ICCCN 2016
- Single greedy update: O(1) rounds ⇨ Ω(n) rounds ☹ ☹
  - *Loop-Free Route Updates for Software-Defined Networks*. K.-T. Foerster, A. Ludwig, J. Marcinkowski, S. Schmid. In: IEEE/ACM Trans. Netw. 2018
- In general: Does a 3-round schedule exist? NP-hard ☹ ☹ ☹
  - *Loop-Free Route Updates for Software-Defined Networks*. K.-T. Foerster, A. Ludwig, J. Marcinkowski, S. Schmid. In: IEEE/ACM Trans. Netw. 2018

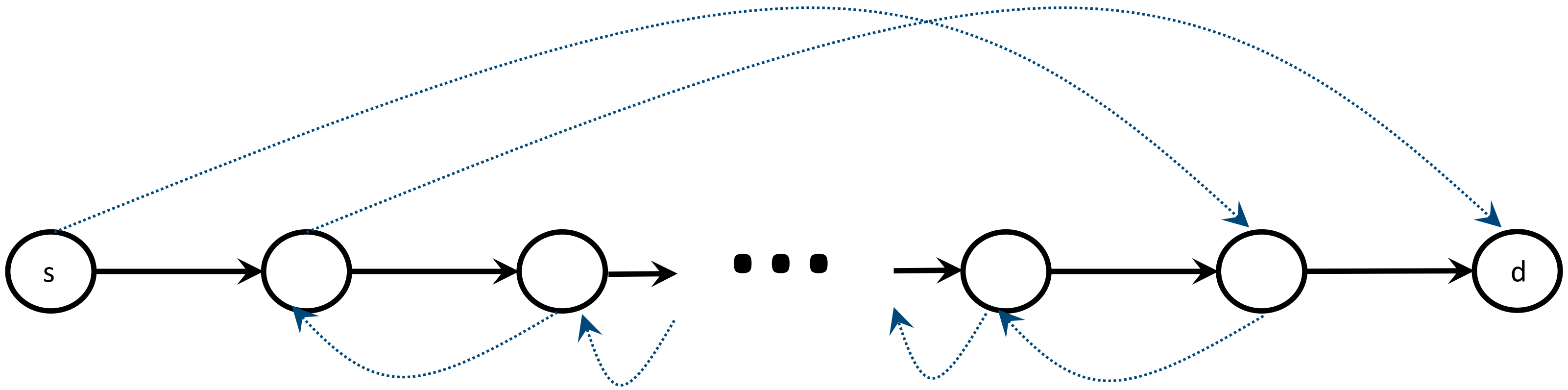

Relax And Take it Easy!
MY BABY FLOAT
INTEX
SURFER

Scheduling Loop-free Network Updates: It's Good to Relax! [Ludwig et al., PODC 2015]

Two key ideas:

1. ~~destination *d* based~~ source-destination pairs <*s*,*d*>
2. ~~no forwarding loops~~ no loops between <*s*,*d*>

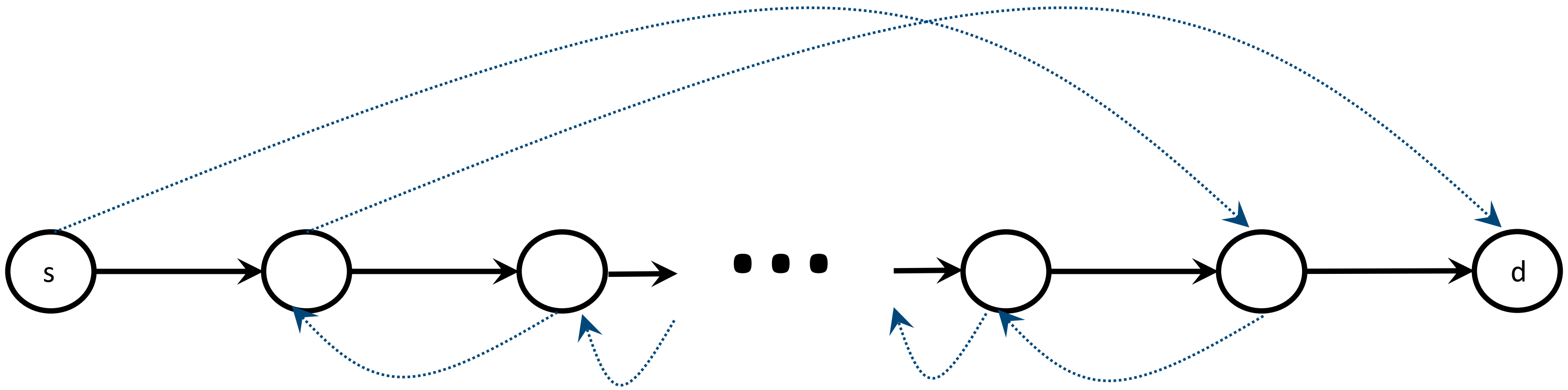
On its own:
Makes 2-round updates polynomial, 3 still NP-hard



Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed?

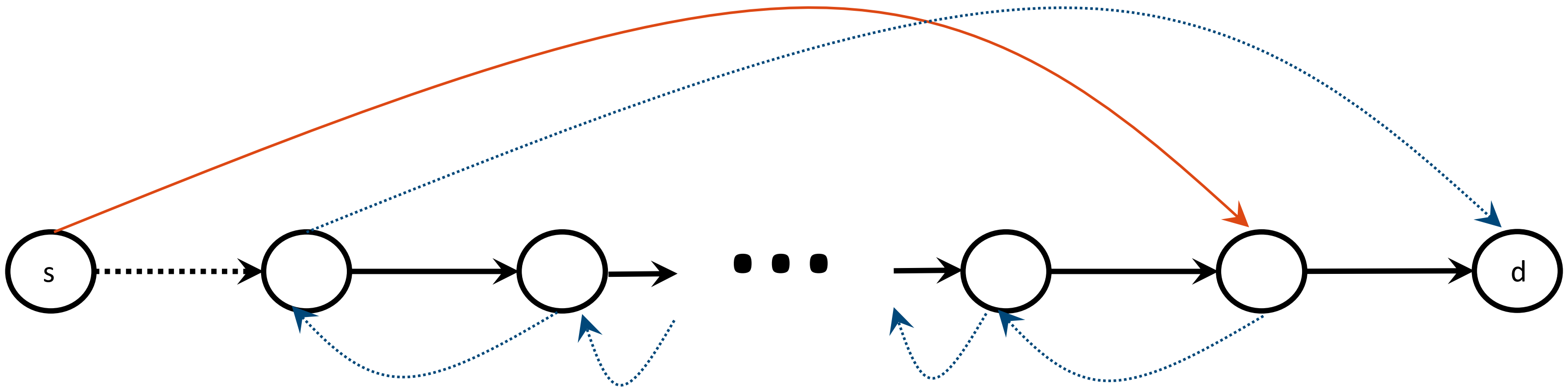


Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed?

Round **1**

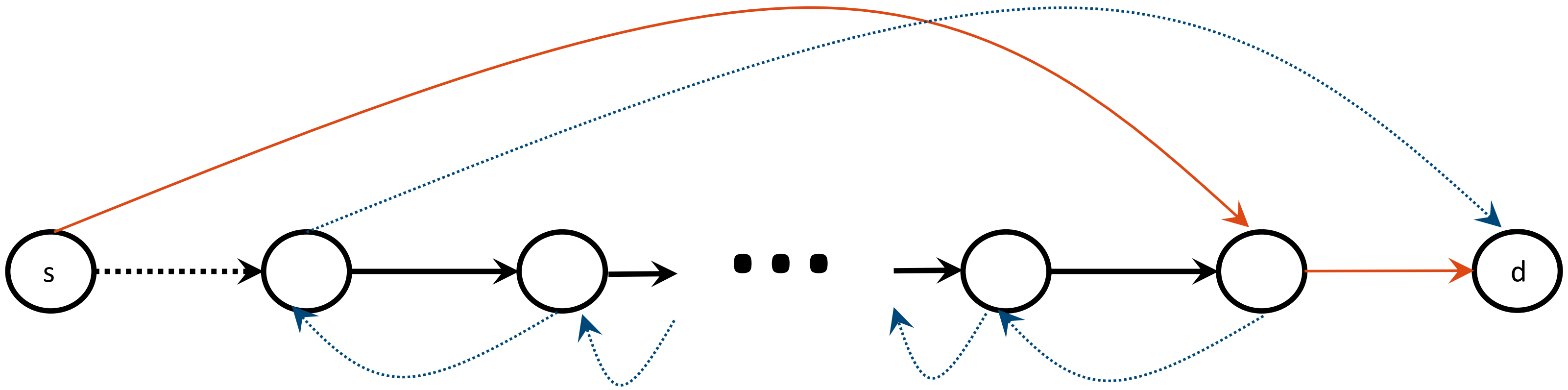


Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed?

Round **1**

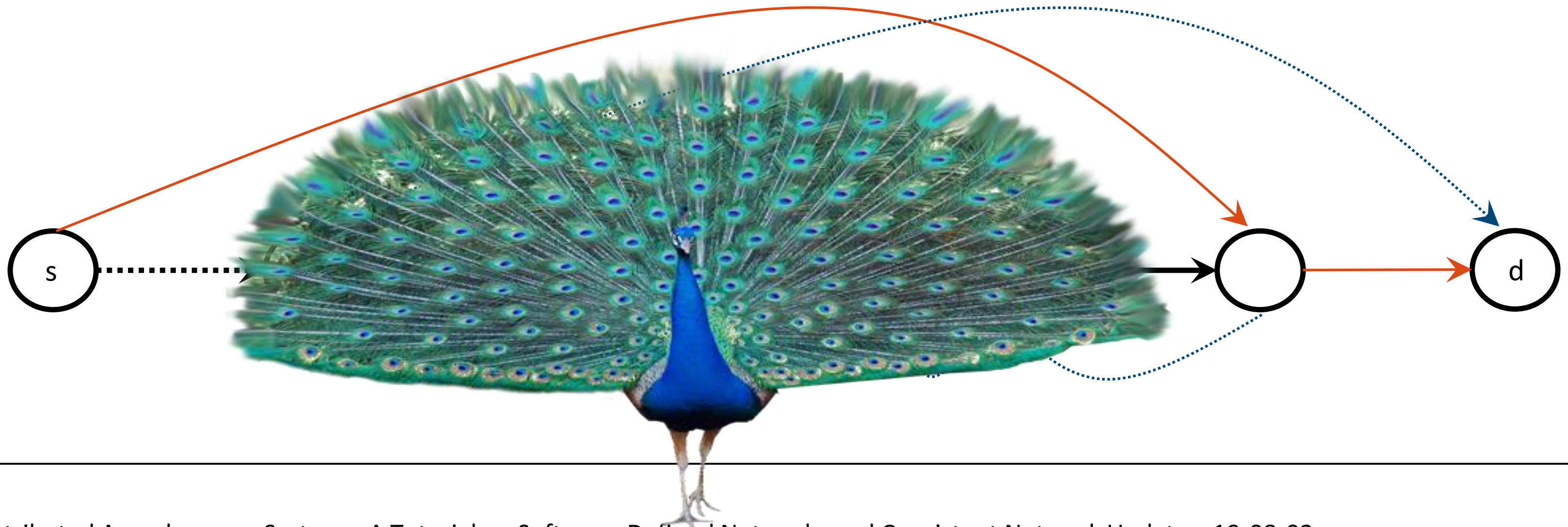


Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed?

Round **1**

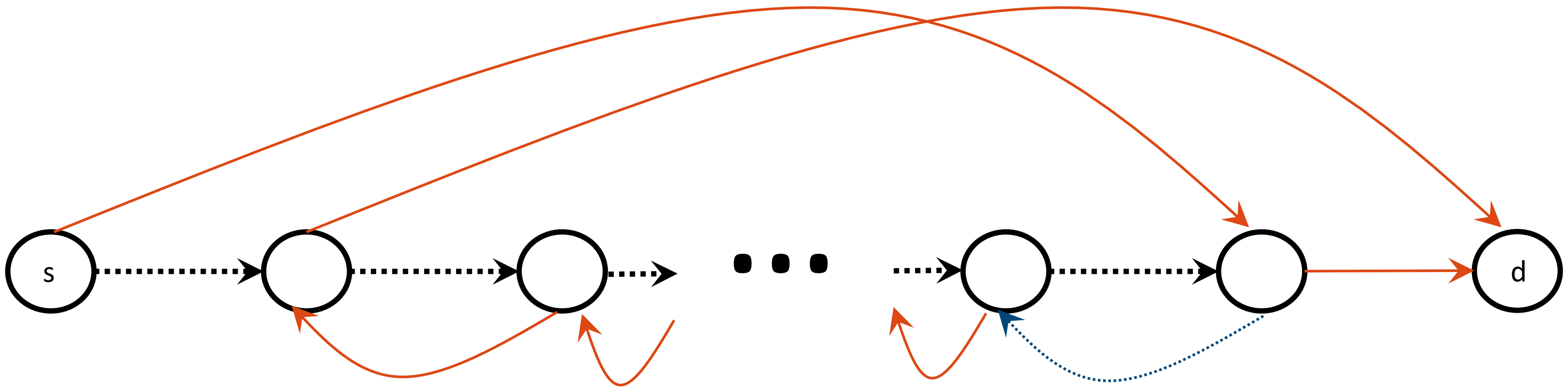


Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed?

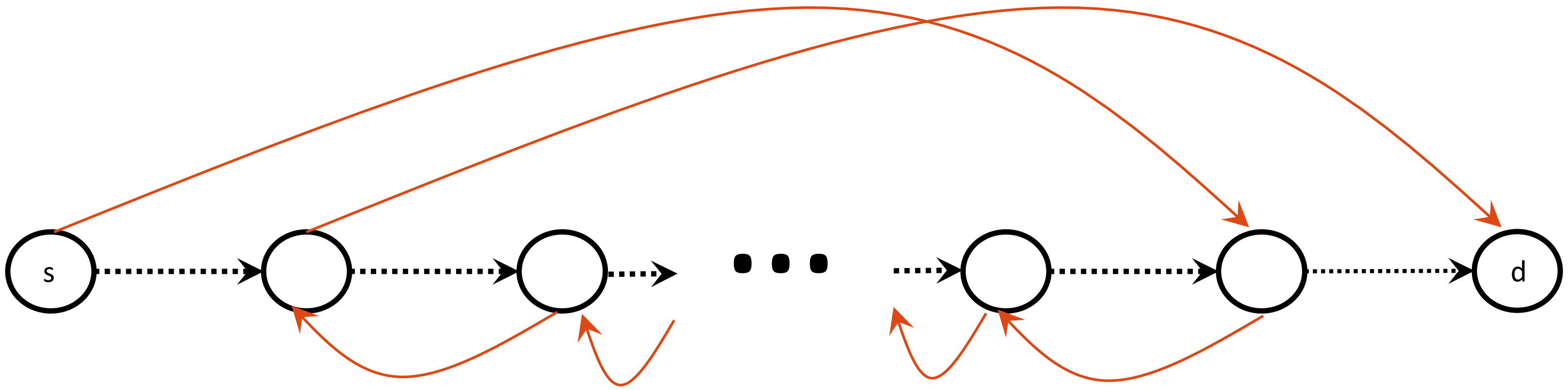
Round **2**



Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds
- Relaxed? Just 3 rounds

Round **3**

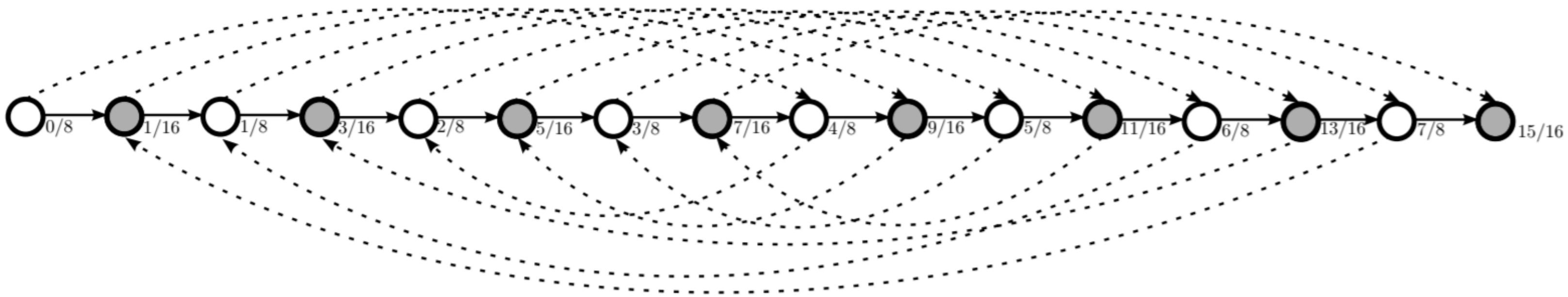


Scheduling Loop-free Network Updates: It's Good to Relax!

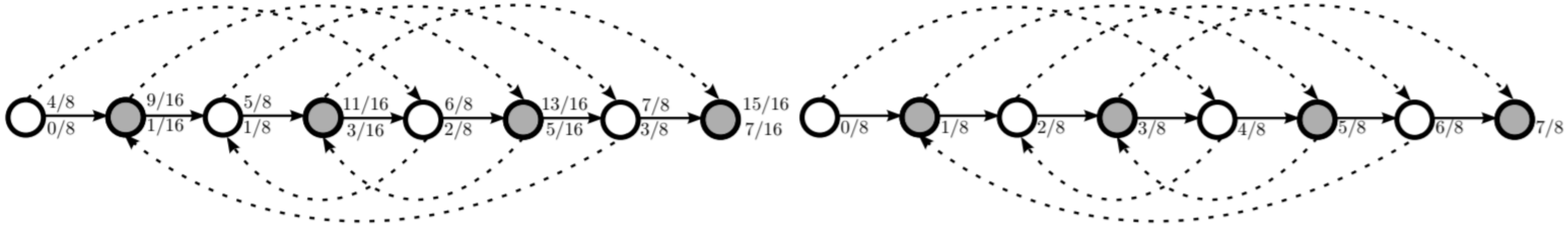
- Non-relaxed? Ω(n) rounds

- Relaxed? Just 3 rounds
  - In general: $O(\log n)$ rounds (“Peacock“)

Loop-Free Route Updates for Software-Defined Networks. K.-T. Foerster, A. Ludwig, J. Marcinkowski, S. Schmid. In: IEEE/ACM Trans. Netw. 2018

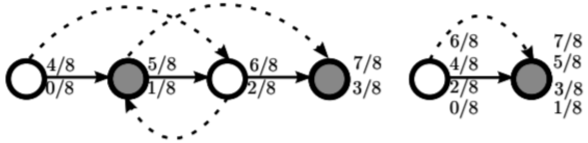


(a) The graph G_1 with 16 nodes. When *Peacock* selects the edge from $0/8$ to $4/8$ as a shortcut, pruning results in the graph in Fig. 10b.

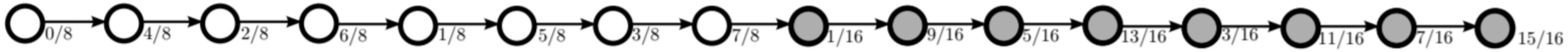


(b) After two rounds with *Peacock*, isomorphic to G_0 in Fig. 10c.

(c) The graph G_0 with 8 nodes. $0/8$ to $4/8$ is the next shortcut.



(d) To the left, the output of *Peacock* on G_0 after two rounds. To the right, after two more rounds, selecting the first forward edge as a shortcut each time.

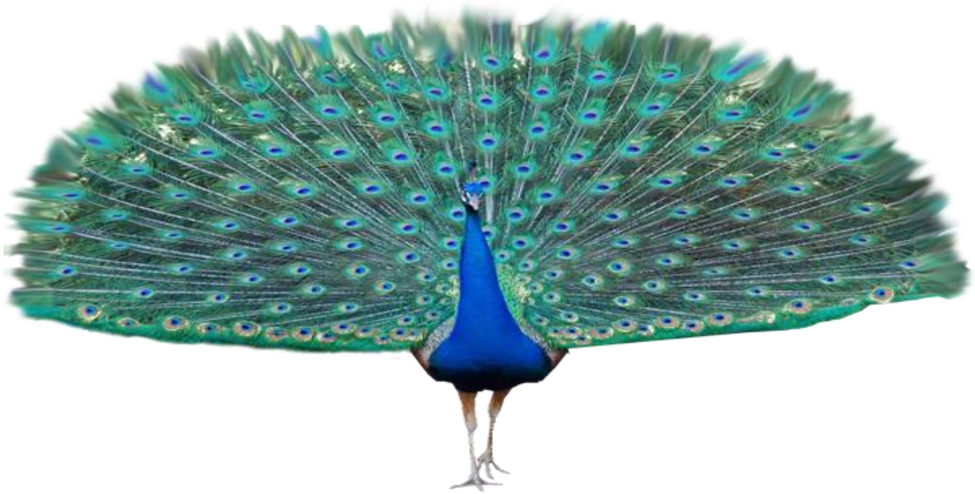


(e) The resulting updated graph, expanded into 16 nodes again.

Scheduling Loop-free Network Updates: It's Good to Relax!

- Non-relaxed? Ω(n) rounds

- Relaxed? Just 3 rounds
  - In general: $O(\log n)$ rounds (“Peacock“)
  - But: Peacock instances with $\Omega(\log n)$ rounds



Loop-Free Route Updates for Software-Defined Networks. K.-T. Foerster, A. Ludwig, J. Marcinkowski, S. Schmid. In: IEEE/ACM Trans. Netw. 2018

Some Open Questions for scheduling loop free updates:

- For both models: Approximation algorithms for #rounds?

In a bit..

Relaxed:

- Optimal #rounds: NP-hard or in P?
- What is the real lower bound?

More open questions and specifics:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019

Non-relaxed:

- NP-hard for $O(1) < k < \Omega(n)$ rounds?

Eg Congestion?
Network functions?

So Far Everything Was Sort of Centralized…

- …can we make it more distributed?

Decentralized Updates for „Tree-Ordering“

- So far: every round:
  - Controller computes and sends out updates
  - Switches implement them and send acks
  - Controller receives acks

Decentralized Updates for „Tree-Ordering“

- So far: every round:
  - Controller computes and sends out updates
  - Switches implement them and send acks
  - Controller receives acks

- Alternative: Use dualism to so-called *proof labeling schemes*

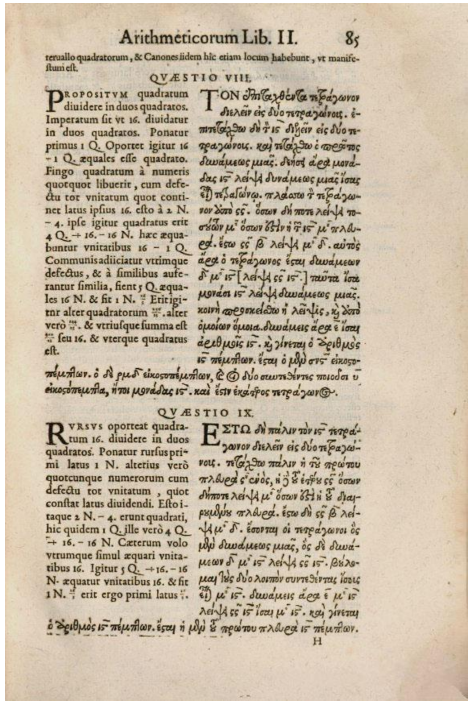
Centralized Controller
(Prover)

Eg P4 switch
(Verifier)

INTERMISSION

Proof-Labeling Schemes

Deciding vs Checking



Arithmeticorum Lib. II. 85

teruallo quadratorum, & Canones iidem hîc etiam locum habebunt, vt manifestum est.

QVÆSTIO VIII.

PROPOSITVM quadratum diuidere in duos quadratos. Imperatum sit vt 16. diuidatur in duos quadratos. Ponatur primus 1 Q. Oportet igitur 16 − 1 Q. æquales esse quadrato. Fingo quadratum à numeris quotquot libuerit, cum defectu tot vnitatum quot continet latus ipsius 16. esto à 2 N. − 4. ipse igitur quadratus erit 4 Q. + 16. − 16 N. hæc æquabuntur vnitatibus 16 − 1 Q. Communis adiiciatur vtrimque defectus, & à similibus auferantur similia, fient 5 Q. æquales 16 N. & fit 1 N. $\frac{16}{5}$ Erit igitur alter quadratorum $\frac{256}{25}$. alter verò $\frac{144}{25}$. & vtriusque summa est $\frac{400}{25}$ seu 16. & vterque quadratus est.

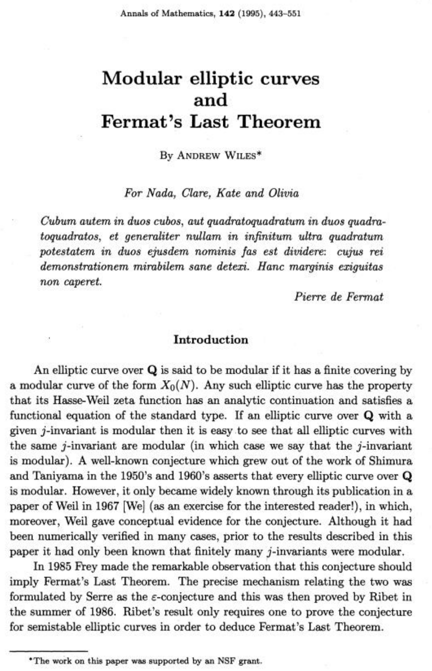
QVÆSTIO IX.

RVRSVS oporteat quadratum 16. diuidere in duos quadratos. Ponatur rursus primi latus 1 N. alterius verò quotcunque numerorum cum defectu tot vnitatum, quot constat latus diuidendi. Esto itaque 2 N. − 4. erunt quadrati, hic quidem 1 Q. ille verò 4 Q. + 16. − 16 N. Cæterum volo vtrumque simul æquari vnitatibus 16. Igitur 5 Q. + 16. − 16 N. æquatur vnitatibus 16. & fit 1 N. [illegible] erit ergo primi latus [illegible].

H

Prove

Verify



Annals of Mathematics, **142** (1995), 443–551

Modular elliptic curves and Fermat's Last Theorem

By ANDREW WILES*

For Nada, Clare, Kate and Olivia

Cubum autem in duos cubos, aut quadratoquadratum in duos quadratoquadratos, et generaliter nullam in infinitum ultra quadratum potestatem in duos ejusdem nominis fas est dividere: cujus rei demonstrationem mirabilem sane detexi. Hanc marginis exiguitas non caperet.

Pierre de Fermat

Introduction

An elliptic curve over **Q** is said to be modular if it has a finite covering by a modular curve of the form $X_0(N)$. Any such elliptic curve has the property that its Hasse-Weil zeta function has an analytic continuation and satisfies a functional equation of the standard type. If an elliptic curve over **Q** with a given j-invariant is modular then it is easy to see that all elliptic curves with the same j-invariant are modular (in which case we say that the j-invariant is modular). A well-known conjecture which grew out of the work of Shimura and Taniyama in the 1950's and 1960's asserts that every elliptic curve over **Q** is modular. However, it only became widely known through its publication in a paper of Weil in 1967 [We] (as an exercise for the interested reader!), in which, moreover, Weil gave conceptual evidence for the conjecture. Although it had been numerically verified in many cases, prior to the results described in this paper it had only been known that finitely many j-invariants were modular.

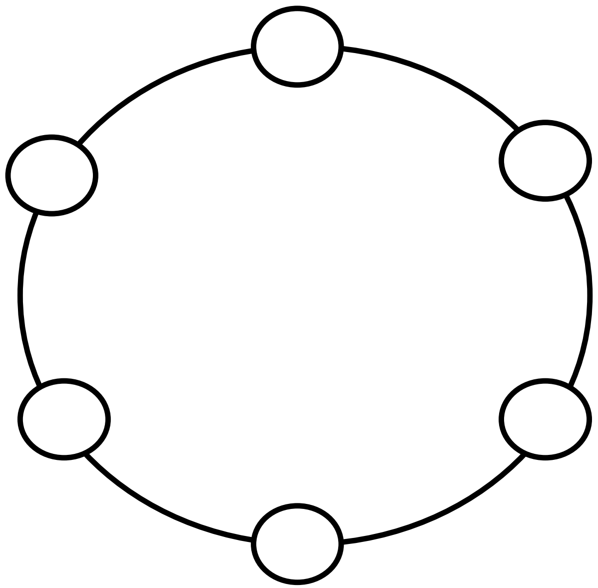
In 1985 Frey made the remarkable observation that this conjecture should imply Fermat's Last Theorem. The precise mechanism relating the two was formulated by Serre as the ε-conjecture and this was then proved by Ribet in the summer of 1986. Ribet's result only requires one to prove the conjecture for semistable elliptic curves in order to deduce Fermat's Last Theorem.

*The work on this paper was supported by an NSF grant.

Brief Selected Background

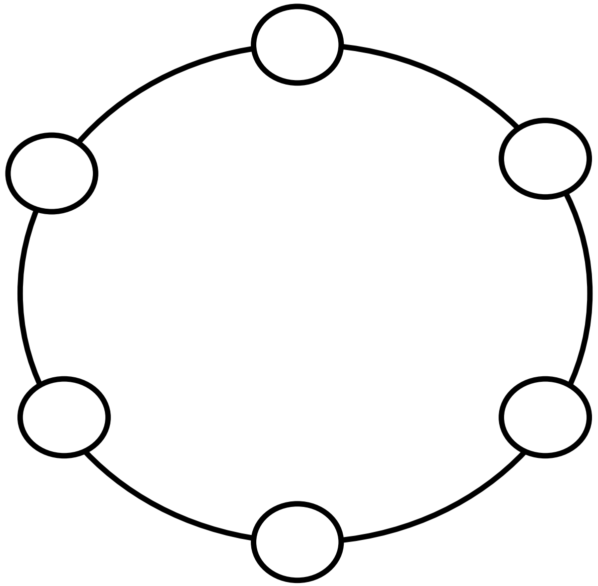
- [Naor and Stockmeyer, STOC 1993]:
  What can be computed locally?
- [Korman et al., PODC 2005]:
  Proof Labeling Schemes (PLS)
- [Göös and Suomela, PODC 2011]:
  Locally Checkable Proofs (LCP)
- [Fraigniaud et al., FOCS 2011,…]:
  Nondeterministic Local Decision (NLD)
- And many more recent works, e.g., on approximation, randomization etc.

Example



universität wien

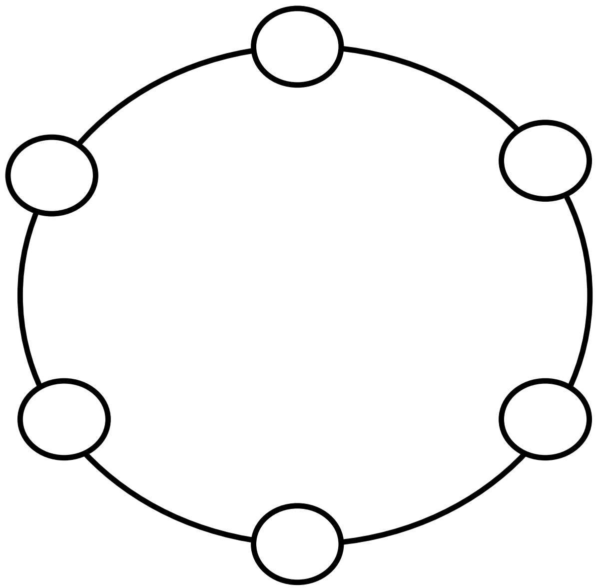
Example



Model

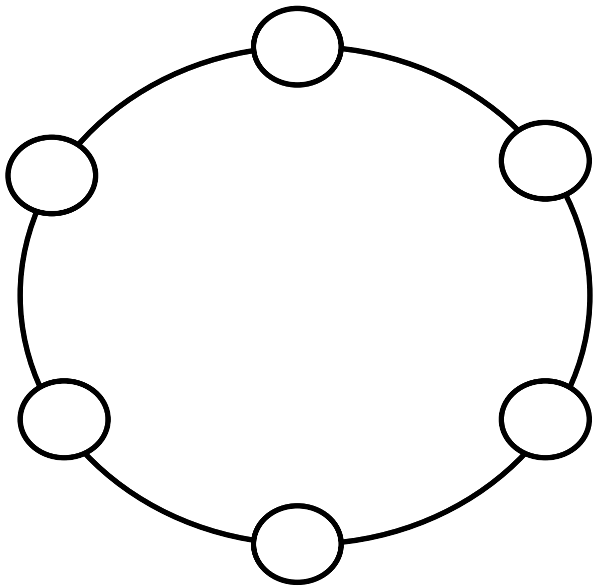
- Each of the n nodes is a computer, connected by links
- Synchronous rounds
  - Simplified: unlimited message size & computational power, unique identifiers for nodes

Example



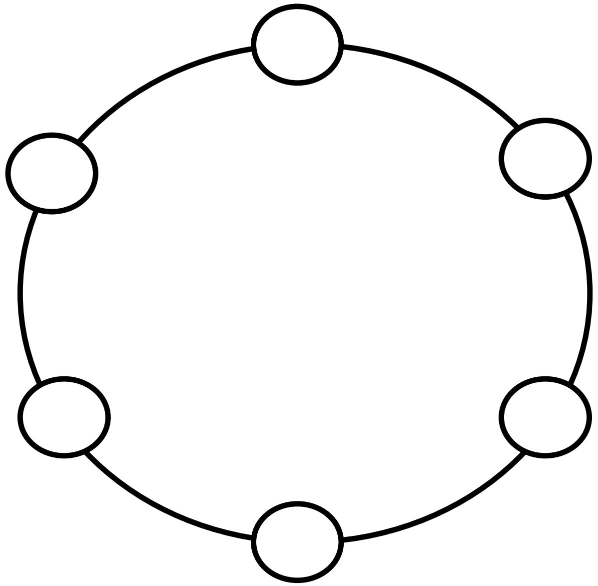
- Is n even?

Example



- Is n even?
- $\Omega(n)$ rounds

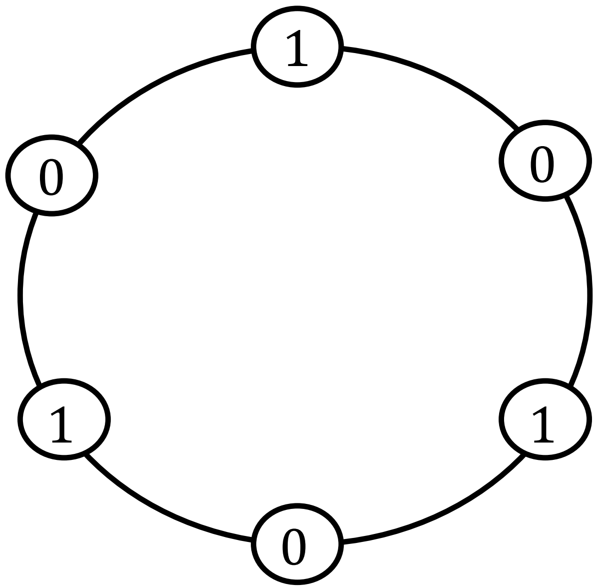
Example



- Is n even?
- $\Omega(n)$ rounds
- What if I tell you it is even? Why should you trust me ☺

universität wien

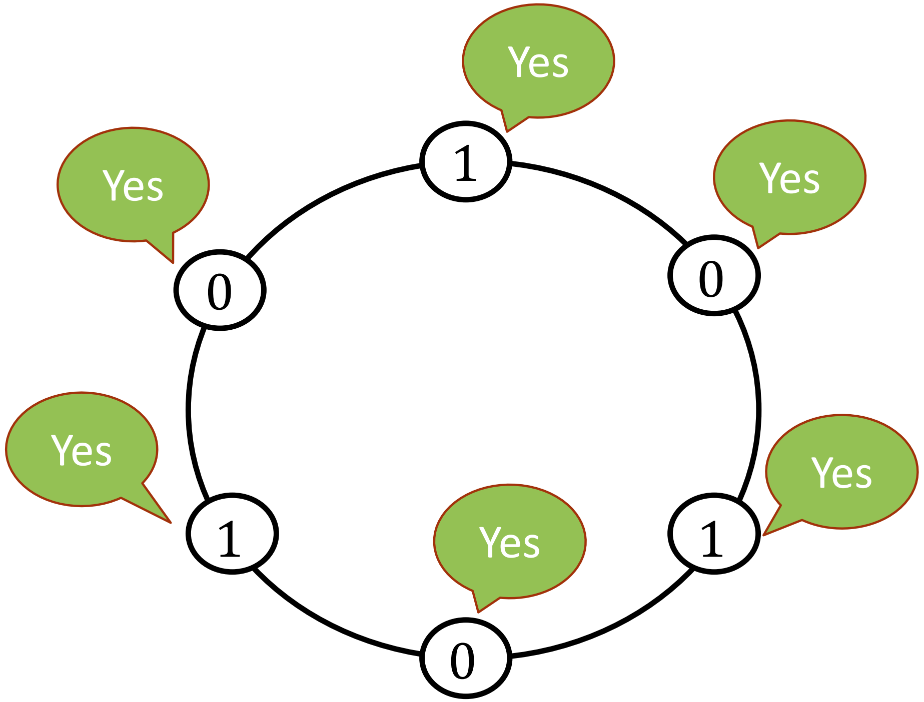
Example



- Is n even?
- $\Omega(n)$ rounds
- $\mathcal{P}$rover assigns 1 bit?

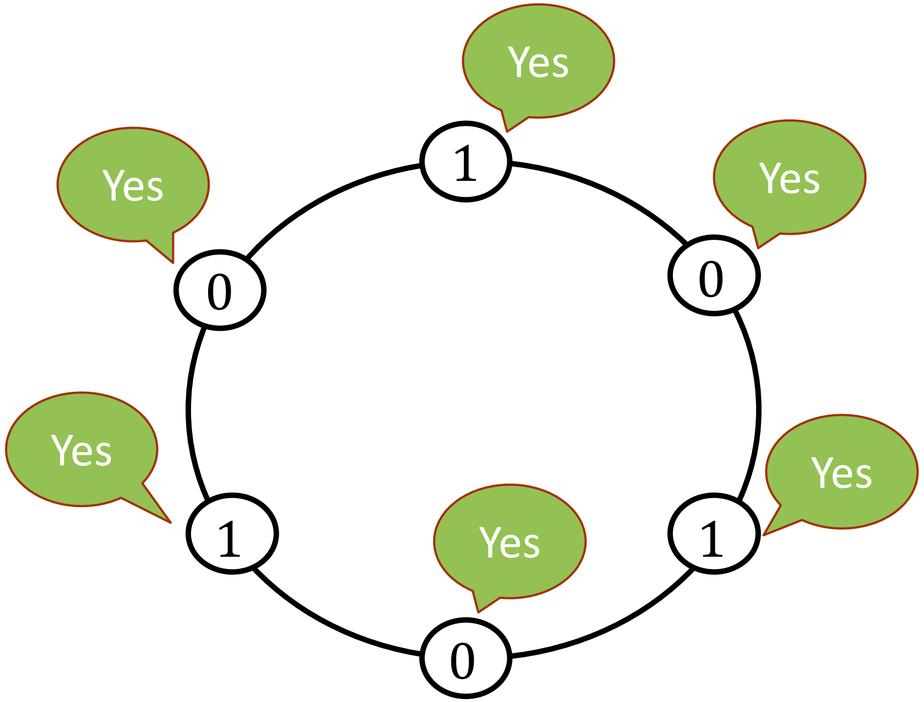
universität wien

Example



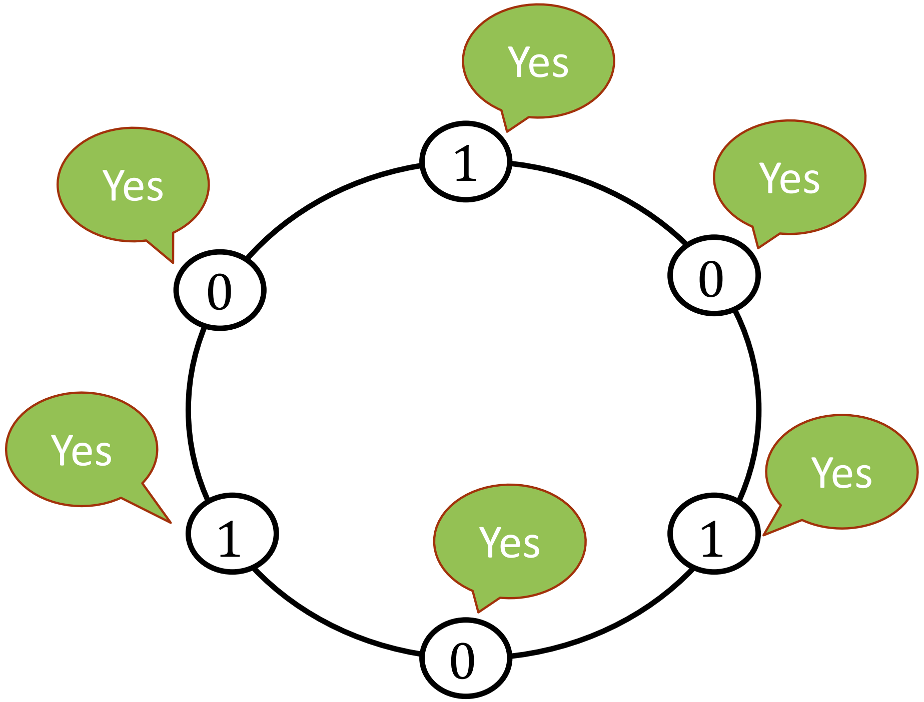
- Is n even?
- $\Omega(n)$ rounds
- $\mathcal{P}$rover assigns 1 bit -> $\mathcal{V}$erify in 1 round

Example



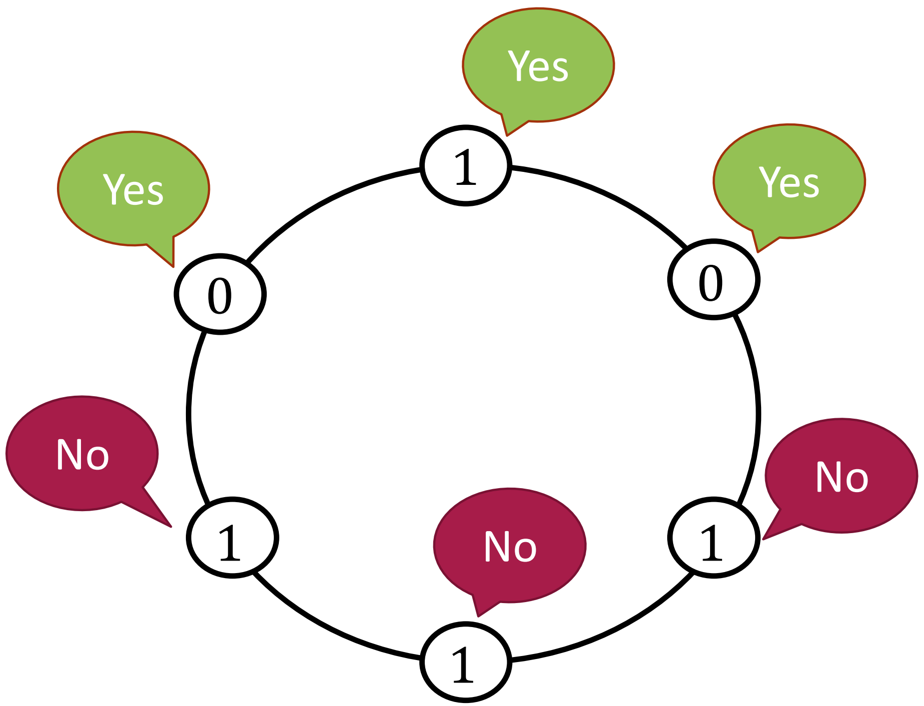
- Is n even?
- $\Omega(n)$ rounds
- $\mathcal{P}$rover assigns 1 bit -> $\mathcal{V}$erify in 1 round
- Other way to think of it: 1 bit of non-determinism
- General question: How many bits necessary/sufficient?

Accepting a proof



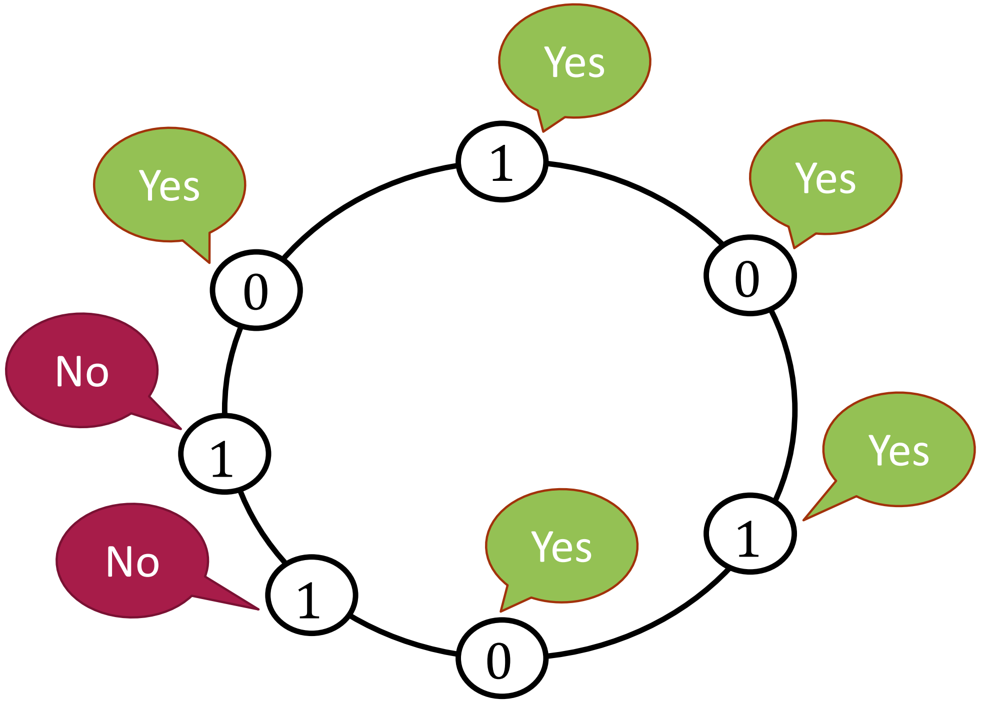
- Every node outputs **Yes** -> Proof accepted
- One node outputs **No** -> Proof rejected

Accepting a proof



- Every node outputs **Yes** -> Proof accepted
- One node outputs **No** -> Proof rejected
  - $\mathcal{P}$rover chose the wrong proof

Accepting a proof



- Every node outputs **Yes** -> Proof accepted
- One node outputs **No** -> Proof rejected
  - $\mathcal{P}$rover chose the wrong proof
  - Property does not hold

Back to SDNs: Switch from a proof to another

universität wien

Decentralized ing“

When should I update?

Decentralized ing“

Once my parent updates!

Decentralized ing“

Once my parent updates!

Send parent ID

Decentralized Updates for „Tree-Ordering“

Decentralized … ing“

I‘ll update too!

I updated

Decentralized Updates for “Tree-Ordering“

+ Only one controller-switch interaction per route change
+ New route changes can be pushed before old ones done *(include “version#”)*
+ Incorrect updates can be locally detected *(include depth in tree, prevents loops)*

+/- Speed benefit/penalty depends on update scenario and topology

\- Requires switch-to-switch communication e.g., [Nguyen et al., SOSR 2017]

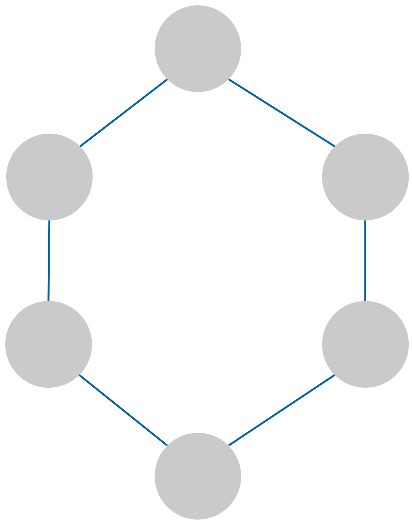
K.-T. Foerster, T. Luedi, J. Seidel, R. Wattenhofer: *Local Checkability, No Strings Attached: (A)cyclicity, Reachability, Loop Free Updates in SDNs* . In: Theoret. Comput. Sci. 2018
K.-T. Foerster, S. Schmid: *Distributed Consistent Network Updates in SDNs: Local Verification for Global Guarantees*. Under submission.

Can we also make the initial computation decentralized?

- Classic setting of distributed computing (e.g. LOCAL or CONGEST model)
  - Possible benefit in SDNs:
    - We do not need to compute from scratch!
      - In wired networks, problems depend on a subset of the network
        - Leverage **Preprocessing**

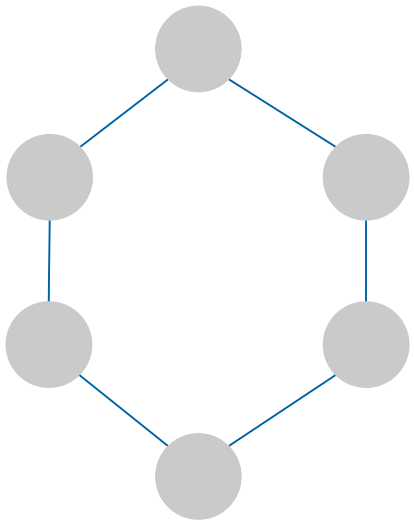
- Further explored in eg:
  - Exploiting Locality in Distributed SDN Control. S. Schmid, J. Suomela, HotSDN 2013
  - On the Power of Preprocessing in Decentralized Network Optimization. K.-T. Foerster, J. Hirvonen, S. Schmid, J. Suomela, INFOCOM 2019
  - BA: Does Preprocessing help under Congestion? K.-T. Foerster, J. Korhonen, J. Rybicki, S. Schmid, PODC 2019

Coloring of rings (LOCAL model)



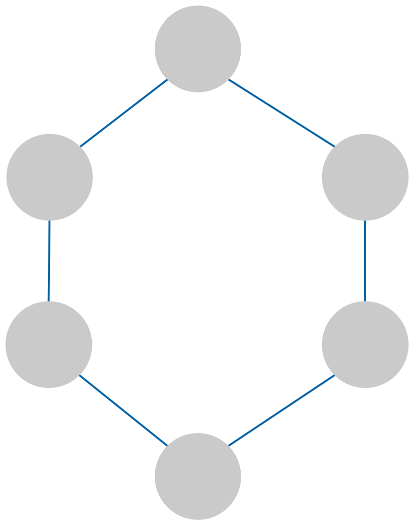
Coloring of rings (LOCAL model)

- 2-coloring:



Coloring of rings (LOCAL model)

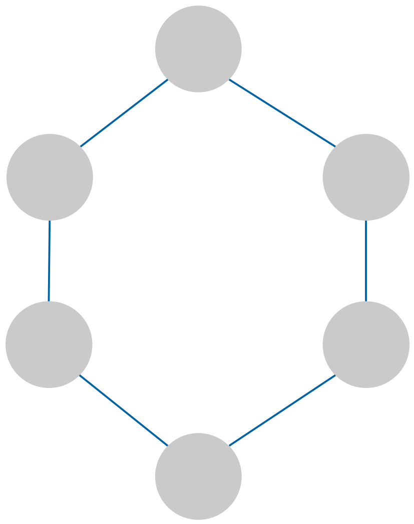
- 2-coloring:
  - Needs $\Omega(n)$ rounds



Coloring of rings (LOCAL model)

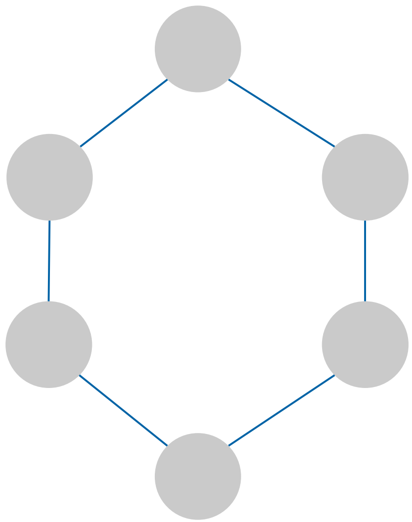
- 2-coloring:
  - Needs $\Omega(n)$ rounds

- 3-coloring:



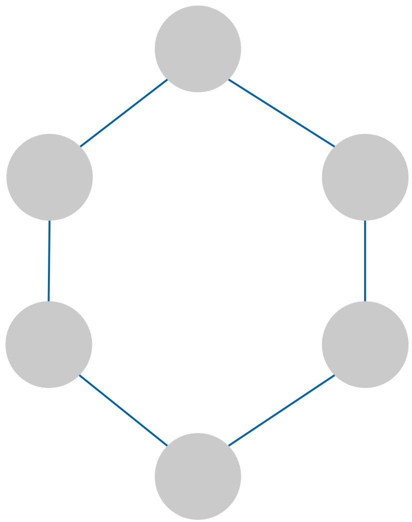
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs $\Omega(n)$ rounds
- 3-coloring:
  - Needs non-constant time



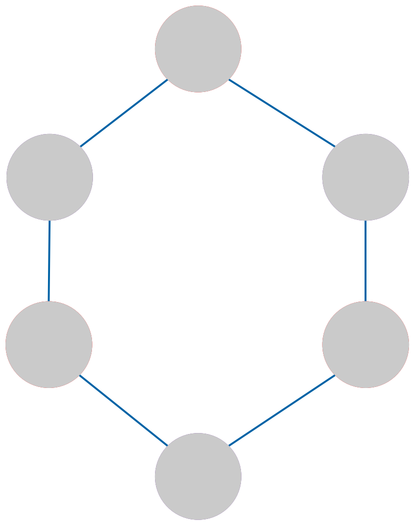
Coloring of rings (LOCAL model)

- 2-coloring:
  - Needs Ω(n) rounds
- 3-coloring:
  - Needs non-constant time
- Cannot improve in the LOCAL model ☹



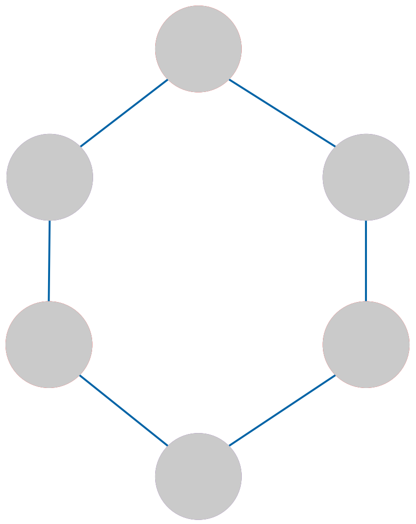
Coloring of rings (LOCAL model) – with Preprocessing

- 2-coloring:
- 3-coloring:



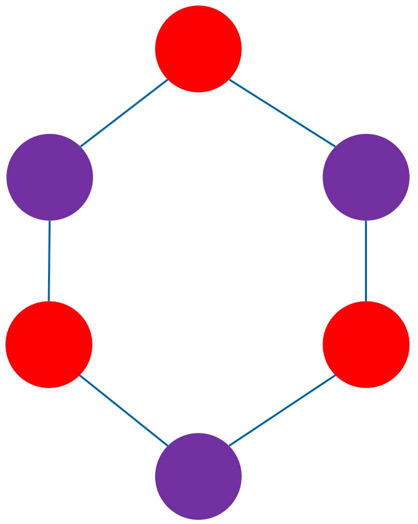
Coloring of rings (LOCAL model) – with Preprocessing

- 2-coloring:
  - 0 rounds ☺
- 3-coloring:
  - 0 rounds ☺



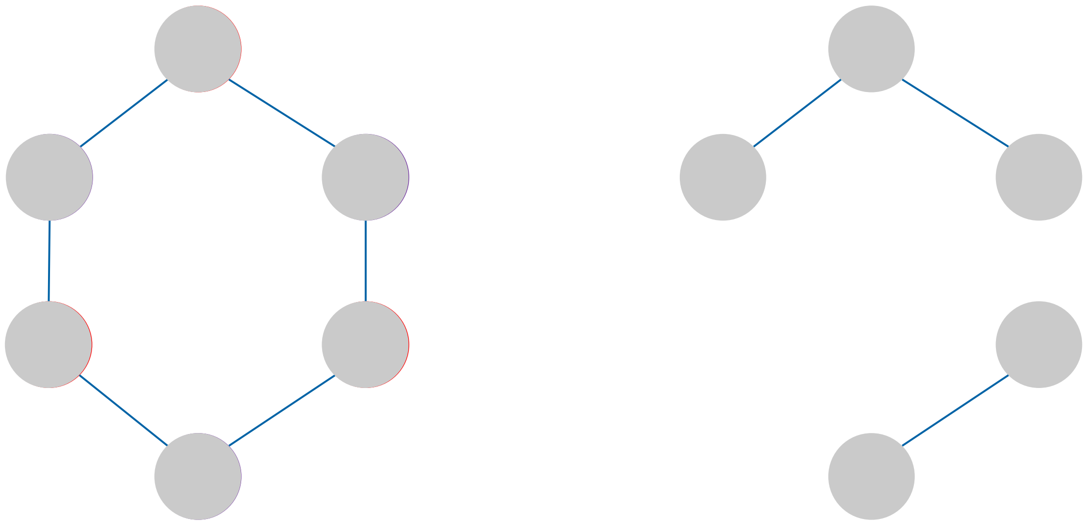
Coloring of rings (LOCAL model) – with Preprocessing

- 2-coloring:
  - 0 rounds ☺
- 3-coloring:
  - 0 rounds ☺



Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

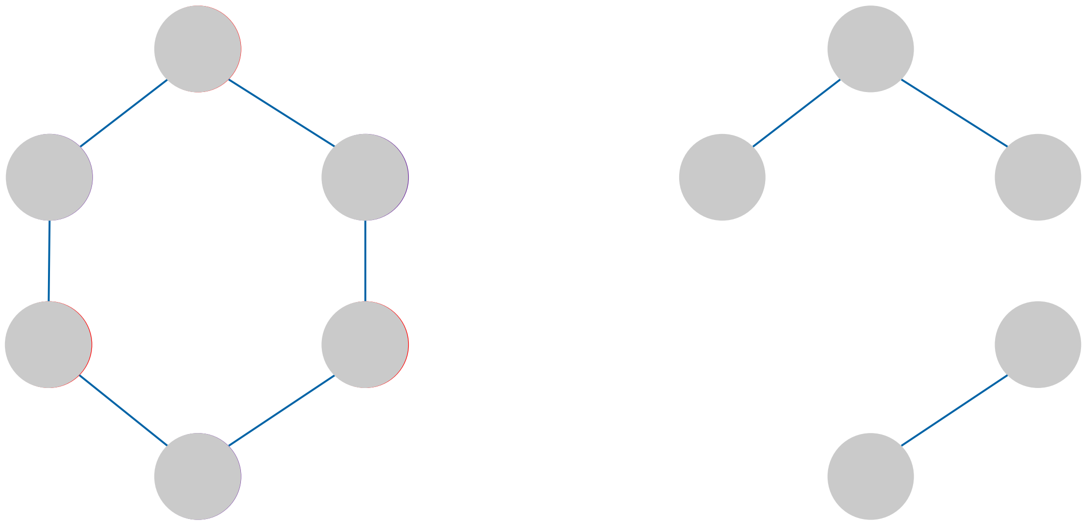
- How about a coloring of a subgraph?



Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

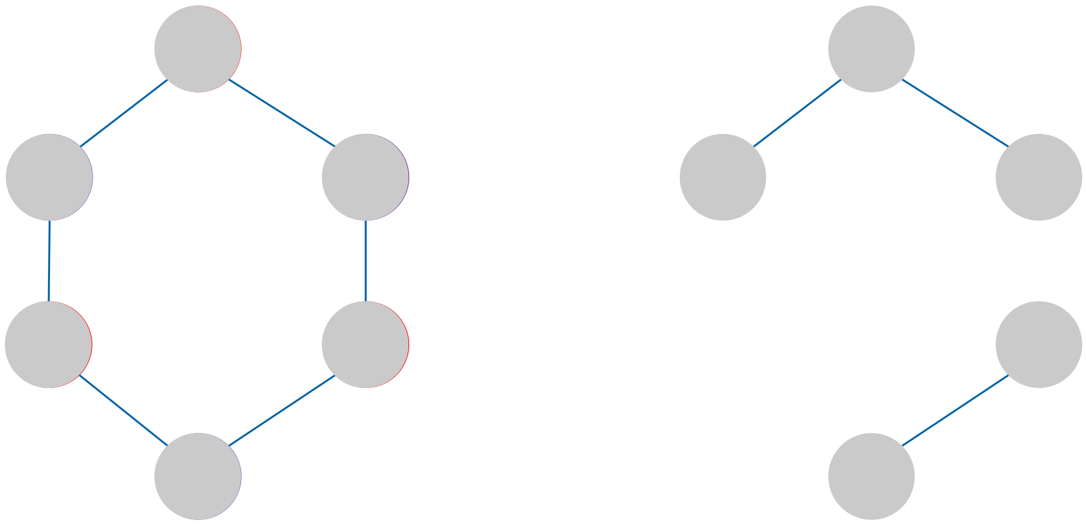
- How about a coloring of a subgraph?

- Local model: runtime does not change



Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!

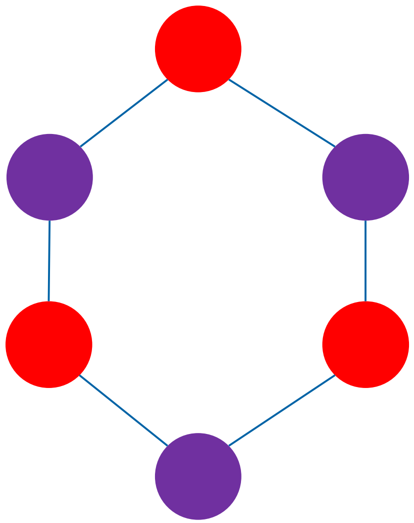


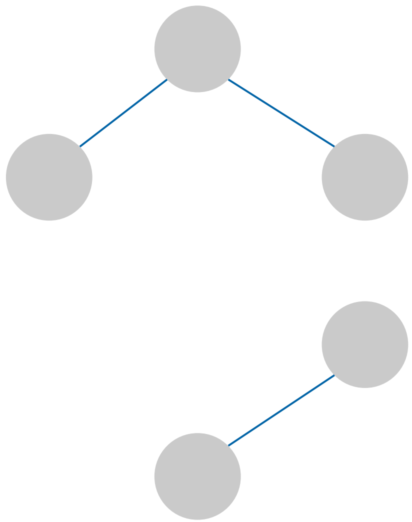
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?

- Local model: runtime does not change

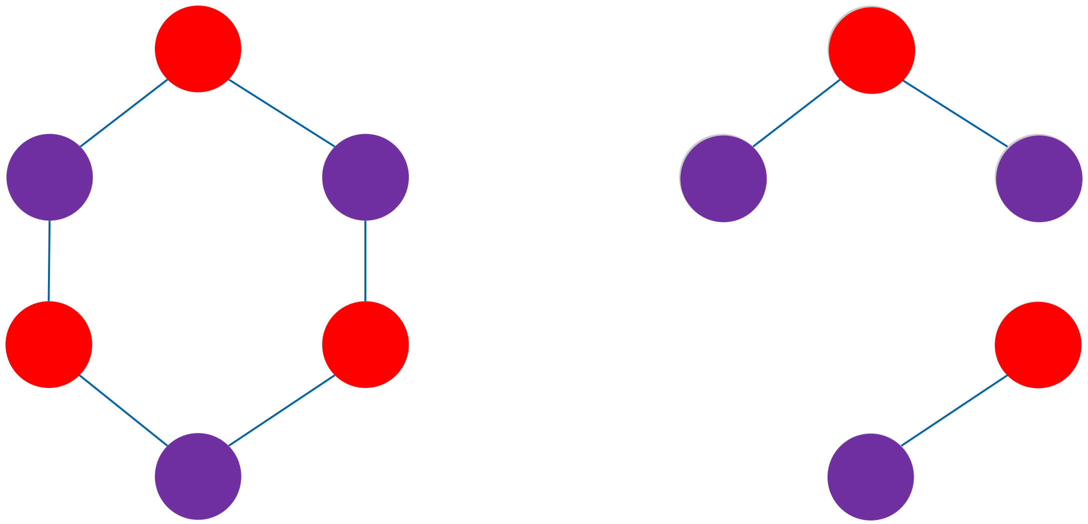
- With preprocessing: fast!





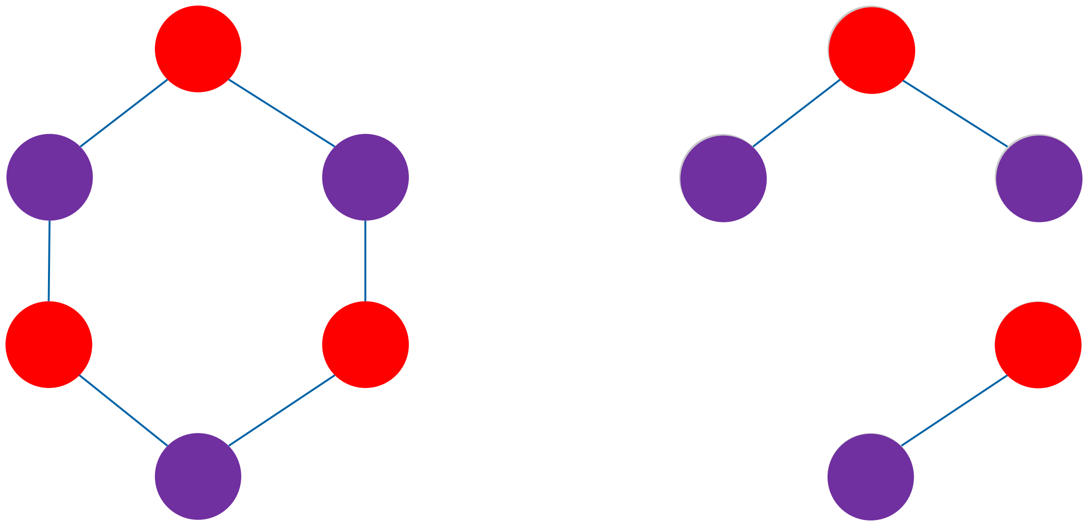
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!



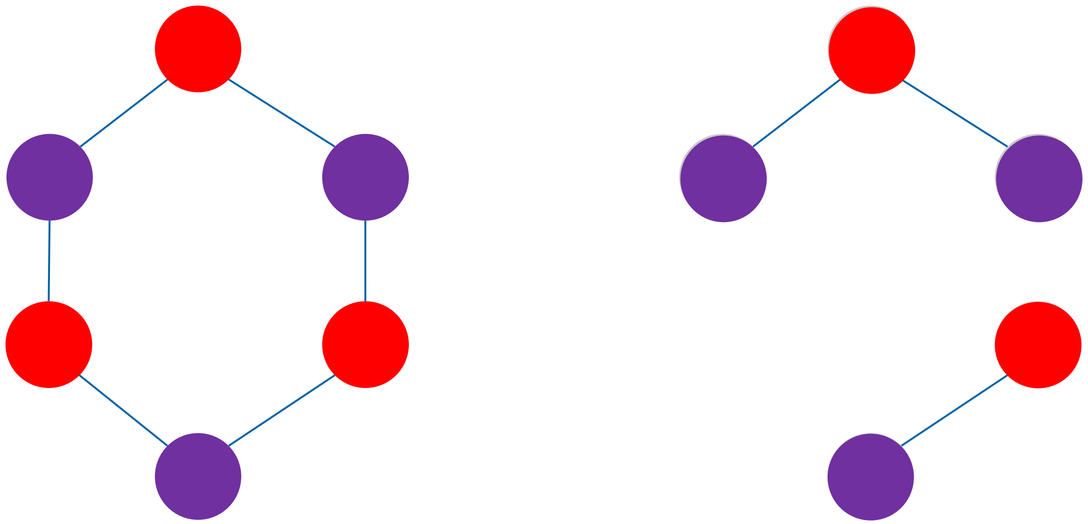
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!
  - Coloring remains valid



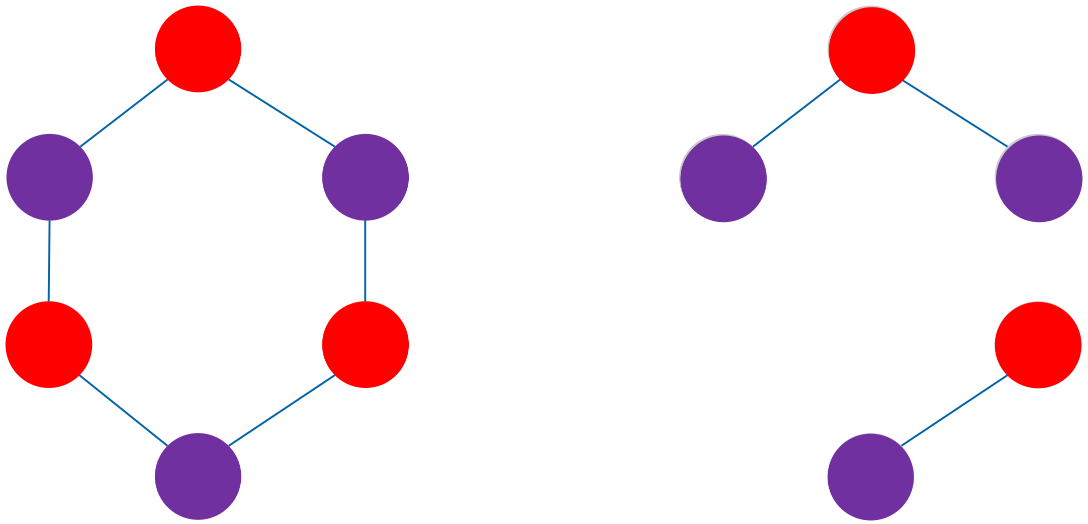
Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!
  - Coloring remains valid
- What are further application scenarios?



Coloring of rings (LOCAL model) – with Preprocessing & Subgraphs

- How about a coloring of a subgraph?
- Local model: runtime does not change
- With preprocessing: fast!
  - Coloring remains valid
- What are further application scenarios?
- What else can we do with the SUPPORT of Preprocessing?



Practical Motivation for Preprocessing

Practical Motivation for Preprocessing

- Decentralization aids scalability

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  - Unknown network state too strong assumption for many scenarios

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  - Unknown network state too strong assumption for many scenarios
  - Often we just react to events, physical topology in wired networks does not grow suddenly

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms
- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  - Unknown network state too strong assumption for many scenarios
  - Often we just react to events, physical topology in wired networks does not grow suddenly
- Example: Software-Defined Networking, single (logically centralized) controller does not scale

Practical Motivation for Preprocessing

- Decentralization aids scalability
  - But: Many problems are not “local” (e.g., coloring)
    - Spanning tree, shortest path, minimizing congestion, good optimization algorithms

- Preprocessing helps scalability (e.g., breaking symmetries ahead of time)
  - Unknown network state too strong assumption for many scenarios
  - Often we just react to events, physical topology in wired networks does not grow suddenly

- Example: Software-Defined Networking, single (logically centralized) controller does not scale
  - Create many local controllers that can react quickly, that control small set of “dumb” nodes

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

universität wien

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

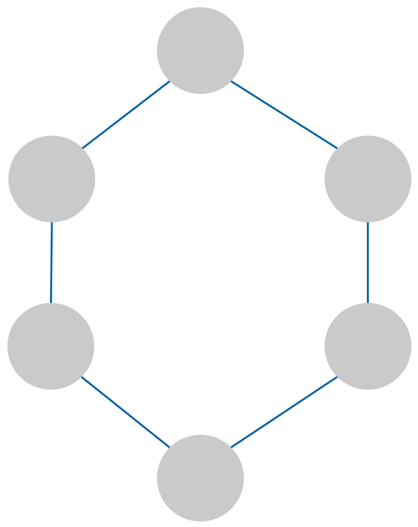
E.g. MAC-address

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))



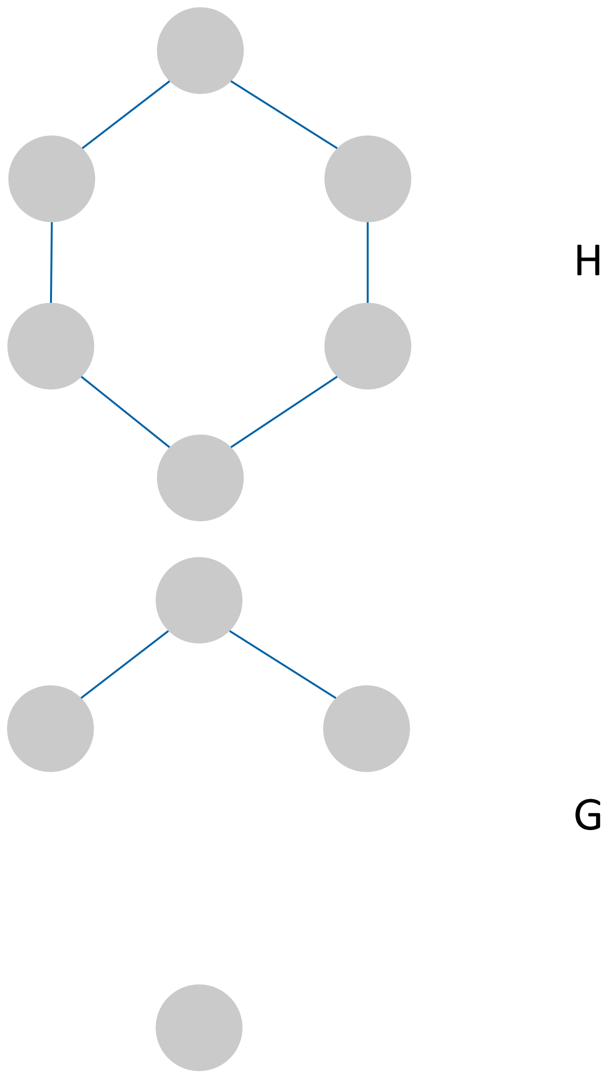
H

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H



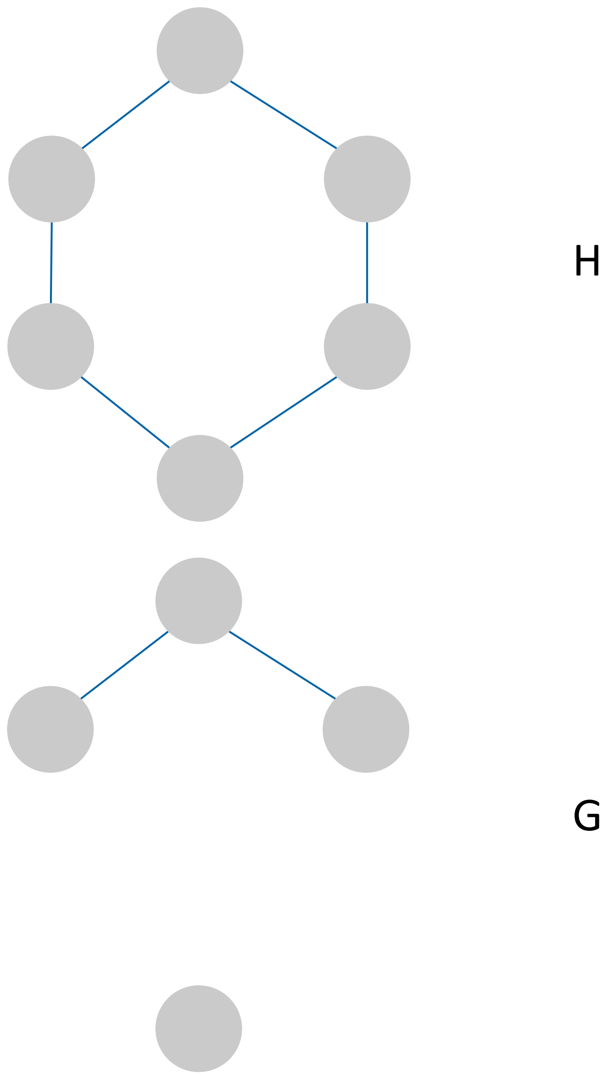
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H

- Two phases:



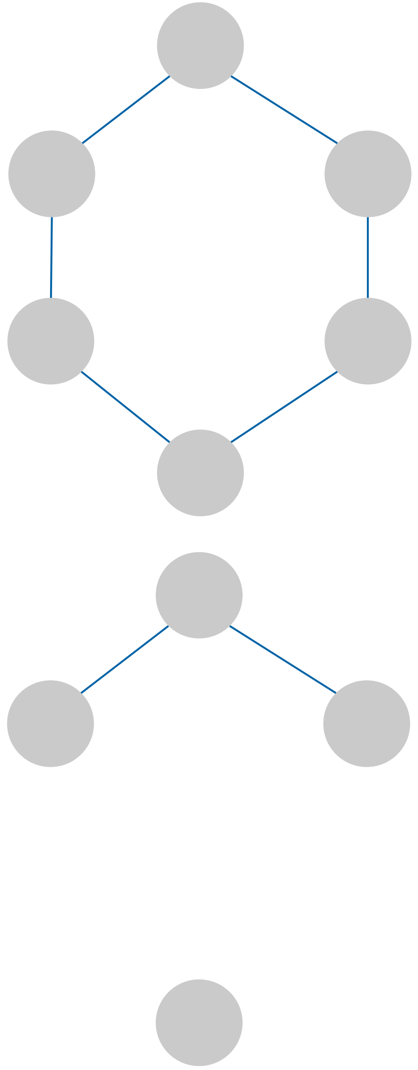
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H
- Two phases:

1. Preprocessing: compute any function on H and store output locally



H

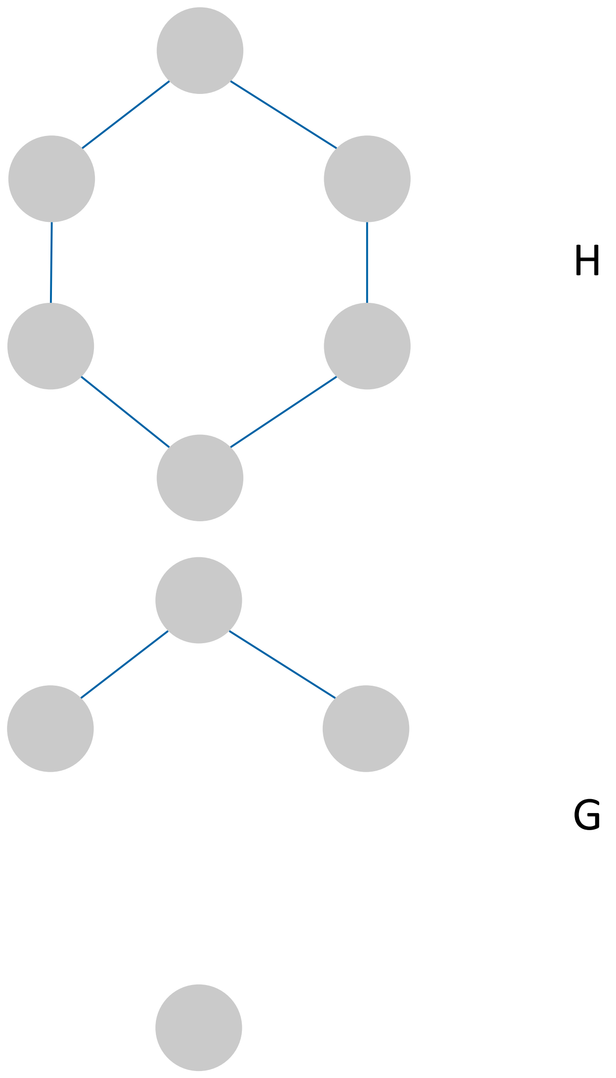
G

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H



- Two phases:
  1. Preprocessing: compute any function on H and store output locally
  2. Solve problem on G in LOCAL model with preprocessed outputs

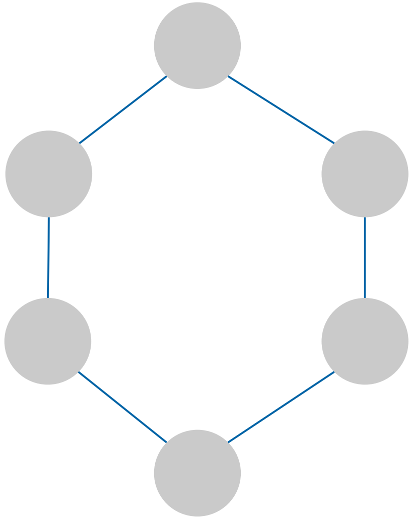
The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

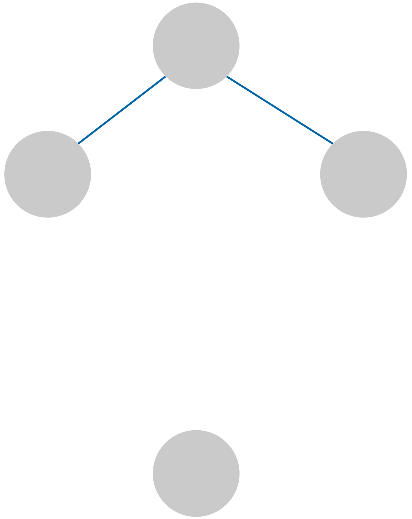
E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H

- Two phases:
  1. Preprocessing: compute any function on H and store output locally
  2. Solve problem on G in LOCAL model with preprocessed outputs
     - Runtime: Number of t rounds in (2), denoted as SUPPORTED(t)



H



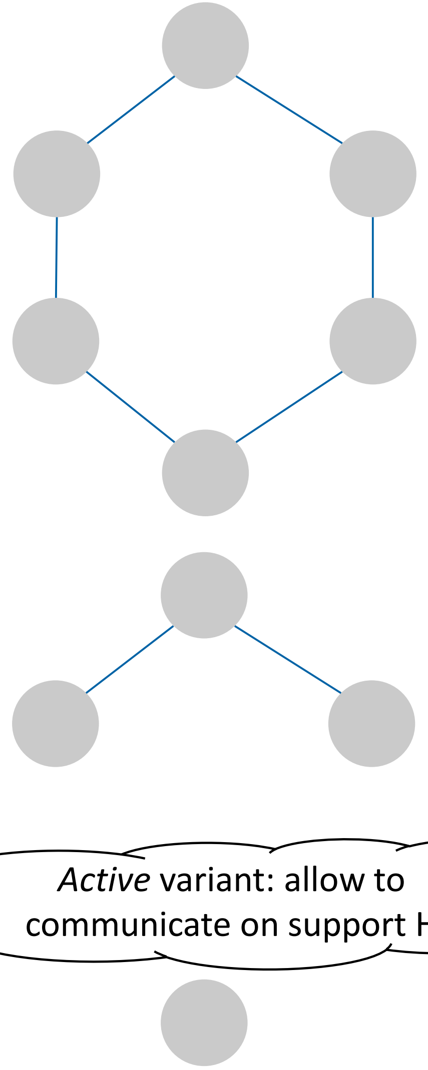
G

The SUPPORTED Model

- Extends the LOCAL model (w. unique IDs) with preprocessing

E.g. MAC-address

- Original structure given as the SUPPORT graph H=(V(H),E(H))
- Problem instance is a subgraph G=(V,E) of H



H

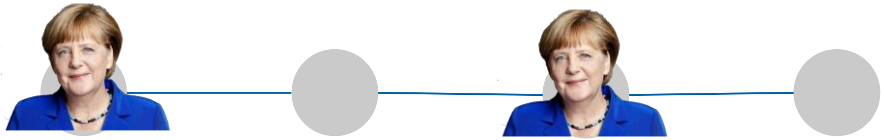
G

- Two phases:
  1. Preprocessing: compute any function on H and store output locally
  2. Solve problem on G in LOCAL model with preprocessed outputs
     - Runtime: Number of t rounds in (2), denoted as SUPPORTED(t)

Active variant: allow to communicate on support H

Does the SUPPORTED Model make everything easy?

- Task: Leader election (Θ(diameter) runtime in LOCAL model)
  - Easy if G=H: precompute leader, 0 rounds
  - But for different G:
    - We need to compute a leader for each connected component of G!
      - Component has no leader? Re-elect ☹
      - Component has multiple leaders? Re-elect ☹
      - Components can have asymptotically same diameter ☹



- SUPPORTED model does not provide a “silver bullet”
  - Not even for the *active* variant

Maybe even useless in general?

Maybe even useless in general?

- Let the support graph H be a complete graph

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size…?

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size…?
  - Problem: G can be arbitrary

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size…?
  - Problem: G can be arbitrary

- For example, if a SUPPORTED algorithm has polylogarithmic runtime
  - ∃ LOCAL algorithm with constant factor overhead

universität wien

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size…?
  - Problem: G can be arbitrary

- For example, if a SUPPORTED algorithm has polylogarithmic runtime
  - ∃ LOCAL algorithm with constant factor overhead

Idea: simulate that support graph H is a complete graph

Maybe even useless in general?

- Let the support graph H be a complete graph
- What sort of meaningful information (for G) can we precompute?
  - Upper bound on ID-space / network size…?
  - Problem: G can be arbitrary

In *active* model:
Congested Clique

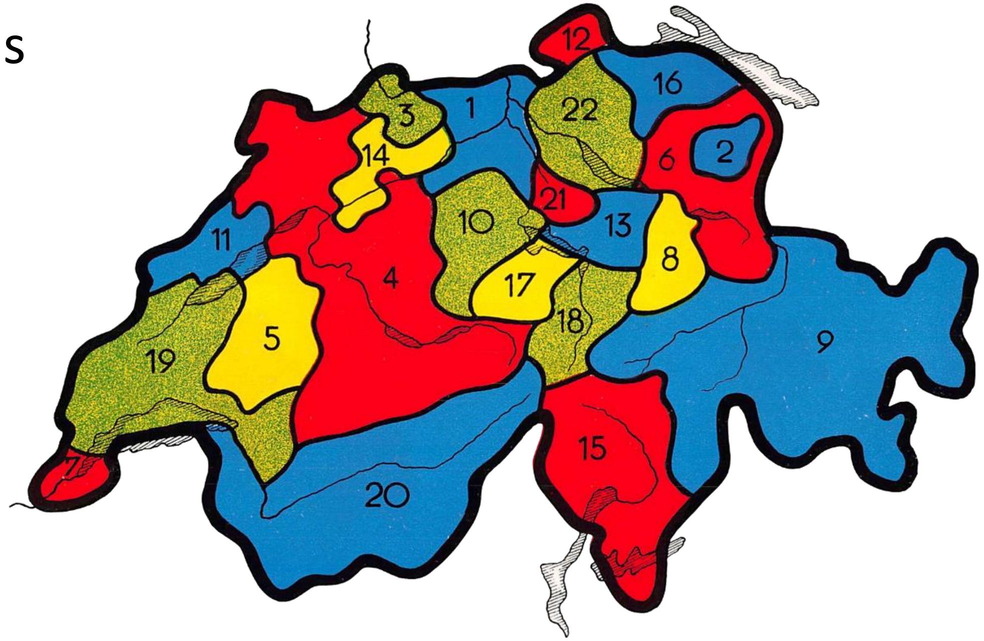
- For example, if a SUPPORTED algorithm has polylogarithmic runtime
  - ∃ LOCAL algorithm with constant factor overhead

Idea: simulate that support graph H is a complete graph

But: Restricted Graph Families are Useful ☺

- Real topologies are usually not complete graphs

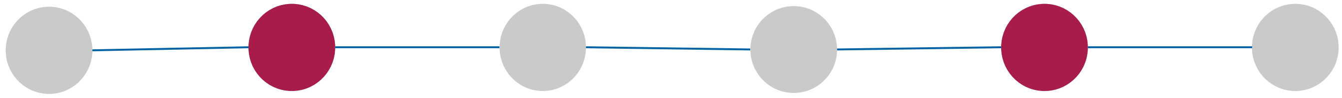
- Case study: planar graphs
  - Remain planar under edge deletions
  - Are 4-colorable



„Geloeste und ungeloeste Mathematische Probleme aus alter und neuer Zeit" by Heinrich Tietze
http://www.math.harvard.edu/~knill/graphgeometry/faqg.html

Case Study: Dominating Set

- Task: Find subset D of nodes s.t. every node
  - Has a neighbor in D or is in D



- Can we pre-compute?
  - A bad one yes: everyone in D!
  - But not an optimal one!
    - Graph can look very different

Case Study: Minimum Dominating Set in Planar Graphs

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let‘s analyze their LOCAL algorithm:

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

- Let‘s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let's analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]

universität wien

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let‘s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let‘s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [C
  - But maybe in the SUPPORTED m
- Let‘s analyze their LOCAL alg
  - Find weight-appropriate pseudo-fo st [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]

SUPPORTED speed-up:
1) precompute 4-coloring
2) reduce 4-colored pseudo-forest to 3 colors in 2 rounds

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let‘s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹] [constant time SUPPORTED model ☺]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
  - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let‘s analyze their LOCAL algorithm:
  - Find weight-appropriate pseudo-forest [constant time ☺]
  - 3-color pseudo-forest [non-constant time ☹] [constant time SUPPORTED model ☺]
  - Run clustering/optimization algorithms on components of constant size [constant time ☺]
- Also works for O(1)-genus graphs [extending work of Akhoondian Amiri et al.]

Case Study: Minimum Dominating Set in Planar Graphs

- (1+δ)-approximation not possible in constant time [Czygrinow et al., DISC 2008]
    - But maybe in the SUPPORTED model?

Max out-degree of 1

- Let‘s analyze their LOCAL algorithm:
    - Find weight-appropriate pseudo-forest [constant time ☺]
    - 3-color pseudo-forest [non-constant time ☹] [constant time SUPPORTED model ☺]
    - Run clustering/optimization algorithms on components of constant size [constant time ☺]
- Also works for O(1)-genus graphs [extending work of Akhoondian Amiri et al.]
    - Also for planar graphs for maximum independent set & maximum matching

Further Results in the *Active* SUPPORTED Model

Use all edges of H
for communication

Further Results in the *Active* SUPPORTED Model

Use all edges of H
for communication

Further Results in the *Active* SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]

Use all edges of H
for communication

Further Results in the *Active* SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t$*$poly log n)): e.g. MIS in SUPPORTED(poly log n)

Further Results in the *Active* SUPPORTED Model

Use all edges of H
for communication

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t$*$poly log n)): e.g. MIS in SUPPORTED(poly log n)

universität wien

Use all edges of
for communicati

Further Results in the *Active* SUPPORTE

- Connection to SLOCAL model [Ghaffari et al., STOC
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗pol

Polylogarithmic-Time Deterministic Network Decomposition and Distributed Derandomization

Václav Rozhoň
ETH Zurich
rozhonv@student.ethz.ch

Mohsen Ghaffari*
ETH Zurich
ghaffari@inf.ethz.ch

Abstract

We present a simple polylogarithmic-time deterministic distributed algorithm for network decomposition. This improves on a celebrated $2^{O(\sqrt{\log n})}$-time algorithm of Panconesi and Srinivasan [STOC'93] and settles one of the long-standing and central questions in distributed graph algorithms. It also leads to the first polylogarithmic-time deterministic distributed algorithms for numerous other graph problems, hence resolving several open problems, including Linial's well-known question about the deterministic complexity of maximal independent set [FOCS'87].

Put together with the results of Ghaffari, Kuhn, and Maus [STOC'17] and Ghaffari, Harris, and Kuhn [FOCS'18], we get a general distributed derandomization result that implies P-RLOCAL = P-LOCAL. That is, for any distributed problem whose solution can be checked in polylogarithmic-time, any polylogarithmic-time randomized algorithm can be derandomized to a polylogarithmic-time deterministic algorithm.

By known connections, our result leads also to substantially faster *randomized* algorithms for a number of fundamental problems including $(\Delta+1)$-coloring, MIS, and Lovász Local Lemma.

Through known connections, this general derandomization leads to better *deterministic* and *randomized* distributed algorithms for numerous problems. A sampling of end-results includes poly$(\log n)$-round deterministic algorithms for MIS, $\Delta+1$ coloring, the Lovász Local Lemma[3], hypergraph splitting, and defective coloring. These also lead to substantially improved randomized algorithms, including a poly$(\log \log n)$-time randomized $\Delta+1$ coloring [CLP18] and a poly$(\log \log n)$-time randomized algorithm for Lovász Local Lemma in constant degree graphs [GHK18].

Use all edges of H
for communication

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

Further Results in the *Active* SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t$*$poly log n)): e.g. MIS in SUPPORTED(poly log n)

Use all edges of H
for communication

Also works in *passive* model:
SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

Further Results in the *Active* SUPPORTED Model

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t$*$poly log n)): e.g. MIS in SUPPORTED(poly log n)

Use all edges of H
for communication

Also works in *passive* model:
SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

Use all edges of H
for communication

Also works in *passive* model:
SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t$*$poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

universität wien

Use all edges of H
for communication

Also works in *passive* model:
SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

- Locally Checkable Labelings LCL:

universität wien

Use all edges of H for communication

Also works in *passive* model: SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm: $2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

Use all edges of H
for communication

Also works in *passive* model:
SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm:
$2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

Also works without
the *active* model

universität wien

Use all edges of H for communication

Also works in *passive* model: SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm: $2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model

Also works without the *active* model

- Optimization problem: Maximum Independent Set, of size α(G)

universität wien

Use all edges of H for communication

Also works in *passive* model: SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Further Results in the *Active* SUPPORTED Model

Best LOCAL algorithm: $2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

Also works without the *active* model

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model
- Optimization problem: Maximum Independent Set, of size α(G)
  - Set of size (α(G)-ε)n in O($\log_{1+\varepsilon}$ n), respectively (1+ε) approximation if maximum degree Δ constant

universität wien

Further Results in the *Active* SUPPORTED Model

Use all edges of H for communication

Also works in *passive* model: SLOCAL(t) →SUPPORTED($\Delta^{O(t)}$)

Best LOCAL algorithm: $2^{O(\sqrt{\log n})}$

- Connection to SLOCAL model [Ghaffari et al., STOC 2017]
  - SLOCAL(t) can be simulated in SUPPORTED(O(t∗poly log n)): e.g. MIS in SUPPORTED(poly log n)
  - Converse not true, respectively open question

e.g. network size, restricted H, known inputs..

Also works without the *active* model

- Locally Checkable Labelings LCL:
  - LCL in LOCAL(o(log n)) can be solved in O(1) in the SUPPORTED model
- Optimization problem: Maximum Independent Set, of size α(G)
  - Set of size (α(G)-ε)n in O($\log_{1+\varepsilon}$ n), respectively (1+ε) approximation if maximum degree Δ constant
  - Cannot be approximated by o(Δ/log Δ) in time o($\log_\Delta$ n) in the active SUPPORTED model

Bigger Open Question/Opportunity

How to efficiently leverage such preprocessing/distributed computing to efficiently scale controllers (and network updates)?

So far largely unexplored

So let's get back things we know about ☺
Congestion and network functions?

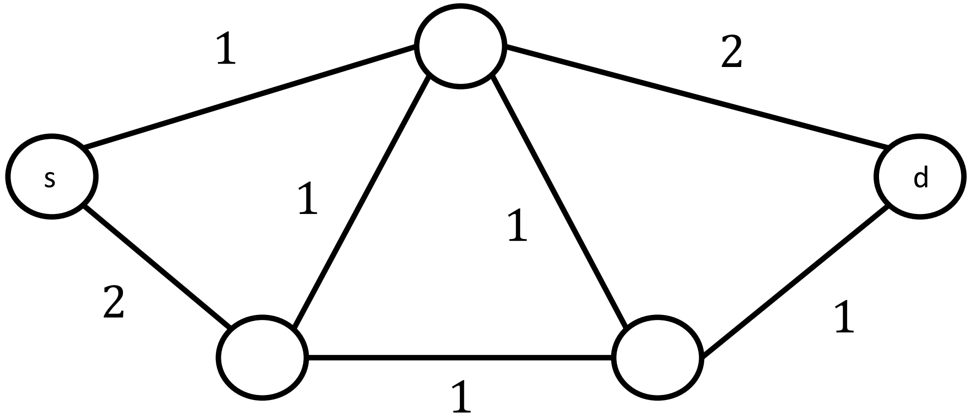

CONGESTION
AHEAD
NEXT 20
YEARS

Congestion?

- “Stronger” consistency constraint: also do not violate link capacities

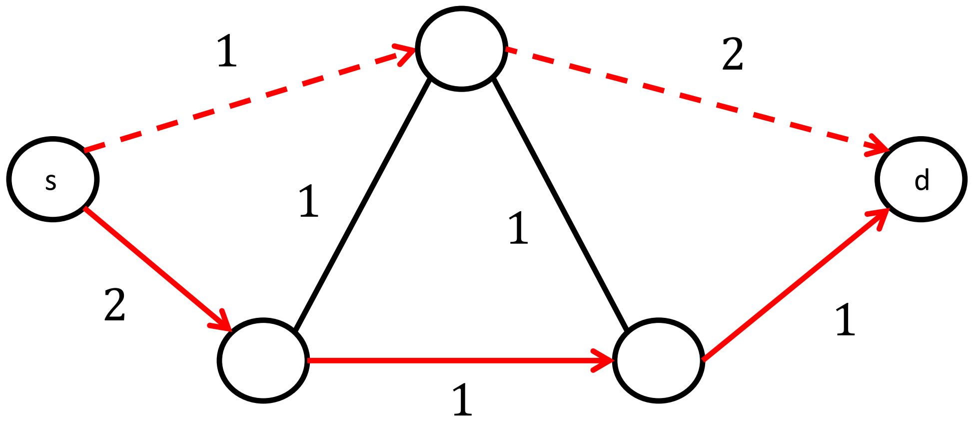
Congestion?

- “Stronger” consistency constraint: also do not violate link capacities



Congestion?

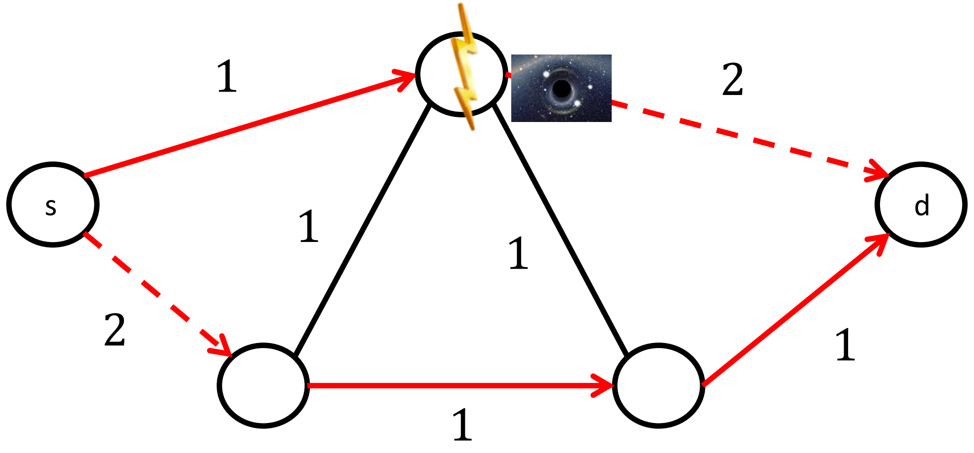
- "Stronger" consistency constraint: also do not violate link capacities
  - Flow size: 1



Round **0**

Congestion?

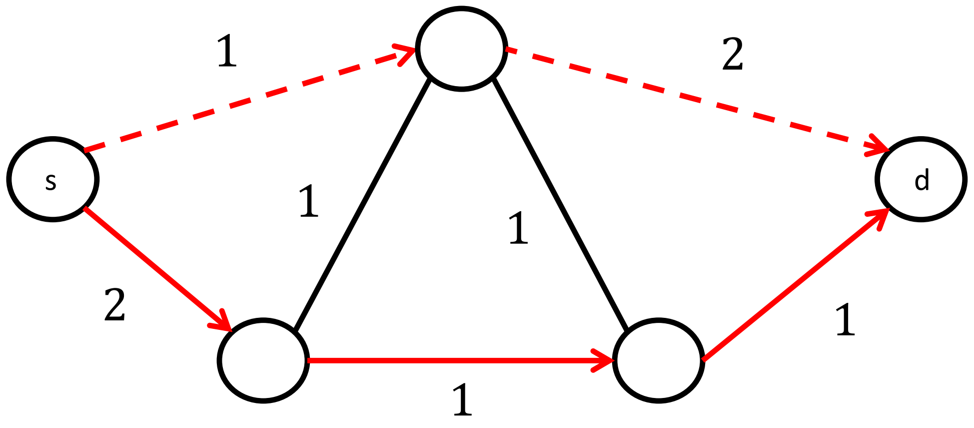
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1



Round **1**

Congestion?

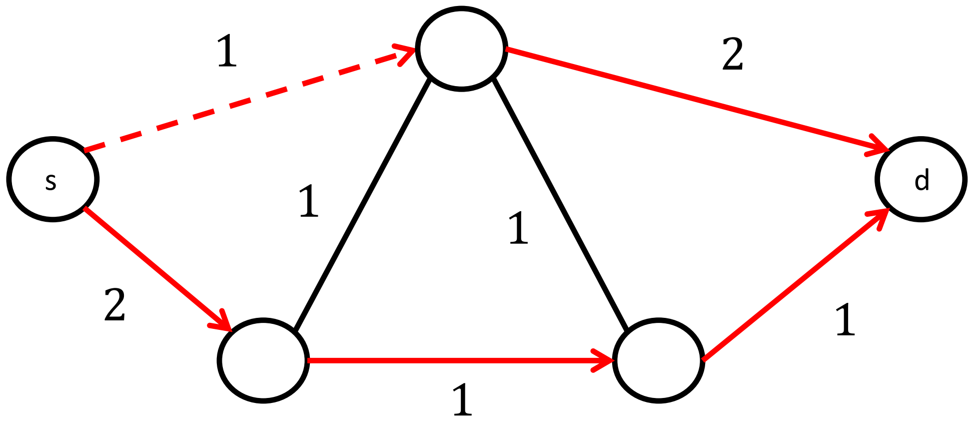
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1



Round **0**

Congestion?

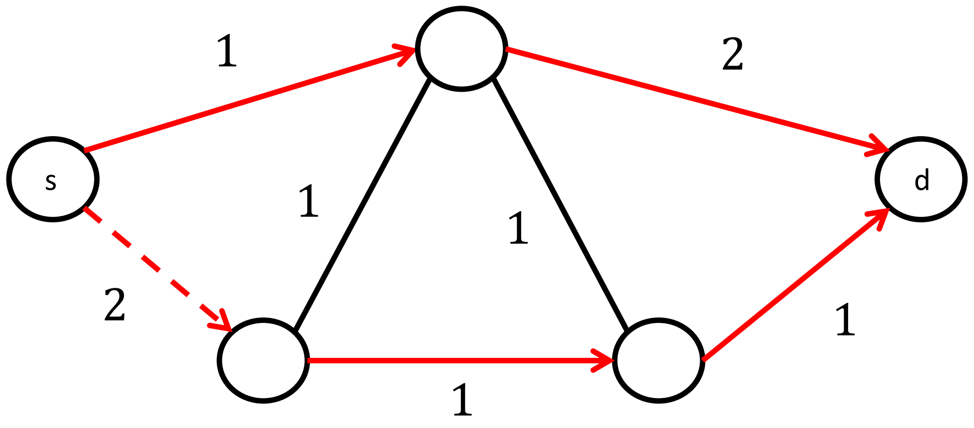
- "Stronger" consistency constraint: also do not violate link capacities
  - Flow size: 1



Round **1**

Congestion?

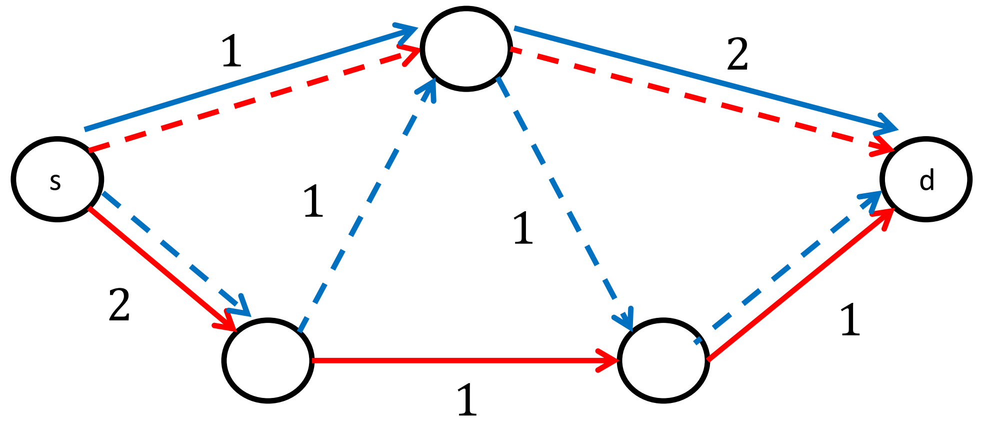
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1



Round **2**

Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1

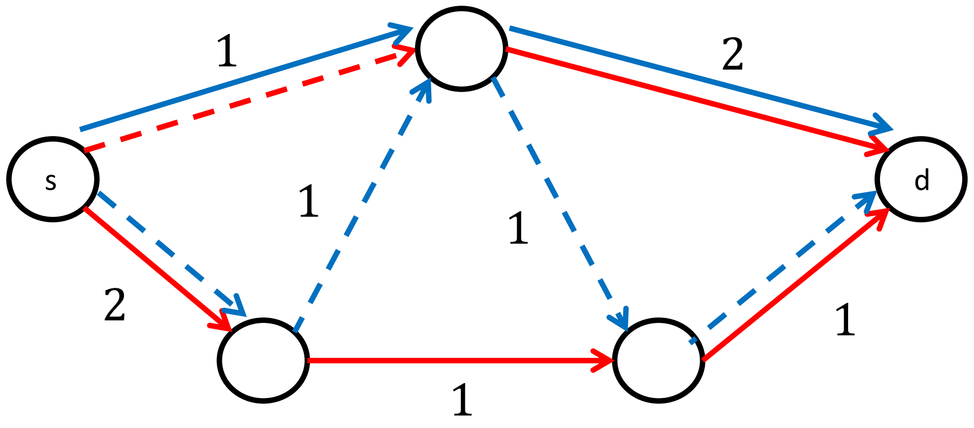


Round **0**

universität wien

Congestion?

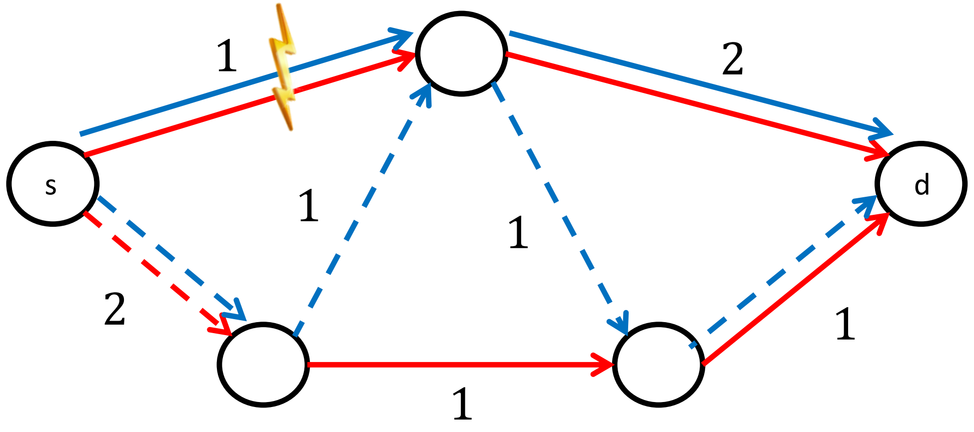
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **1**

Congestion?

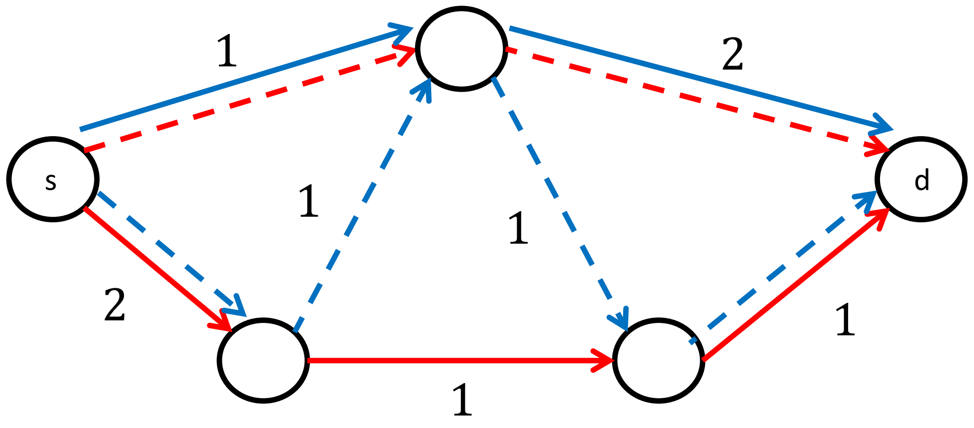
- "Stronger" consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **2**

Congestion?

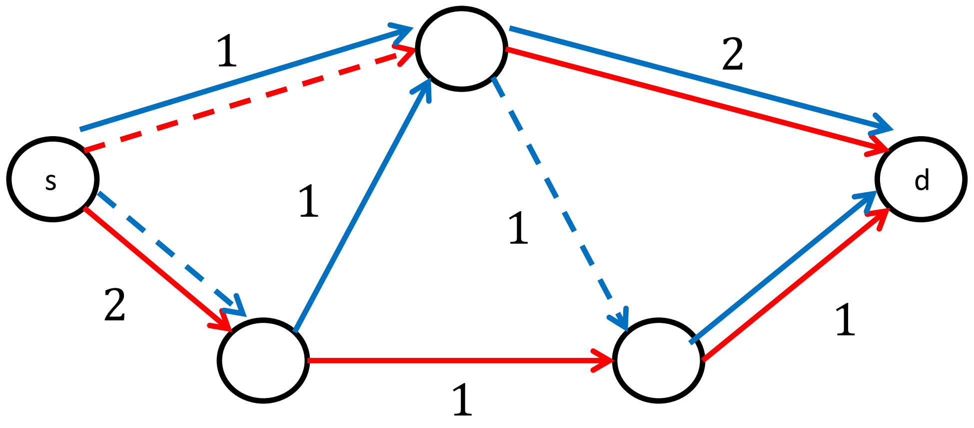
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **0**

Congestion?

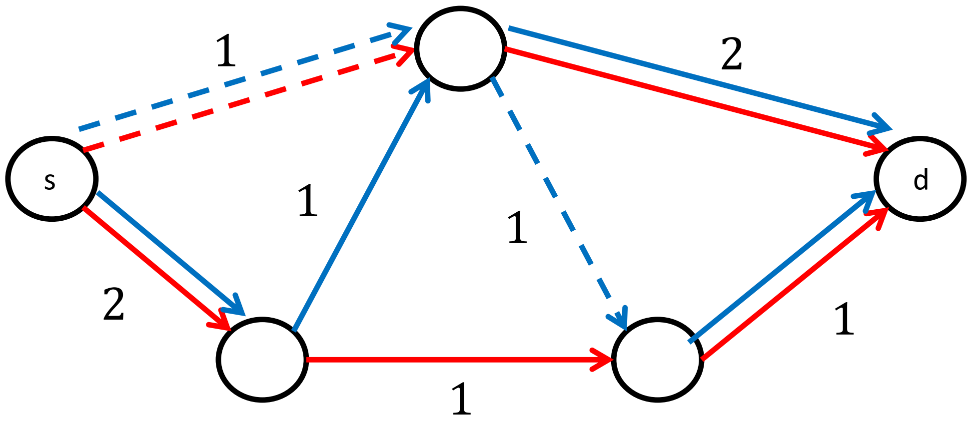
- "Stronger" consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **1**

Congestion?

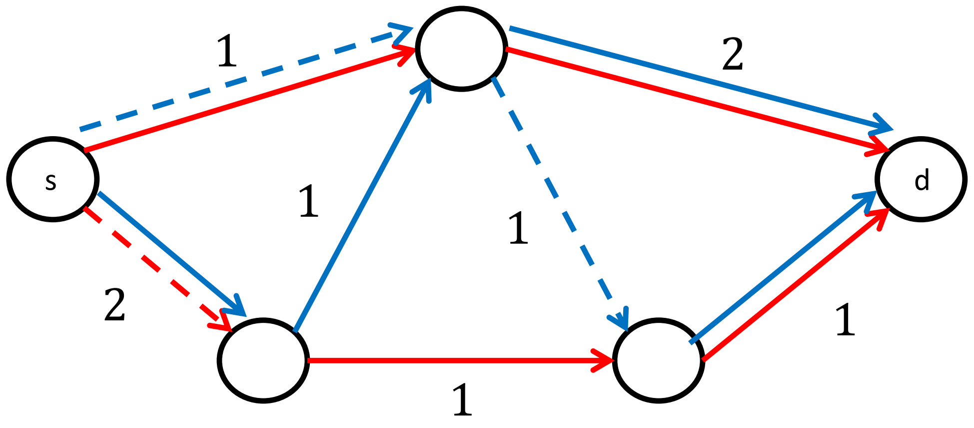
- "Stronger" consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **2**

Congestion?

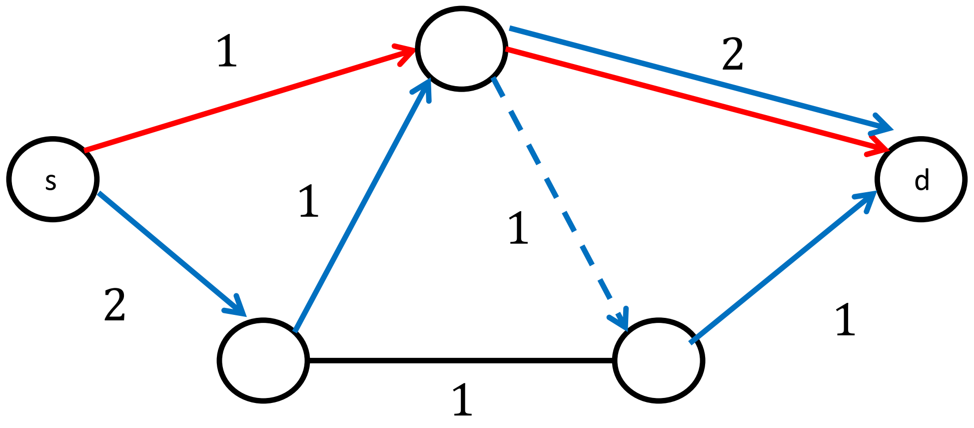
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **3**

Congestion?

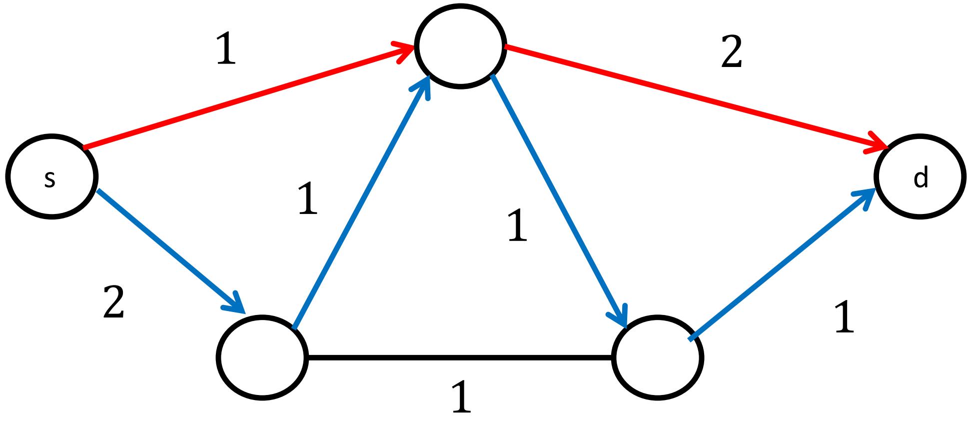
- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **3**

Congestion?

- “Stronger” consistency constraint: also do not violate link capacities
  - Flow size: 1, 1



Round **4**

Complexity of Avoiding Congestion?

- NP-hard already for 2 unit size flows on general graphs

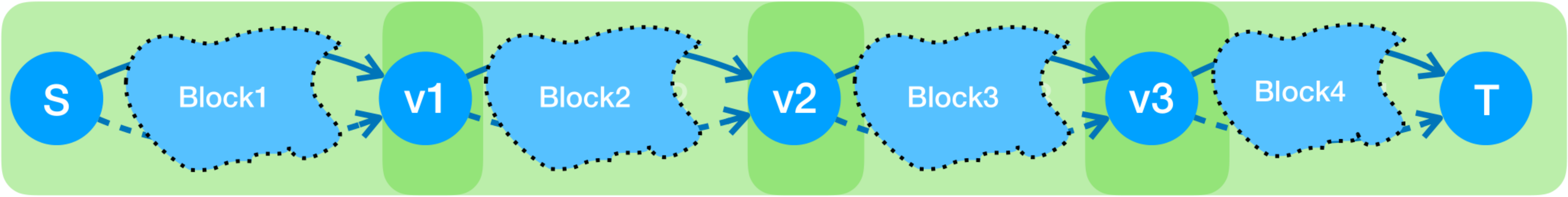
- Also NP-hard on acyclic graphs for k flows
  - But can be FPT characterized for k flows on acyclic graphs: $O\left(2^{O(k \log k)}|G|\right)$
    - In other words, linear runtime for constant k on DAGs

- For just 2 unit size flows (where old/new ***individually*** is a DAG): Optimal schedule in P (NPH for 6)

Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP'18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking'19

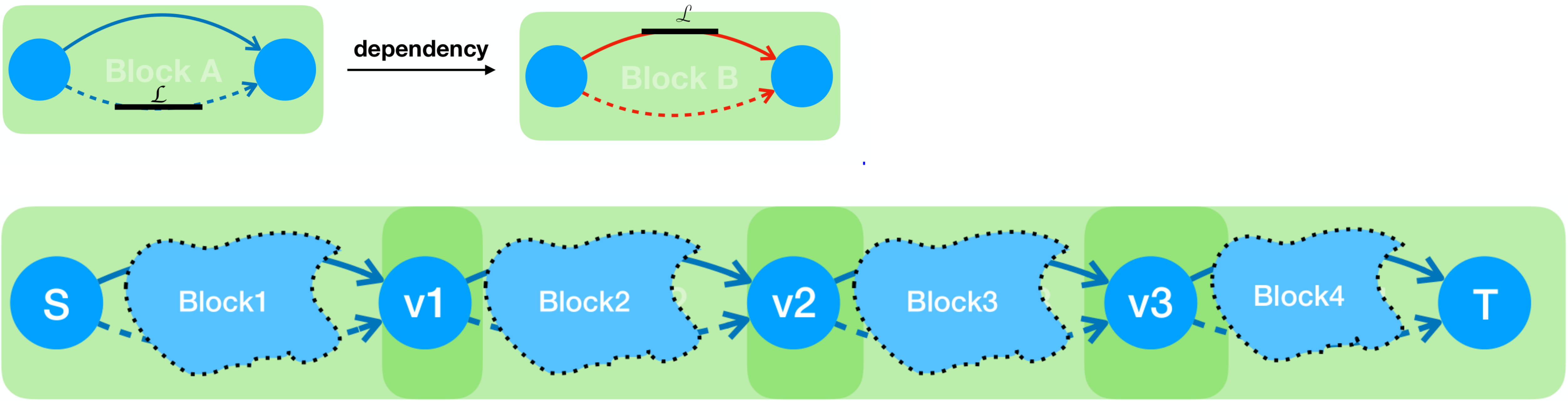
Complexity of Avoiding Congestion?

- NP-hard already for 2 unit size flows on general graphs



- For just 2 unit size flows (where old/new ***individually*** is a DAG): Optimal schedule in P (NPH for 6)

Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP'18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking'19



- For just 2 unit size flows (where old/new ***individually*** is a DAG): Optimal schedule in P (NPH for 6)

Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP'18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking'19

Complexity of Avoiding Congestion?

?

- NP-hard already for 2 unit size flows on general graphs

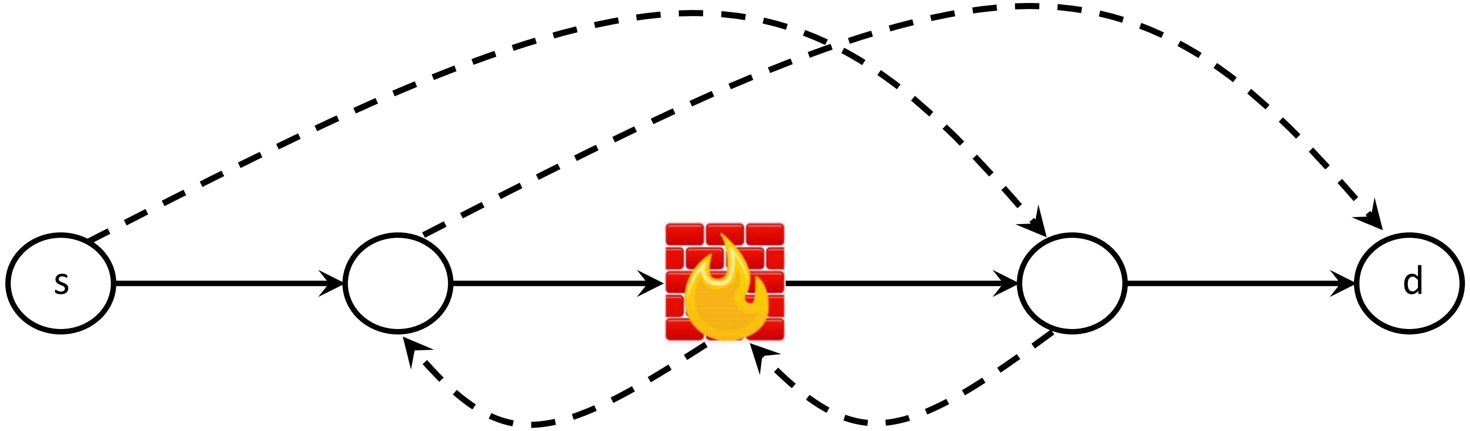
- Also NP-hard on acyclic graphs for 6 flows
  - But can be FPT characterized for k flows on acyclic graphs: $O\left(2^{O(k \log k)}|G|\right)$
    - In other words, linear runtime for constant k on DAGs

- For just 2 unit size flows (where old/new individually is a DAG): Optimal schedule in P

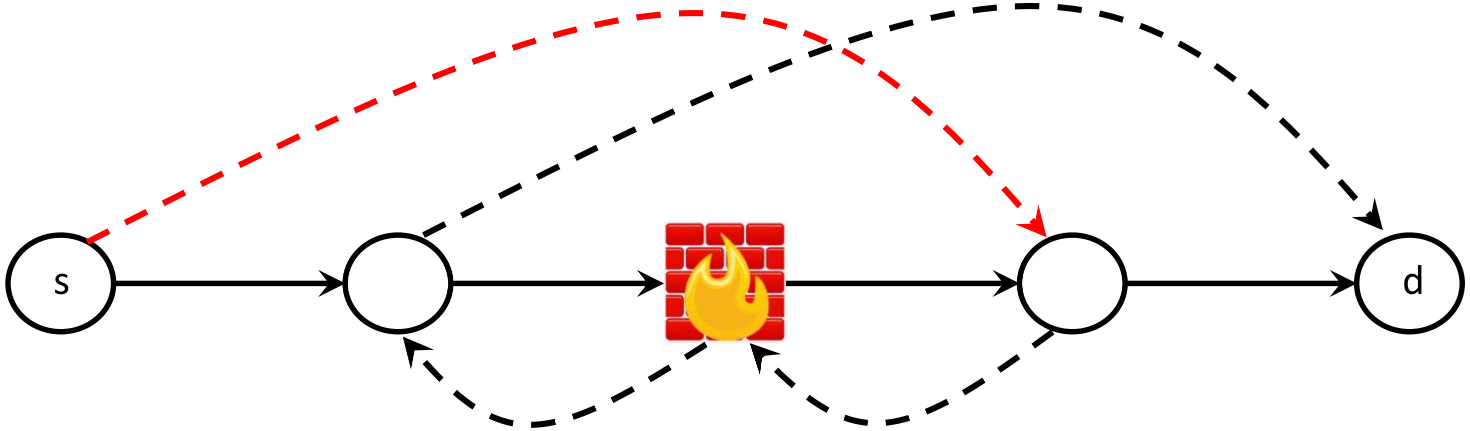
Congestion-Free Rerouting of Flows on DAGs. S. Akhoondian Amiri, S. Dudycz, S. Schmid, S. Widerrecht, ICALP'18
On Polynomial-Time Congestion-Free Software-Defined Network Updates. AA, D., M. Parham, S., S. W., Networking'19

Take a Step Back: No Loops and a Firewall

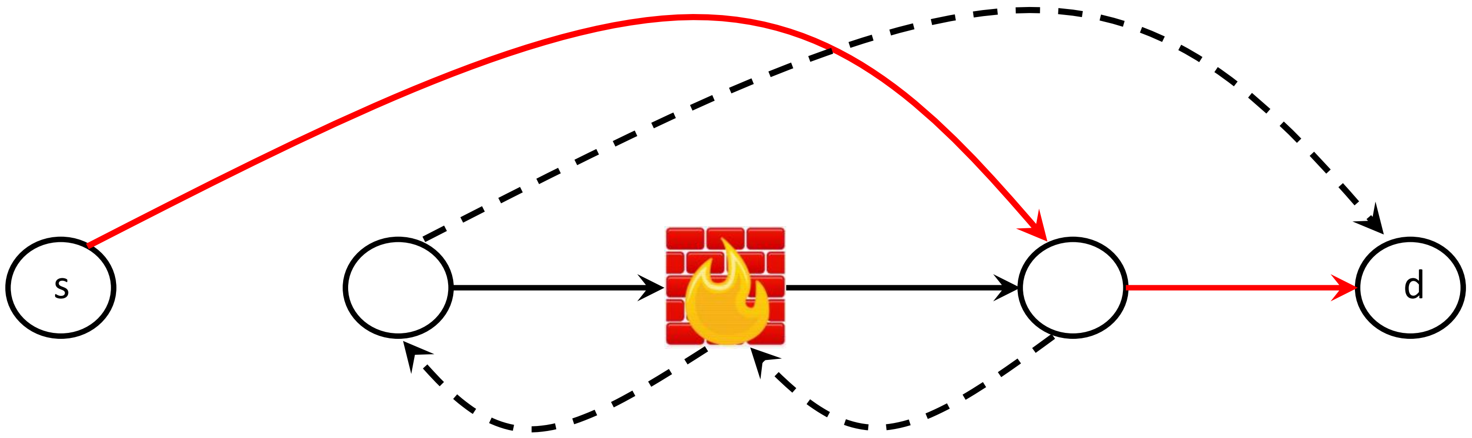
Which forwarding rule to update first?



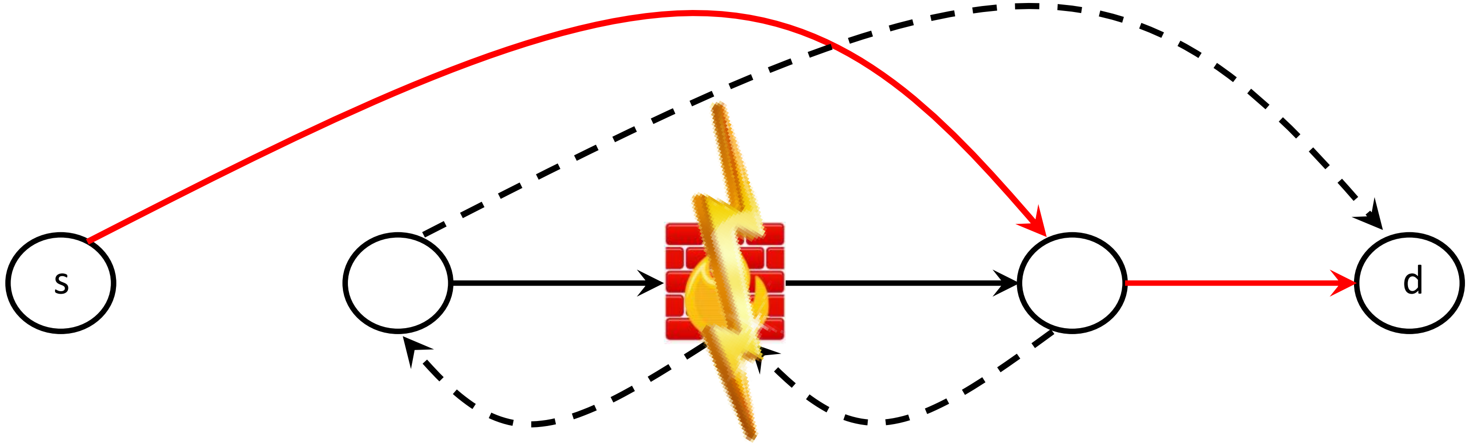
Take a Step Back: No Loops and a Firewall



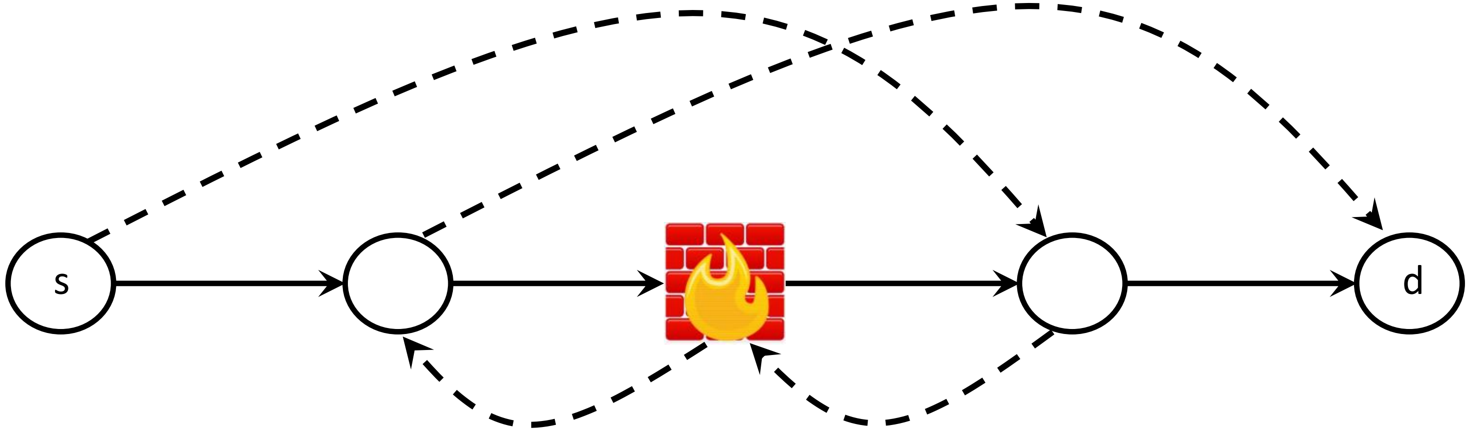
Take a Step Back: No Loops and a Firewall



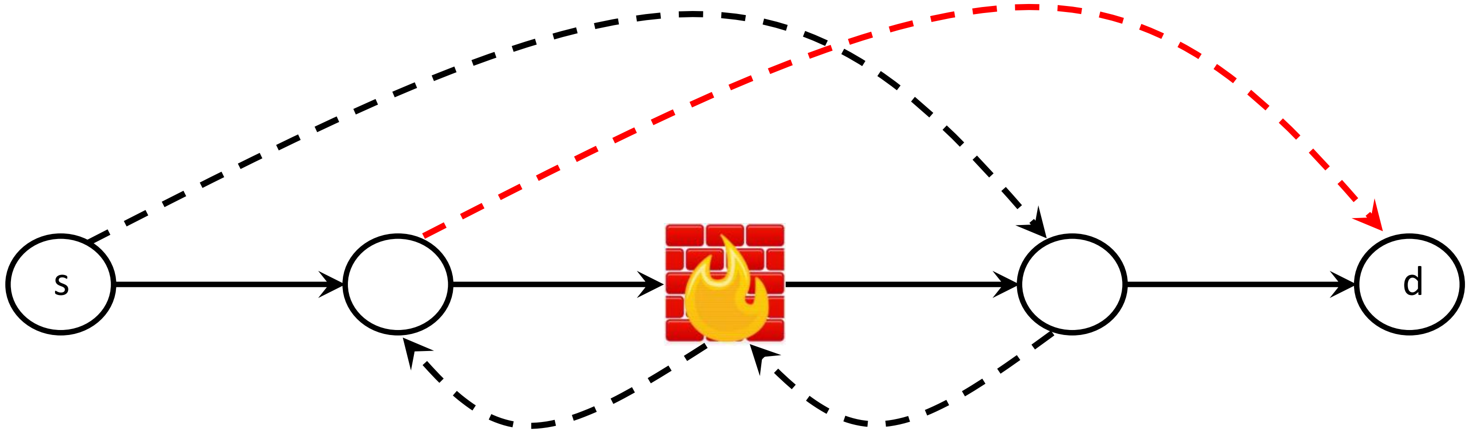
Take a Step Back: No Loops and a Firewall



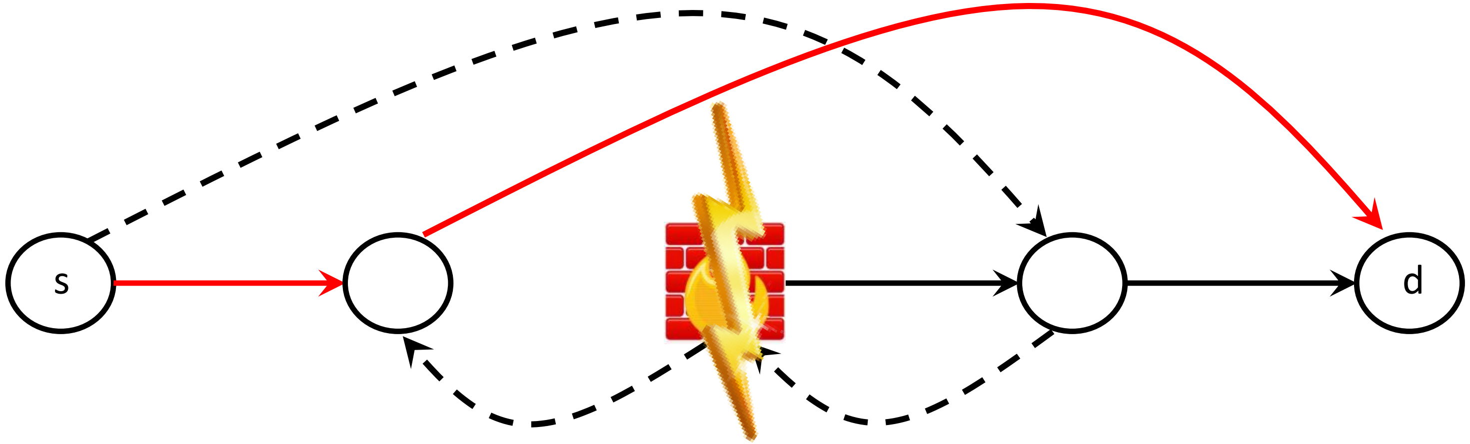
Take a Step Back: No Loops and a Firewall



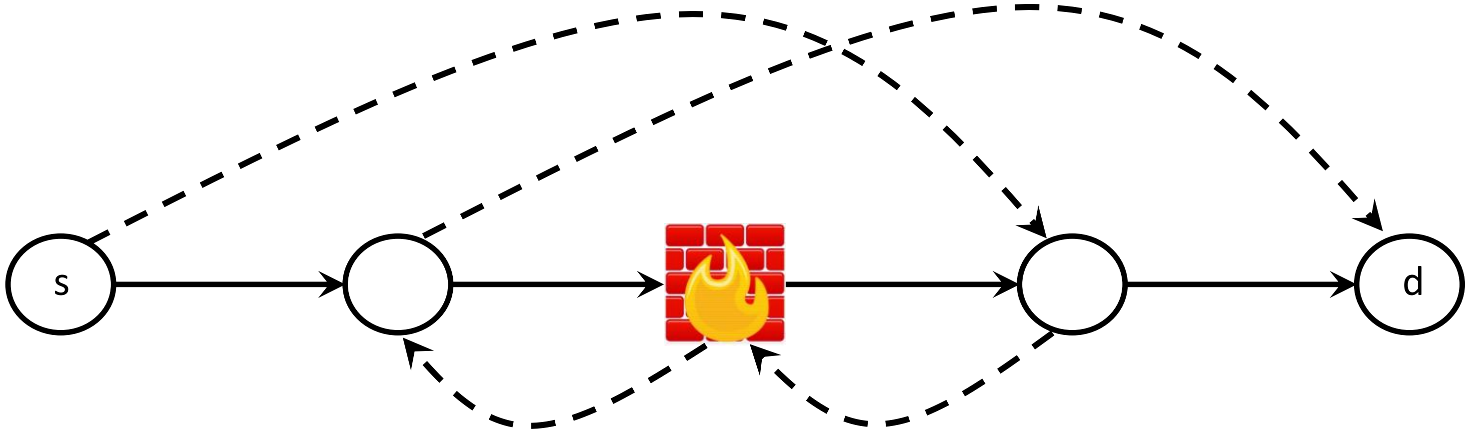
Take a Step Back: No Loops and a Firewall



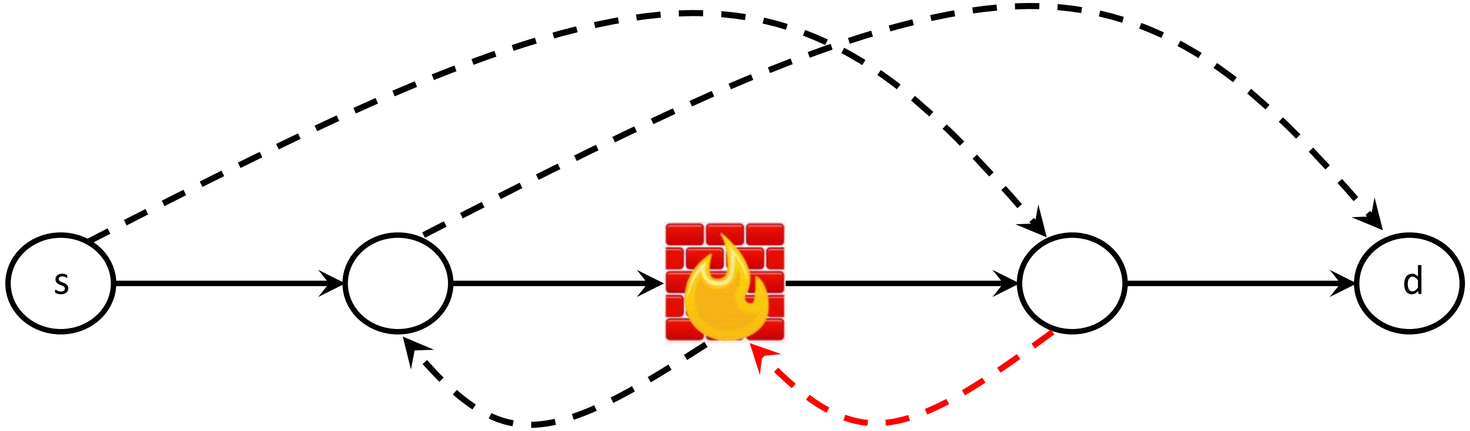
Take a Step Back: No Loops and a Firewall



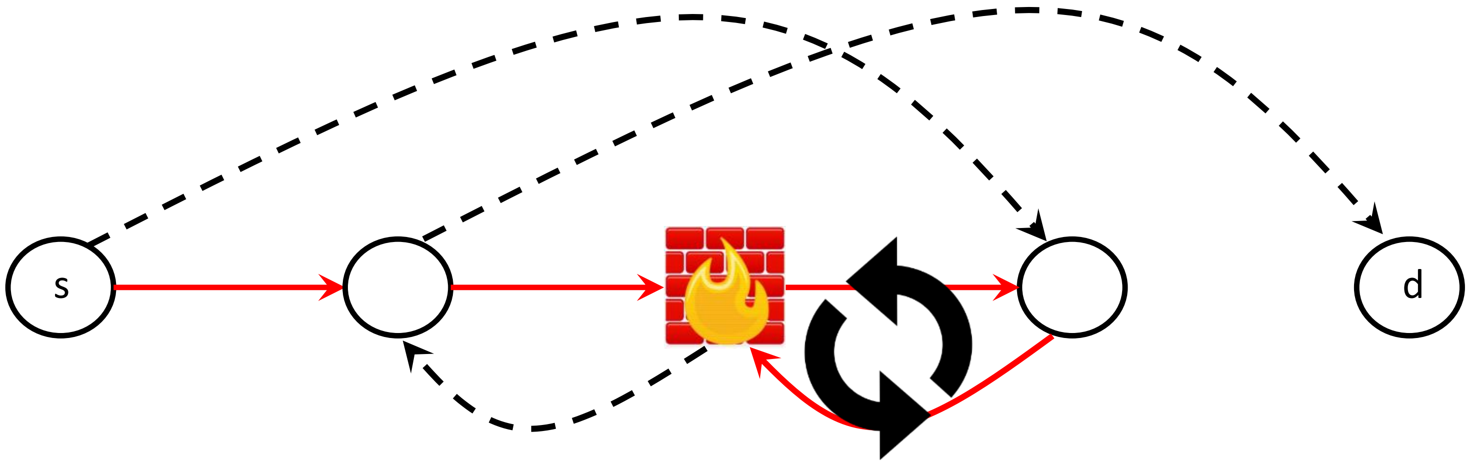
Take a Step Back: No Loops and a Firewall



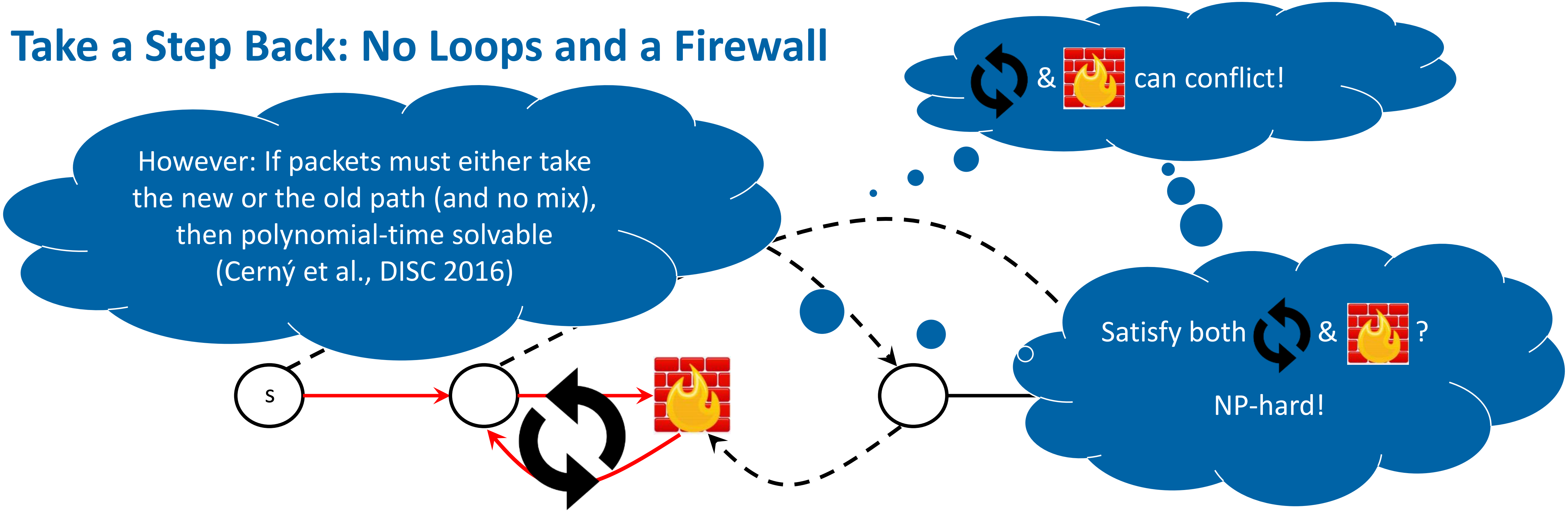
Take a Step Back: No Loops and a Firewall



Take a Step Back: No Loops and a Firewall



Take a Step Back: No Loops and a Firewall



Transiently Secure Network Updates. A. Ludwig, S. Dudycz, M. Rost, S. Schmid. SIGMETRICS 2016.

Different model: “tagged” Flows

- Identified by a “tag“ in the packet header, update via
  - Install new tag‘ rules
  - Switch from tag to tag‘ at source

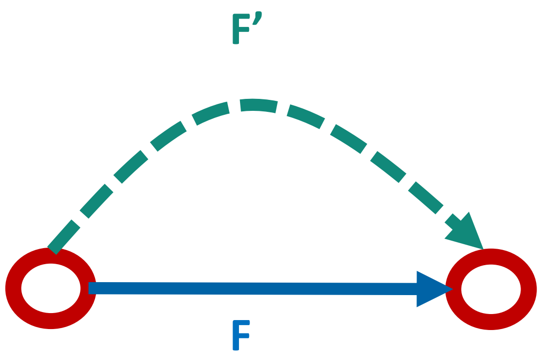
If we move a flow, will there be congestion?

If we move a flow, will there be congestion?

- How do we move a flow **F**? Usually: 2-phase commit: [*Reitblatt et al., SIGCOMM'12*]

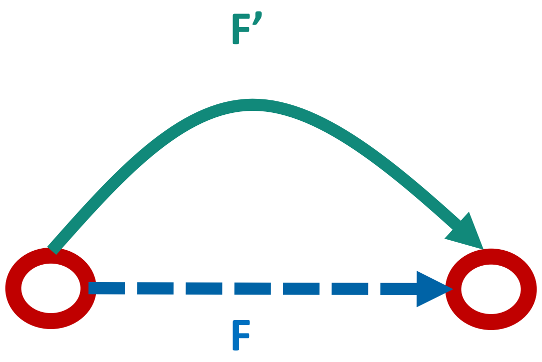
If we move a flow, will there be congestion?

- How do we move a flow F? Usually: 2-phase commit:
  - Deploy new flow rules F'



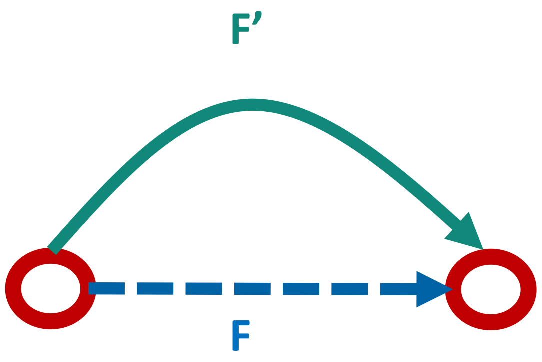
If we move a flow, will there be congestion?

- How do we move a flow **F**? Usually: 2-phase commit:
  - Deploy new flow rules **F’**
  - Change packet tag at source from **F** to **F’**



If we move a flow, will there be congestion?

- How do we move a flow F? Usually: 2-phase commit:
  - Deploy new flow rules F’
  - Change packet tag at source from F to F’



Can also be implemented by proof-labeling techniques

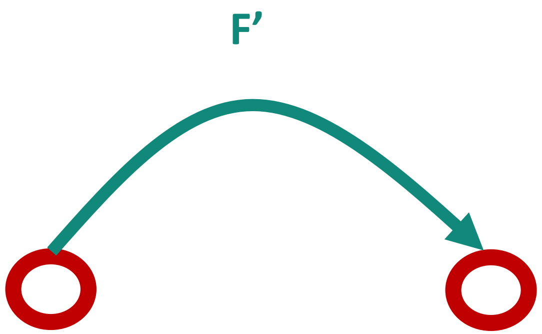
“hand holding”?

Respects network functions!

Go backwards with distance information

If we move a flow, will there be congestion?

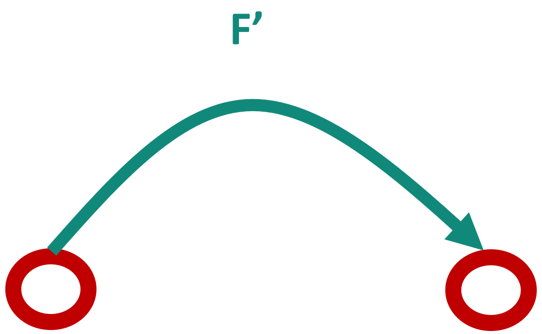
- How do we move a flow **F**? Usually: 2-phase commit:
  - Deploy new flow rules **F'**
  - Change packet tag at source from **F** to **F'**
  - Clean-up of old rules



universität wien

If we move a flow, will there be congestion?

- How do we move a flow **F**? Usually: 2-phase commit:
  - Deploy new flow rules **F’**
  - Change packet tag at source from **F** to **F’**
  - Clean-up of old rules

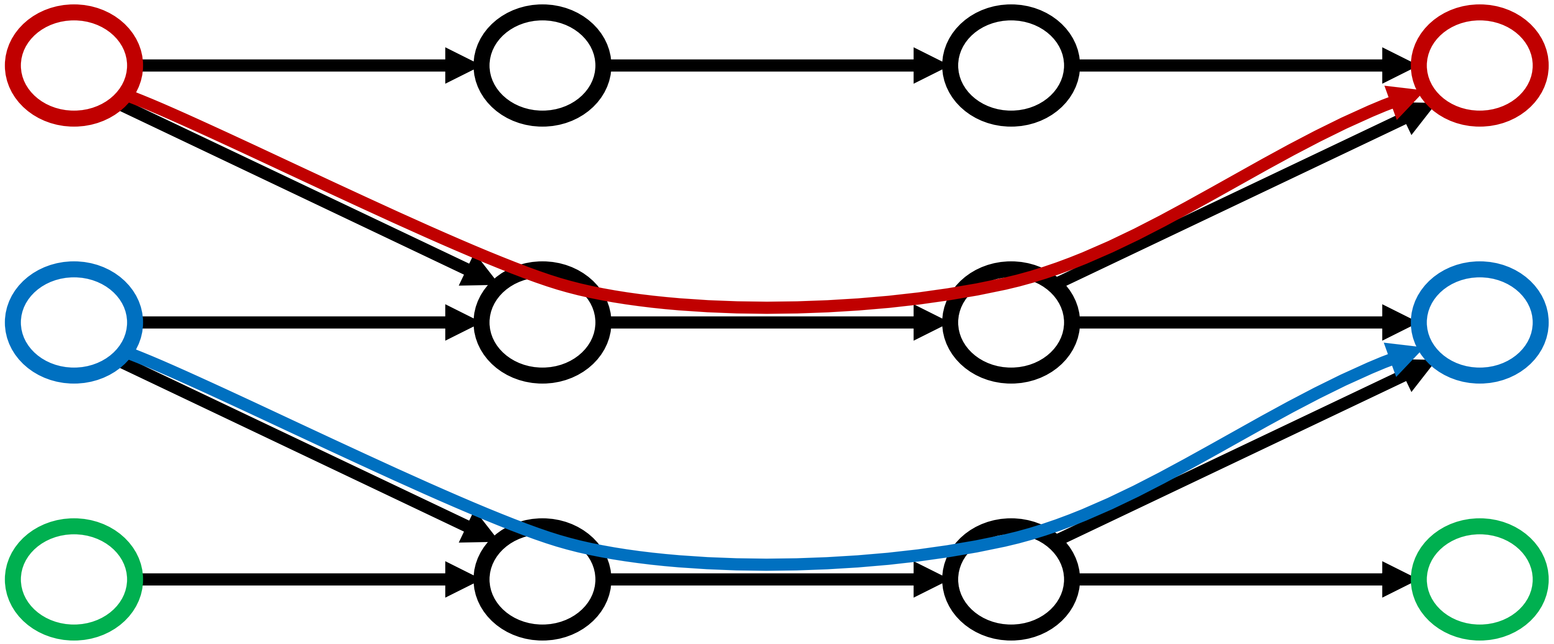


- First check:
  - Is the new network state without congestion?
  - Easy ☺ (flow size versus capacity)

Also verifiable by proof-labeling techniques

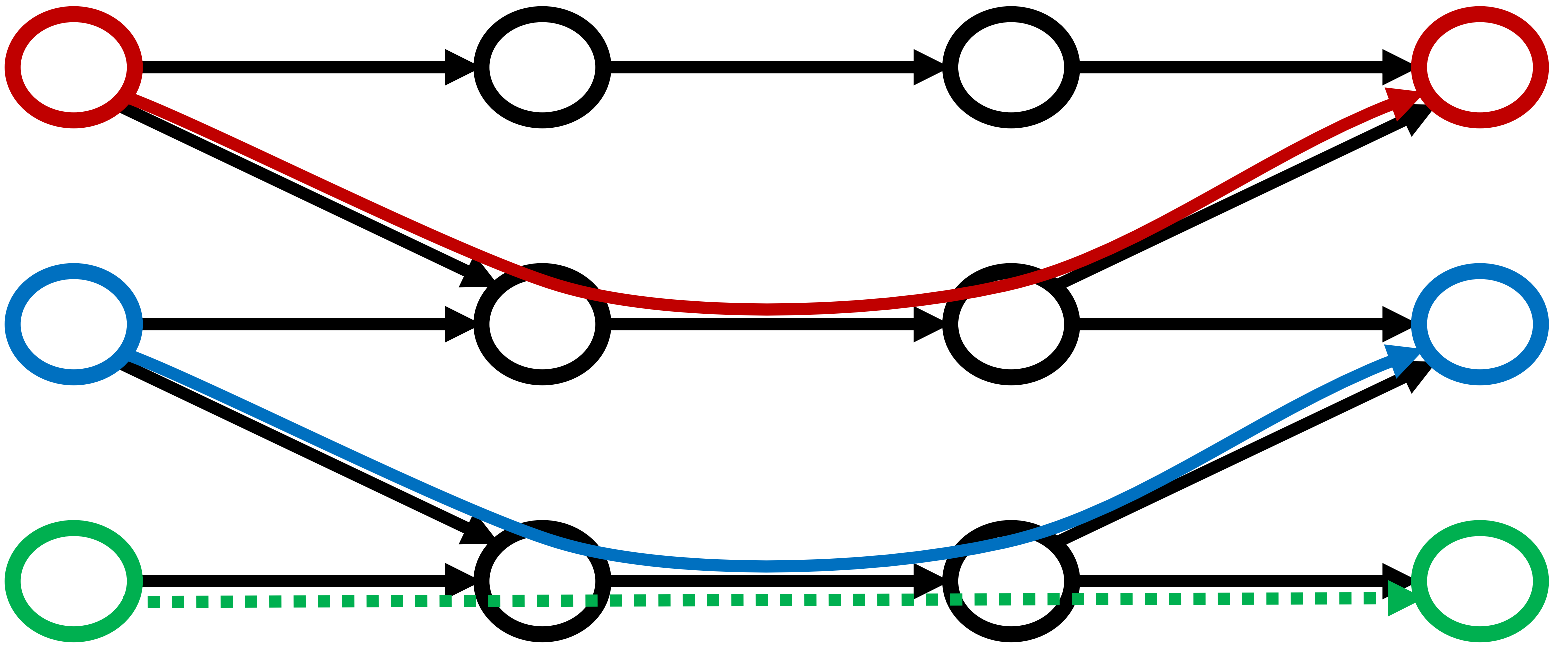
- Is that it?

A Small Sample Network



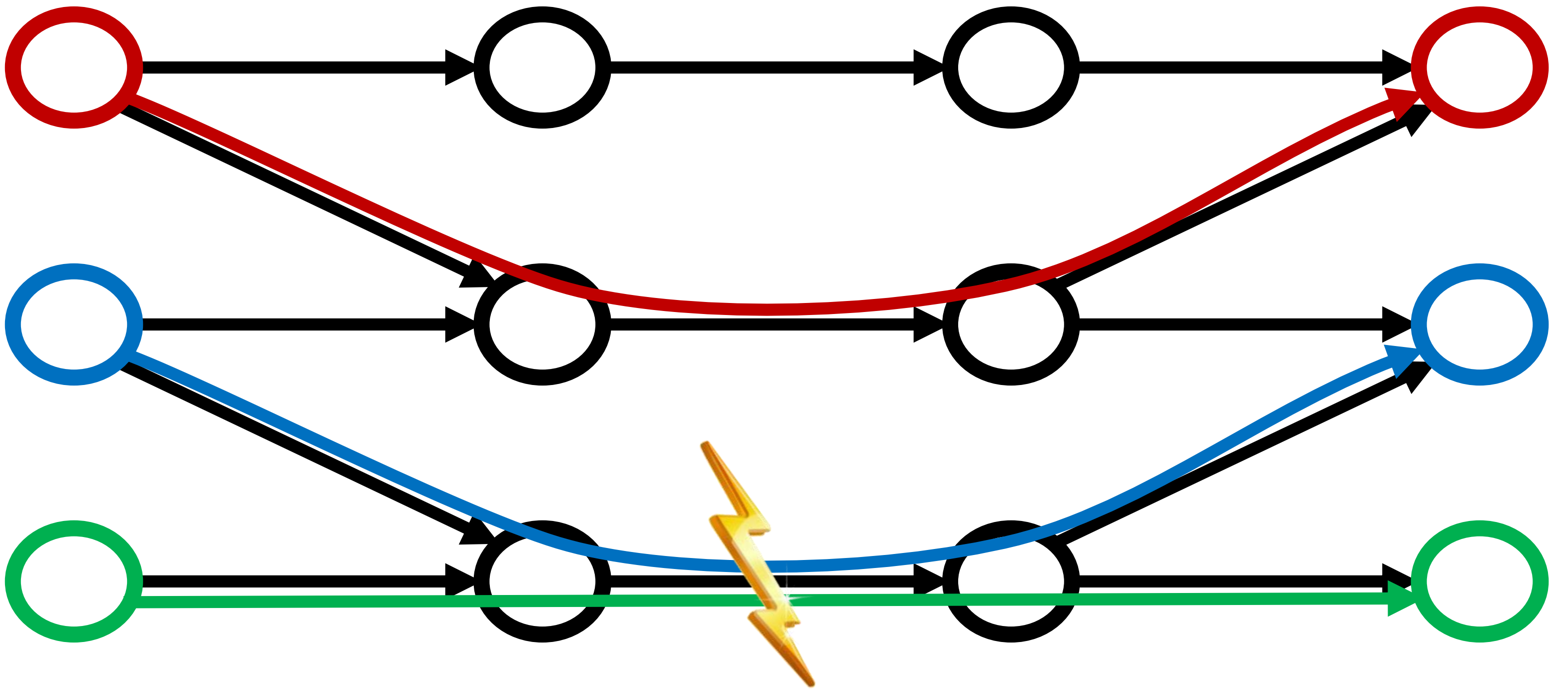
Unit size flows and capacities

Green wants to send as well



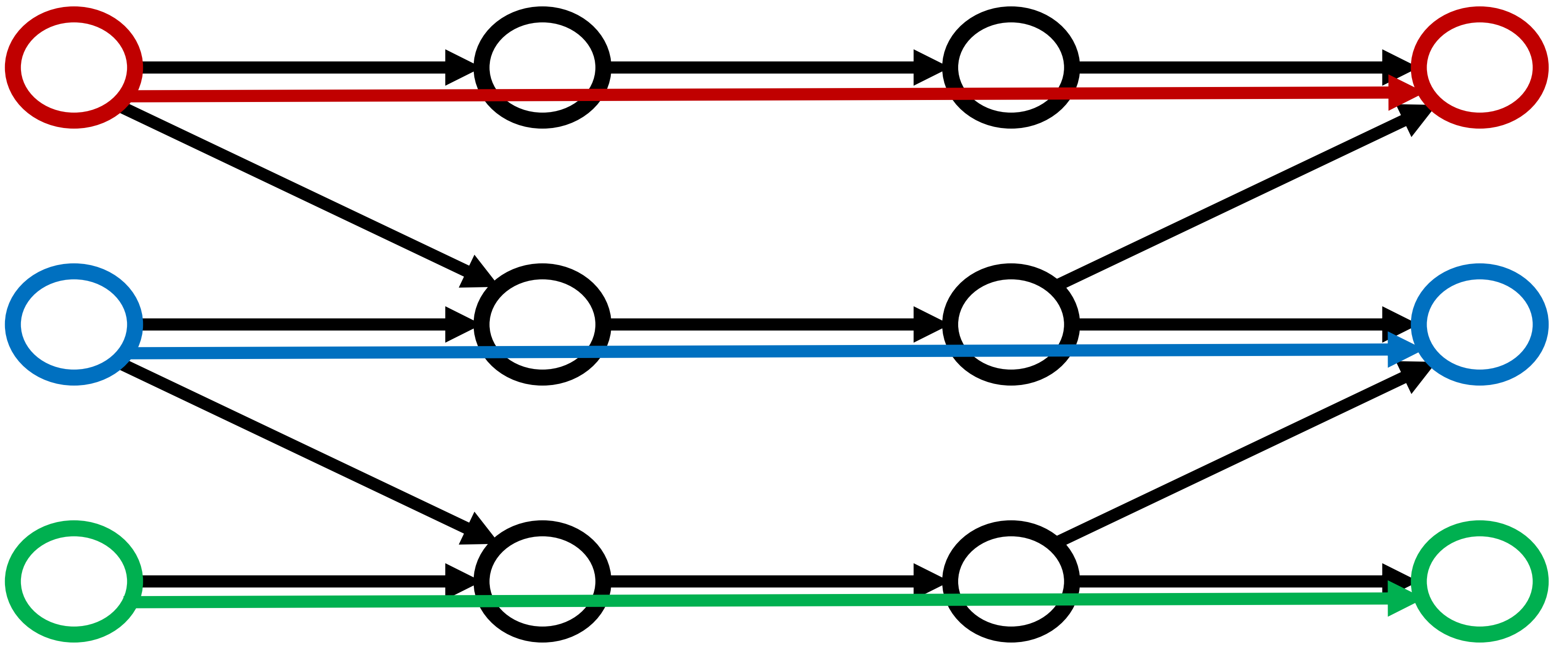
Unit size flows and capacities

Congestion!



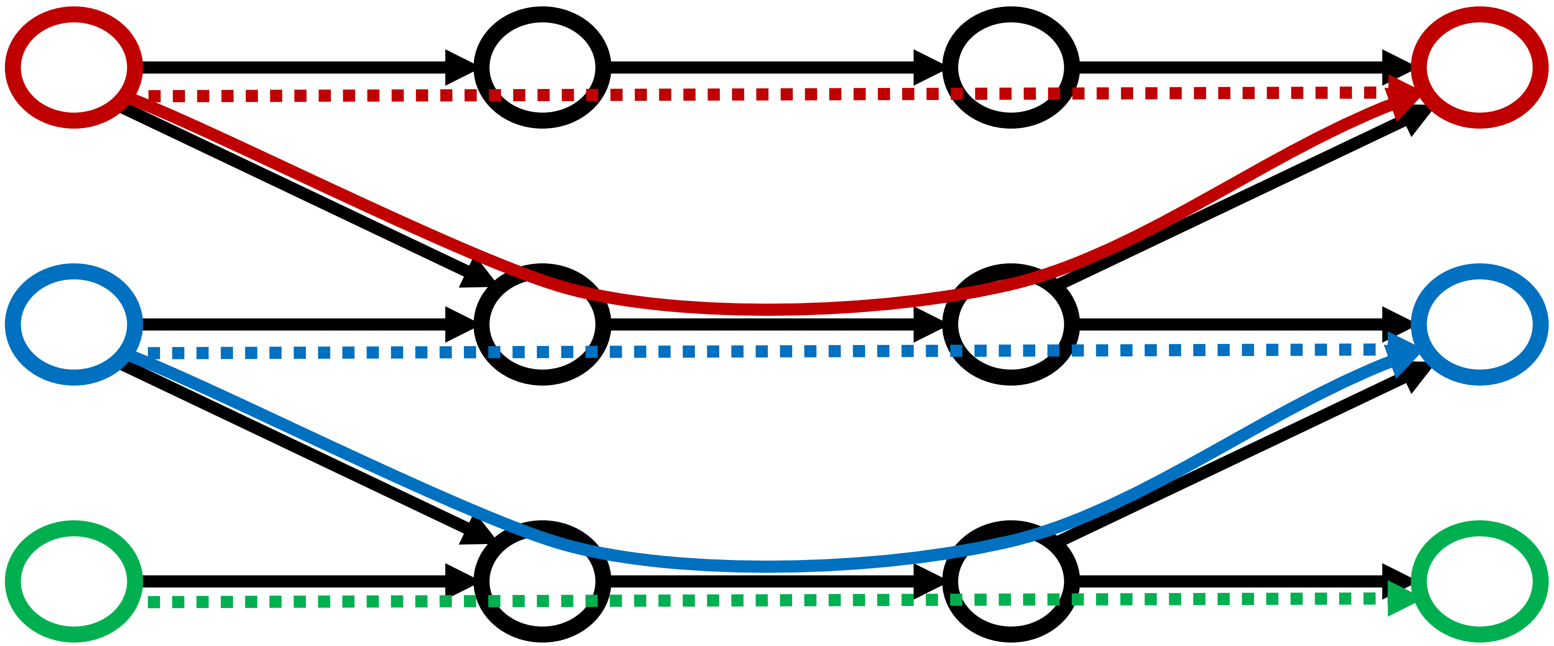
Unit size flows and capacities

This would work



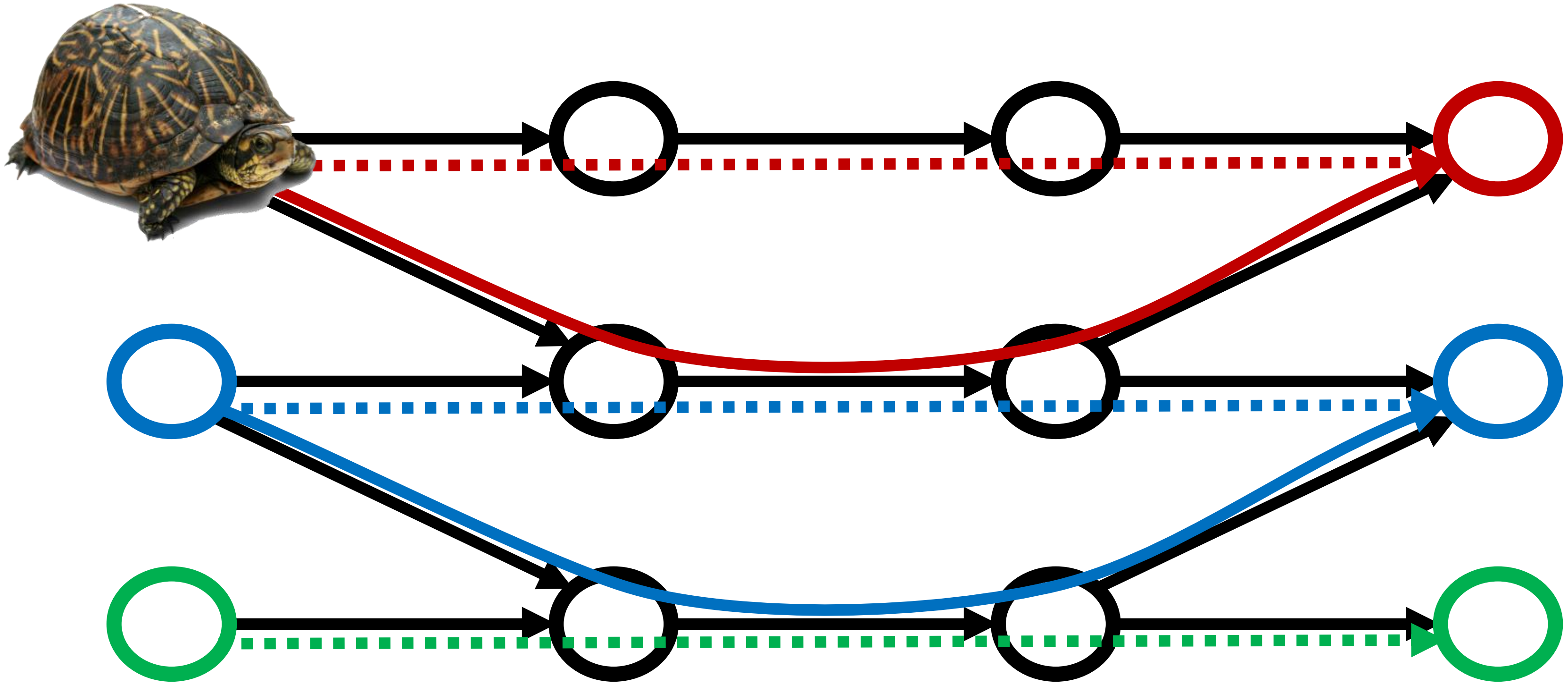
Unit size flows and capacities

So lets go back



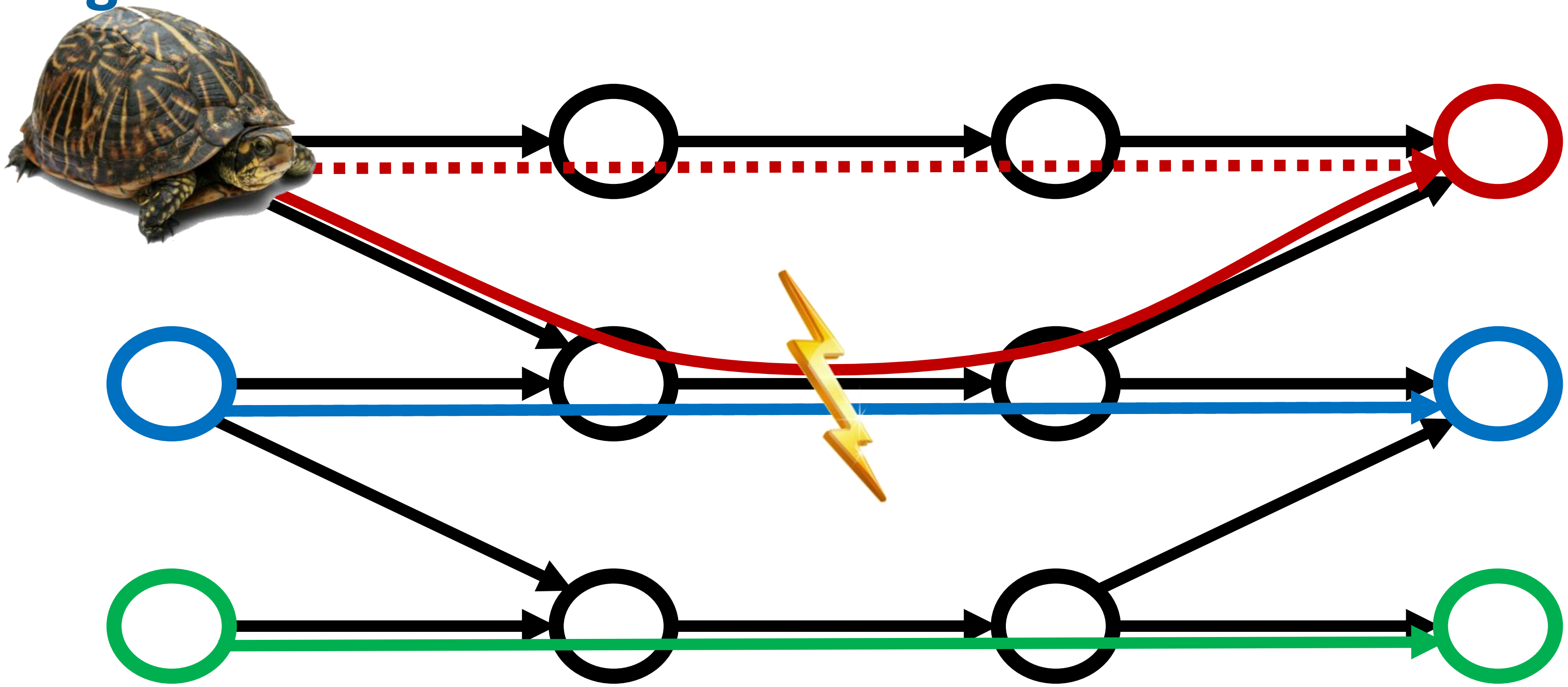
Unit size flows and capacities

But Red is a bit Slow..



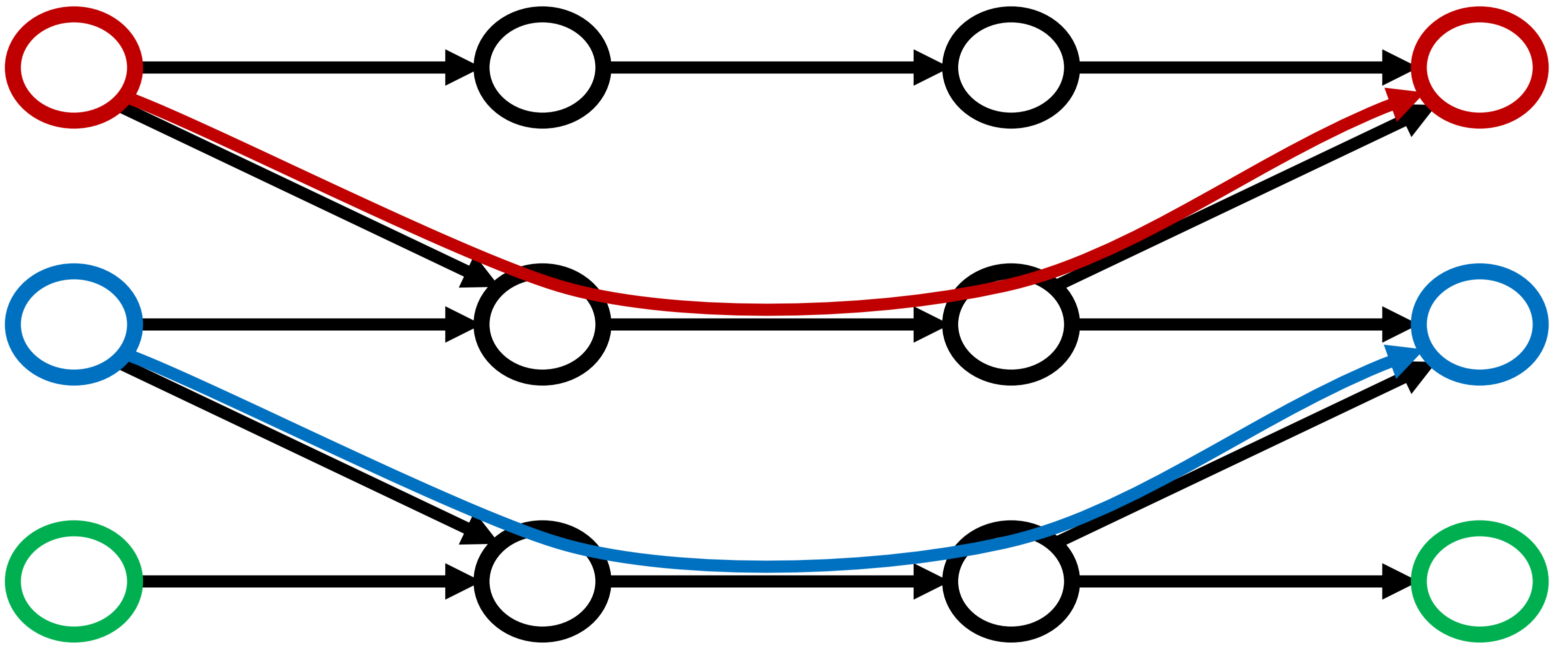
Unit size flows and capacities

Congestion Again!



Unit size flows and capacities

So lets go Back …

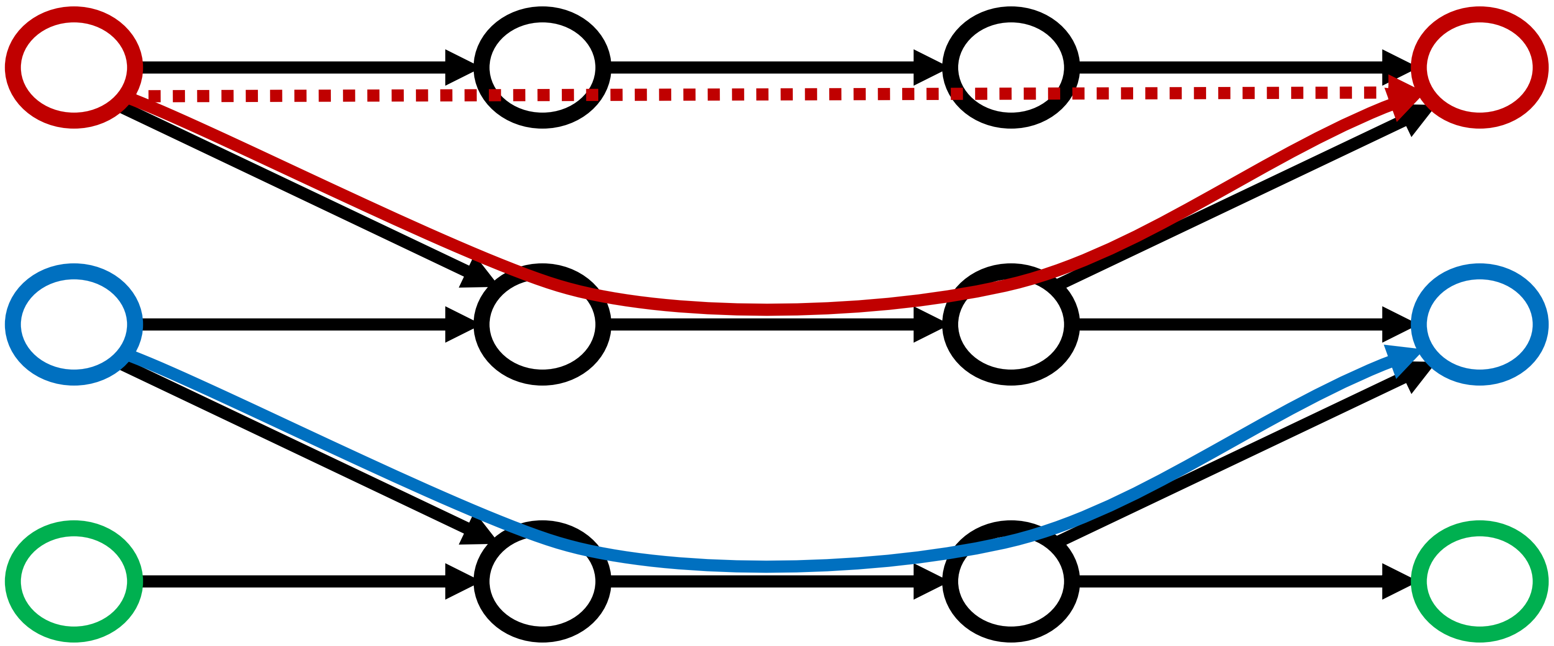


Round **0** *(old)*

Unit size flows and capacities

universität wien

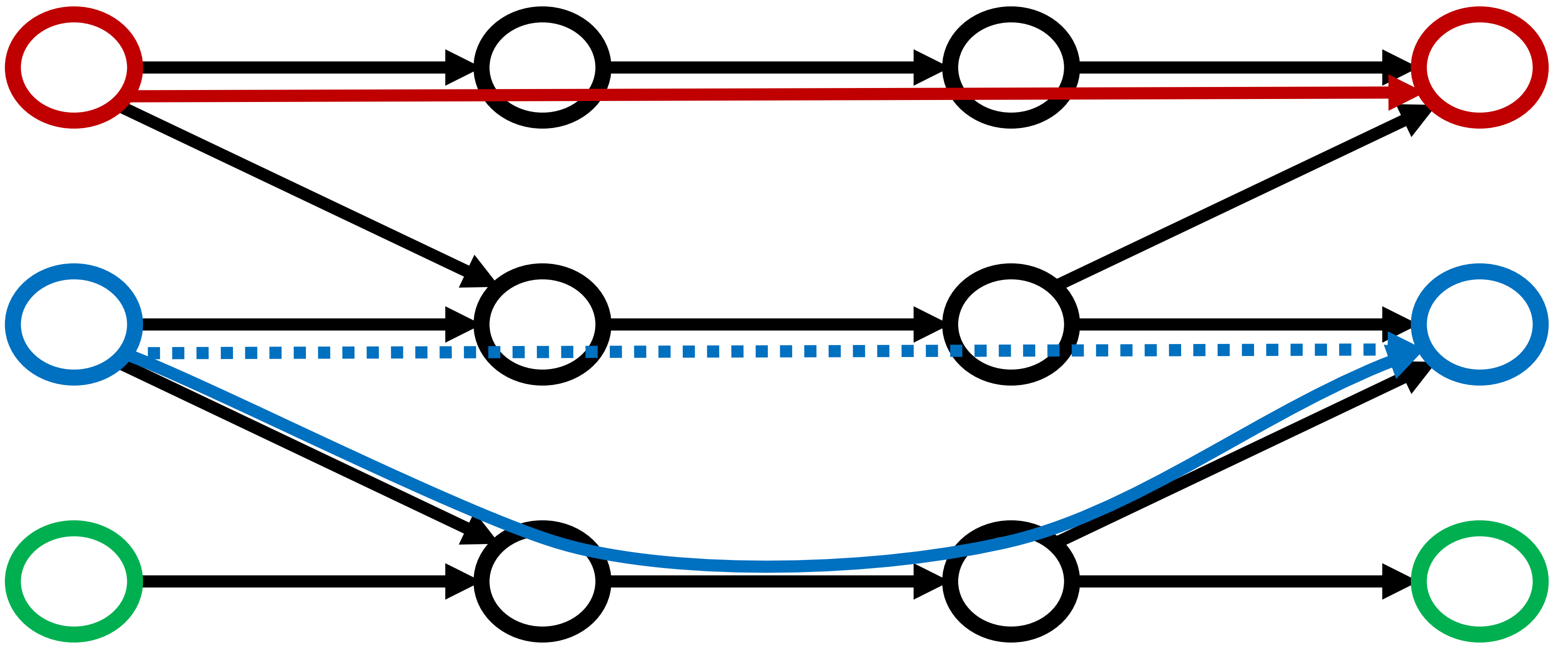
First, Red switches



Round **1**

Unit size flows and capacities

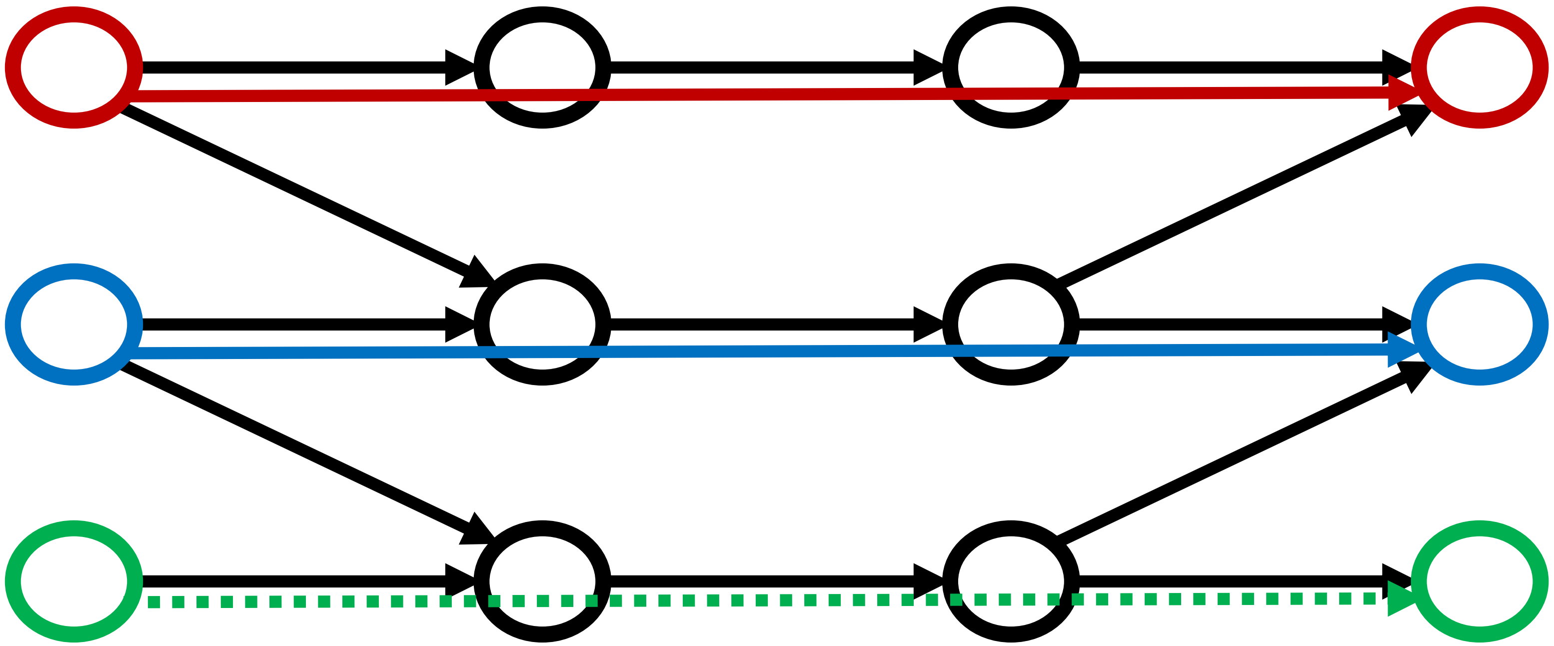
Then, Blue …



Round **2**

Unit size flows and capacities

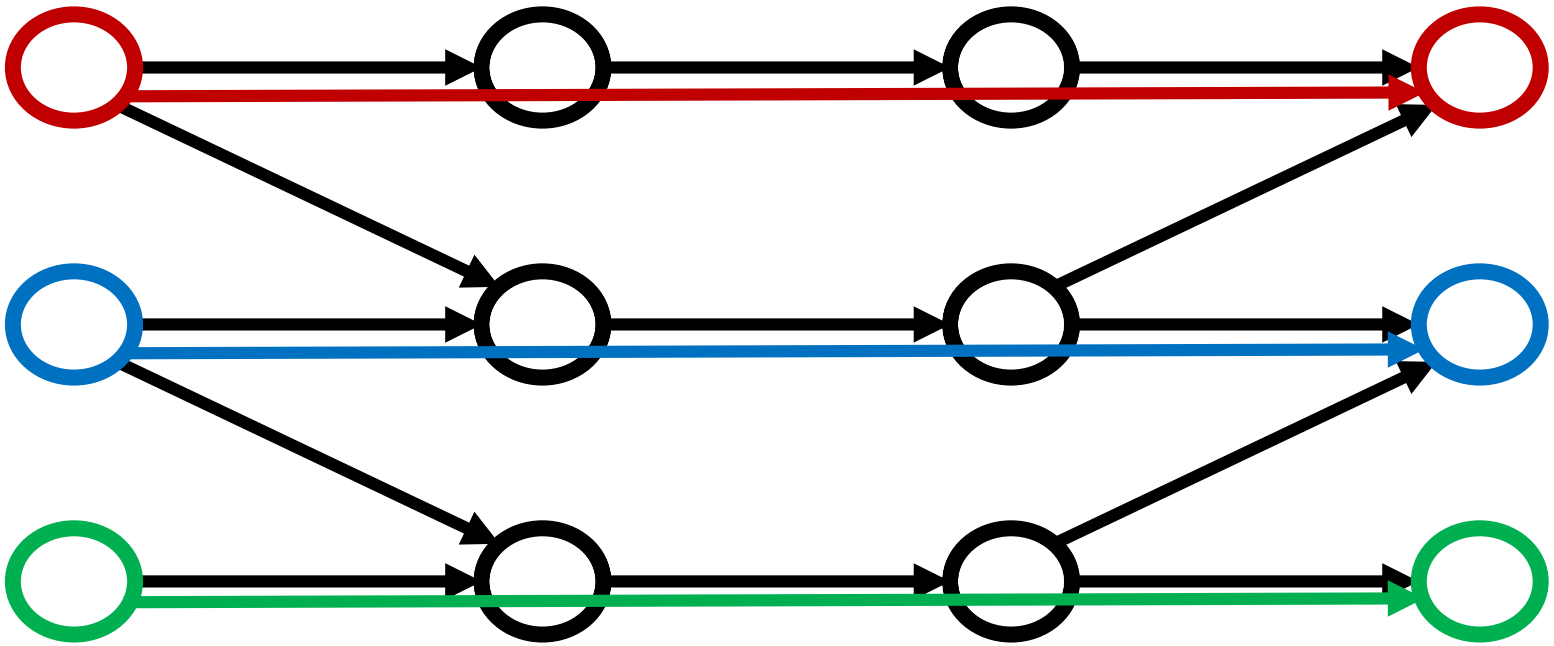
And then, Green …



Round **3**

Unit size flows and capacities

Done



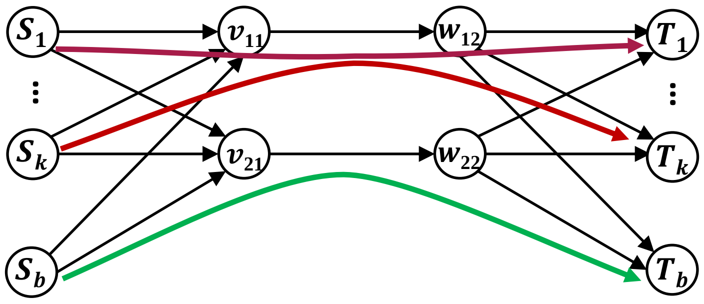
Round **3** *(new)*

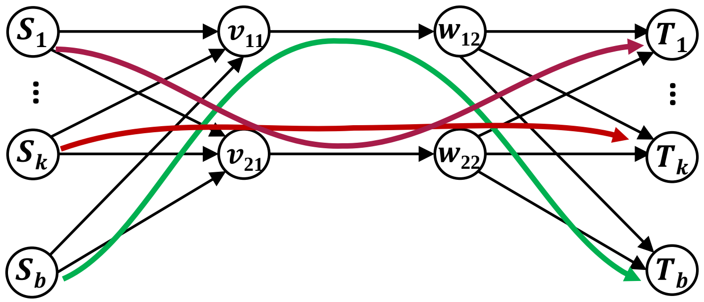
Unit size flows and capacities

How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition





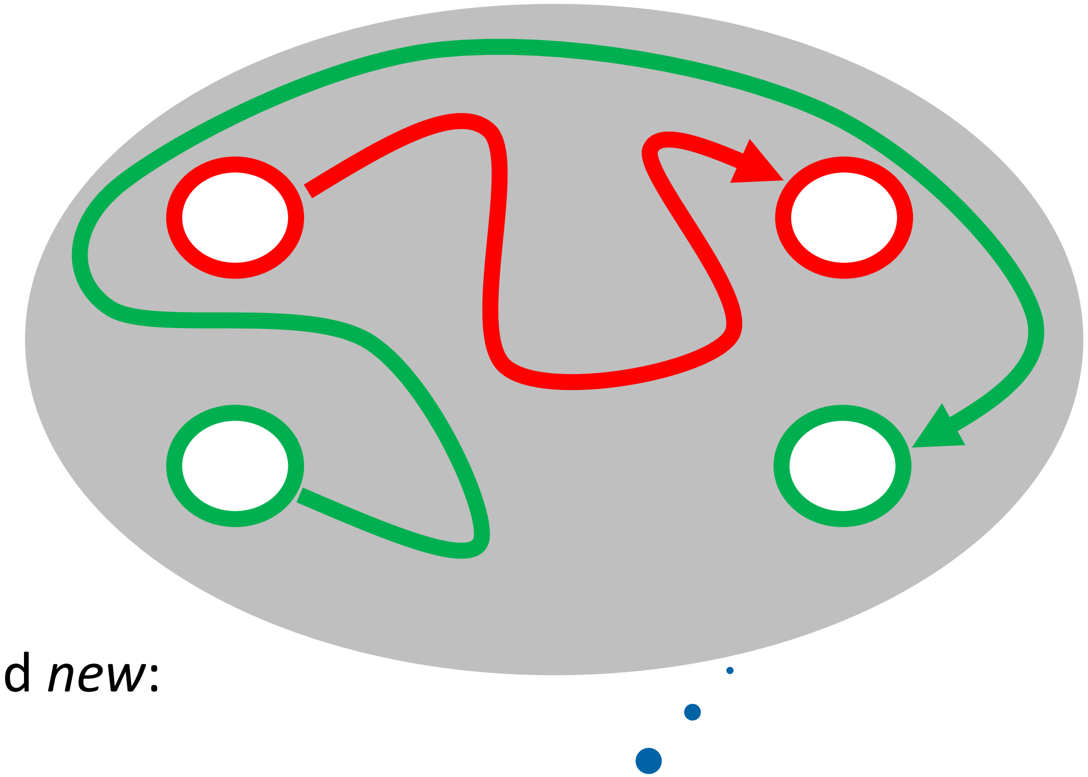
How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition

Intermediate flow allocations not restricted to *old* and *new*:

- NP-hard already for just 2 unit size flows



Hardness intuition: find intermediate path for “storage”

On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds (Foerster, NCA 2017)

How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition

Intermediate flow allocations not restricted to *old* and *new*:

- NP-hard already for just 2 unit size flows

- Is the problem at least in NP?

Some flows might need to move back and forth repeatedly° ☹

On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds (Foerster, NCA 2017)

How hard is this (feasibility)?

Flows may only take *old* or *new* paths:

- NP-hard via reduction from Partition

How about *splittable* flows?

Intermediate flow allocations not restricted to *old* and *new*:

- NP-hard already for just 2 unit size flows

Not clear if the problem is in NP! (It is known to be in EXPTIME)

On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds (Foerster, NCA 2017)

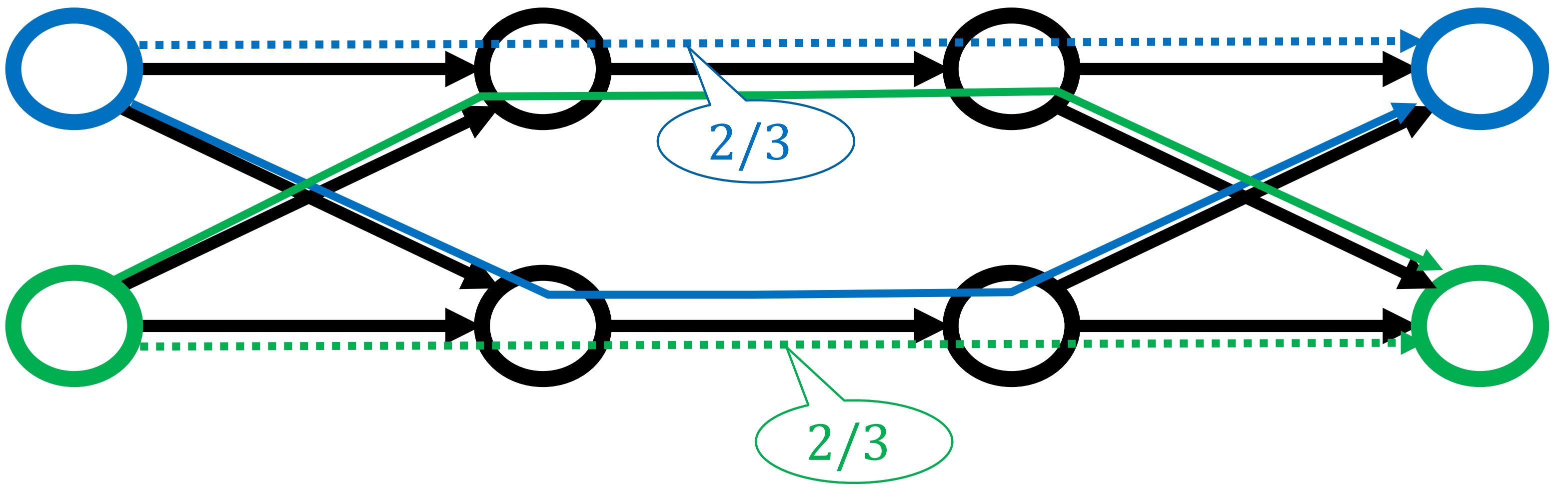
Captures Asynchrony

Consistent Migration of Splittable Flows

Idea: Flows can be on the **old** or **new** route w.r.t. an update

For all edges: $\sum_{\forall F} \max\left(\mathbf{old}, \mathbf{new}\right) \leq capacity$

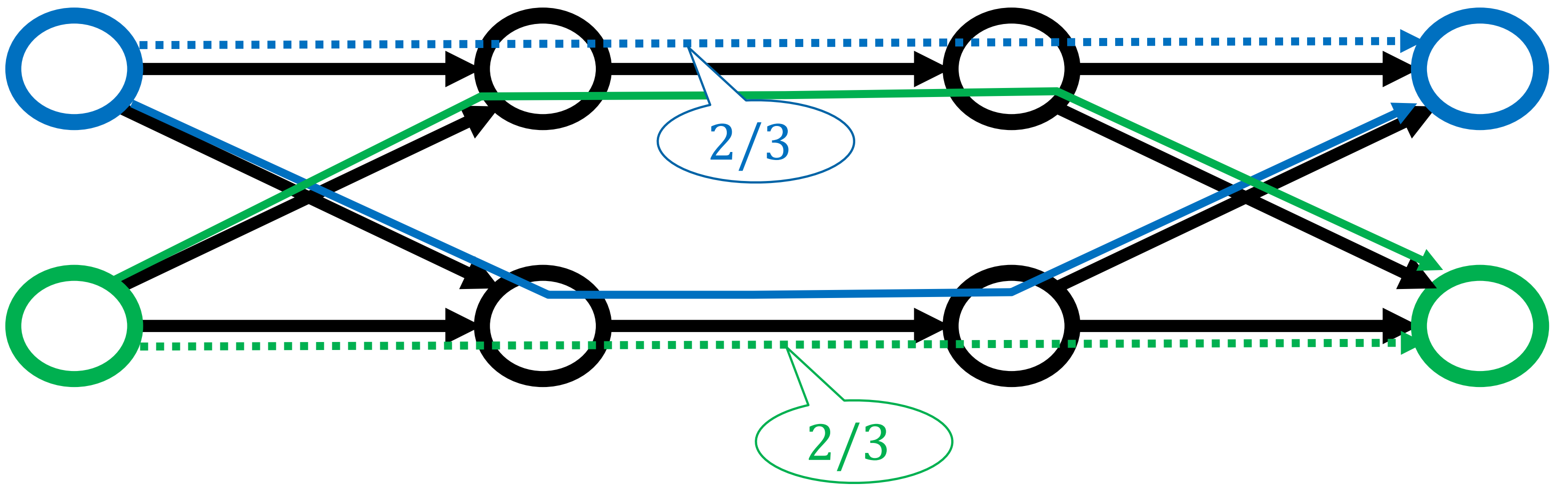
No ordering exists $(2/3 + 2/3 > 1)$



Consistent Migration of Splittable Flows

Approach of *SWAN**: use slack x (i.e., **%**)

- Here $x = 1/3$
- Move slack $x \Rightarrow \lceil 1/x \rceil - 1$ staged partial moves

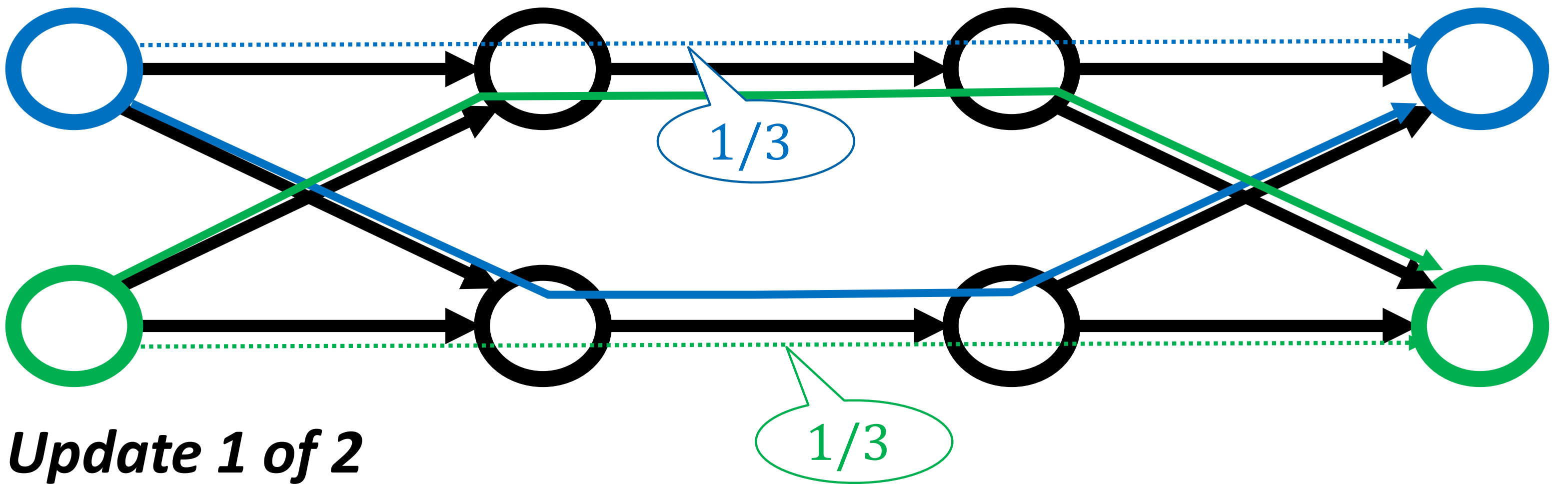


*: Achieving High Utilization with Software-Driven WAN, SIGCOMM 2013

Consistent Migration of Splittable Flows

Approach of *SWAN*: use slack x (i.e., **%**)

- Here $x = 1/3$
- Move slack $x \Rrightarrow \lceil 1/x \rceil - 1$ staged partial moves



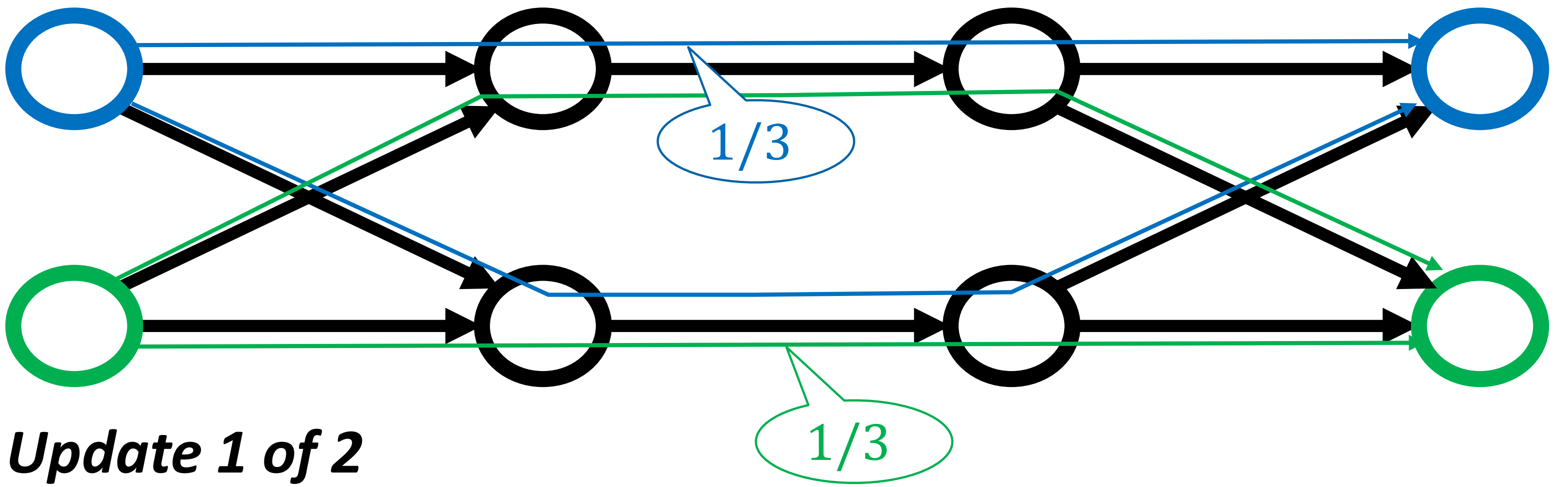
Update 1 of 2

Consistent Migration of Splittable Flows

Approach of *SWAN*: use slack x (i.e., **%**)

Here $x = 1/3$

Move slack $x \Rrightarrow \lceil 1/x \rceil - 1$ staged partial moves



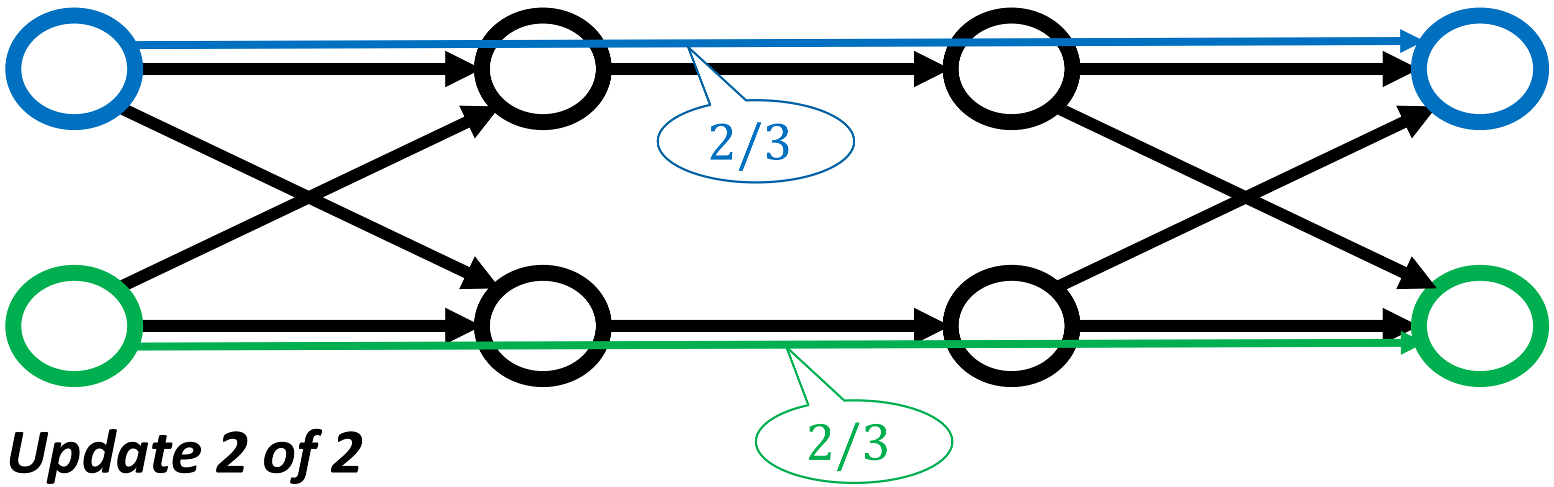
Update 1 of 2

Consistent Migration of Splittable Flows

Approach of *SWAN*: use slack x (i.e., %)

Here $x = 1/3$

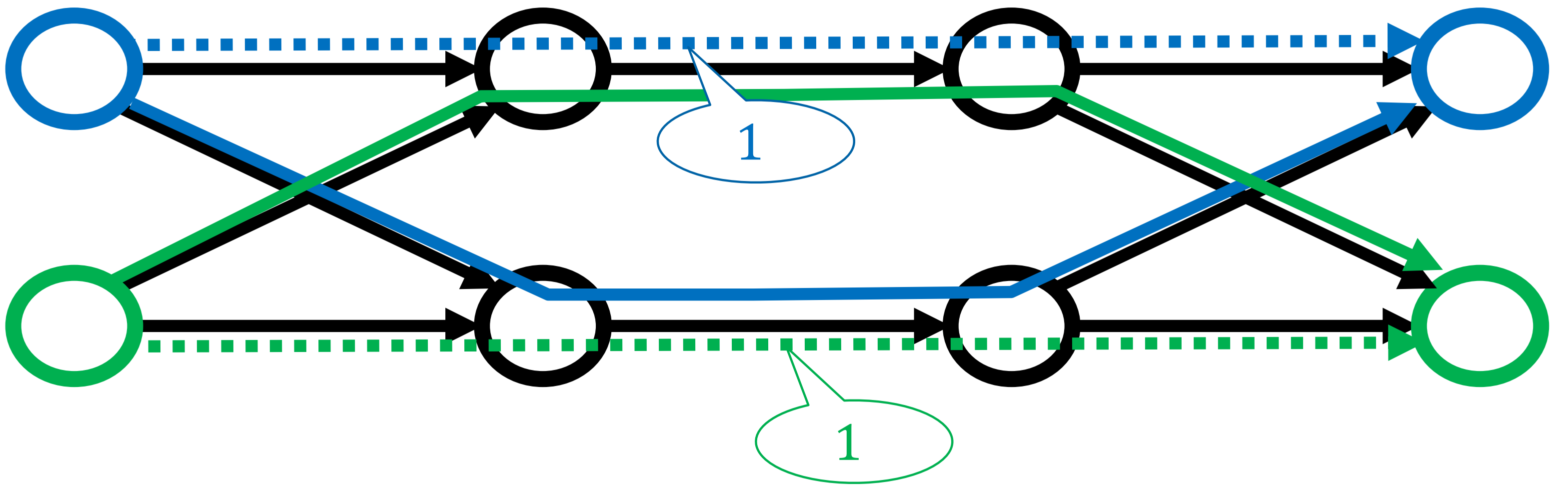
Move slack $x \Rrightarrow \lceil 1/x \rceil - 1$ staged partial moves



Update 2 of 2

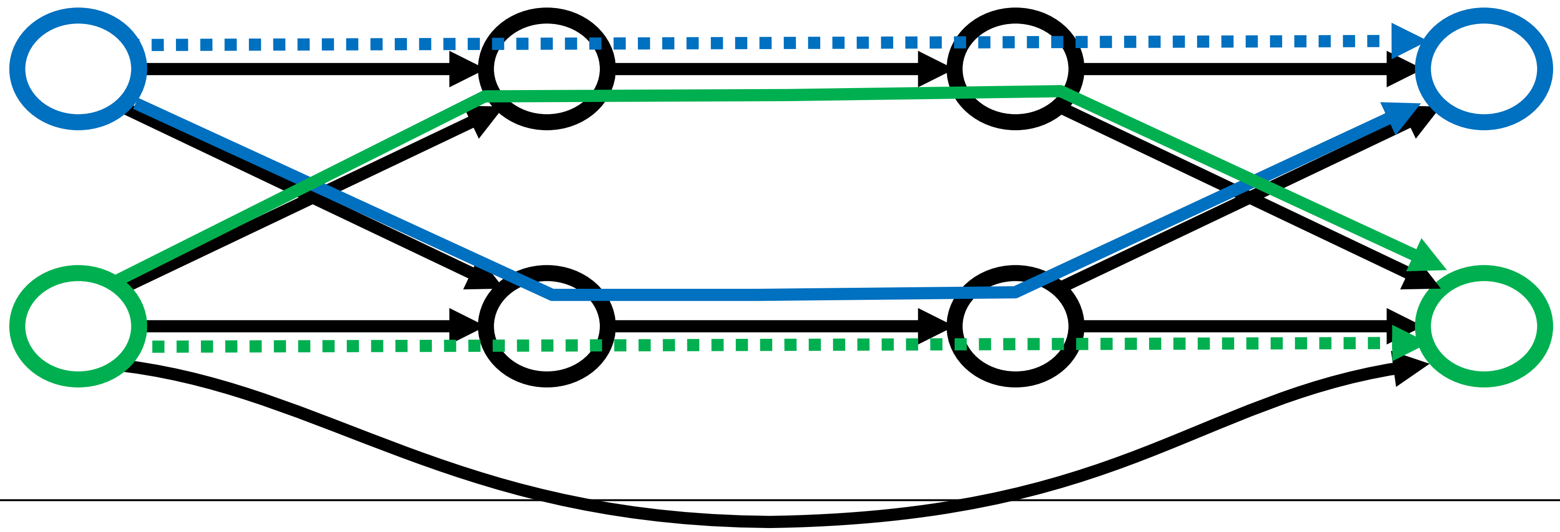
Consistent Migration of Splittable Flows

No slack on flow edges?



Consistent Migration of Splittable Flows

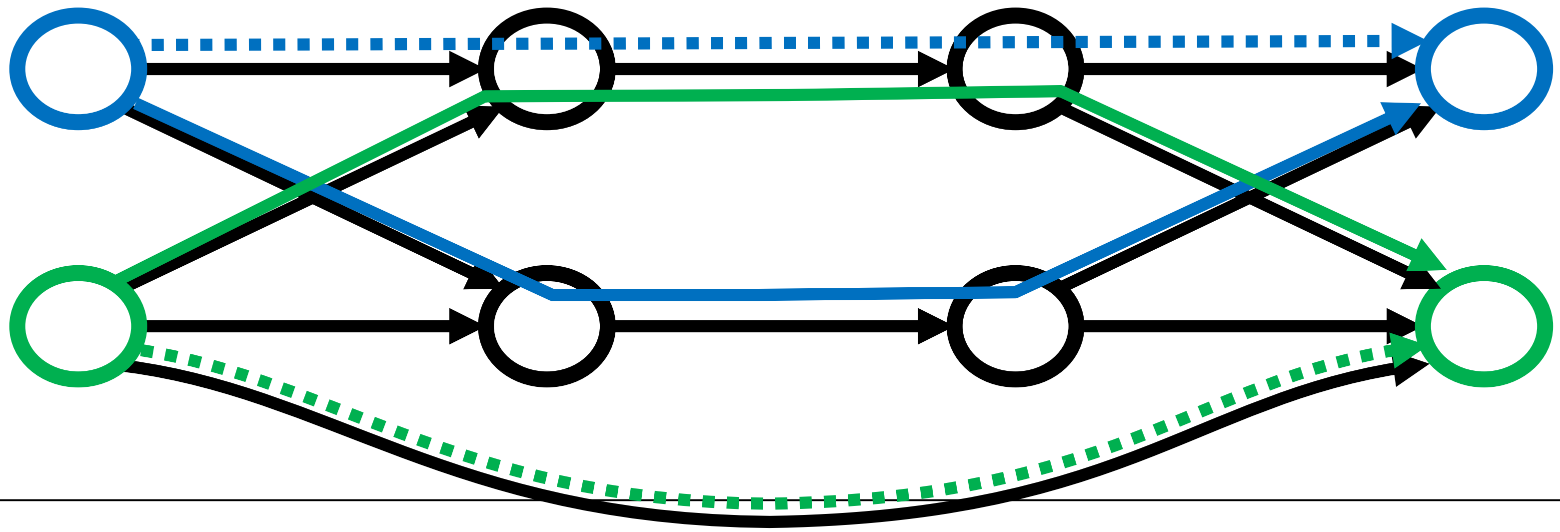
Alternate routes?



Consistent Migration of Splittable Flows

Think: variable swapping of b & g

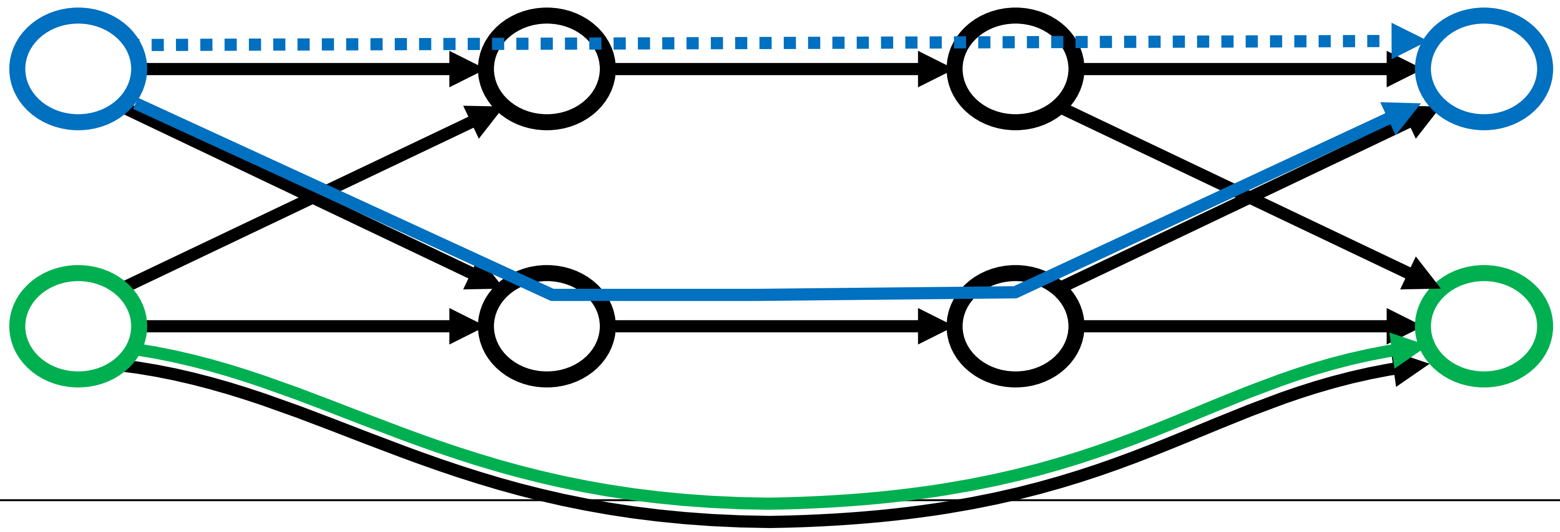
1. $x \coloneqq b$, 2. b $\coloneqq g$, 3. $g \coloneqq x$



Consistent Migration of Splittable Flows

Think: variable swapping of b & g

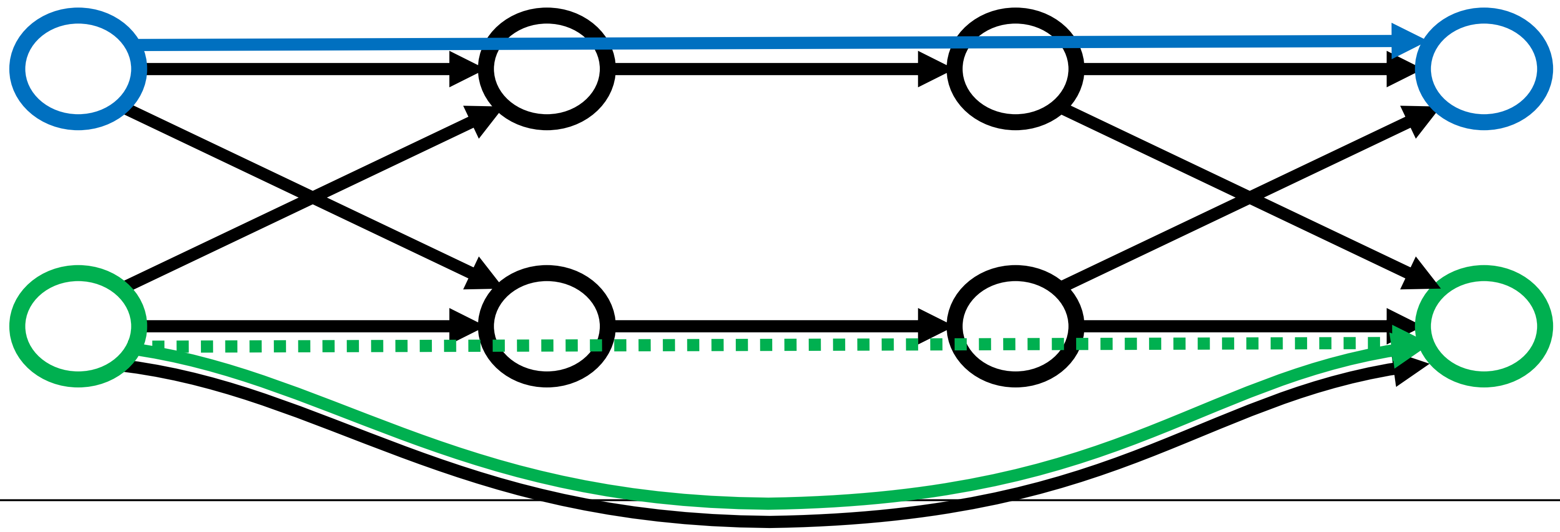
1. $x \coloneqq b$, 2. $\mathrm{b} \coloneqq g$, 3. $g \coloneqq x$



Consistent Migration of Splittable Flows

Think: variable swapping of b & g

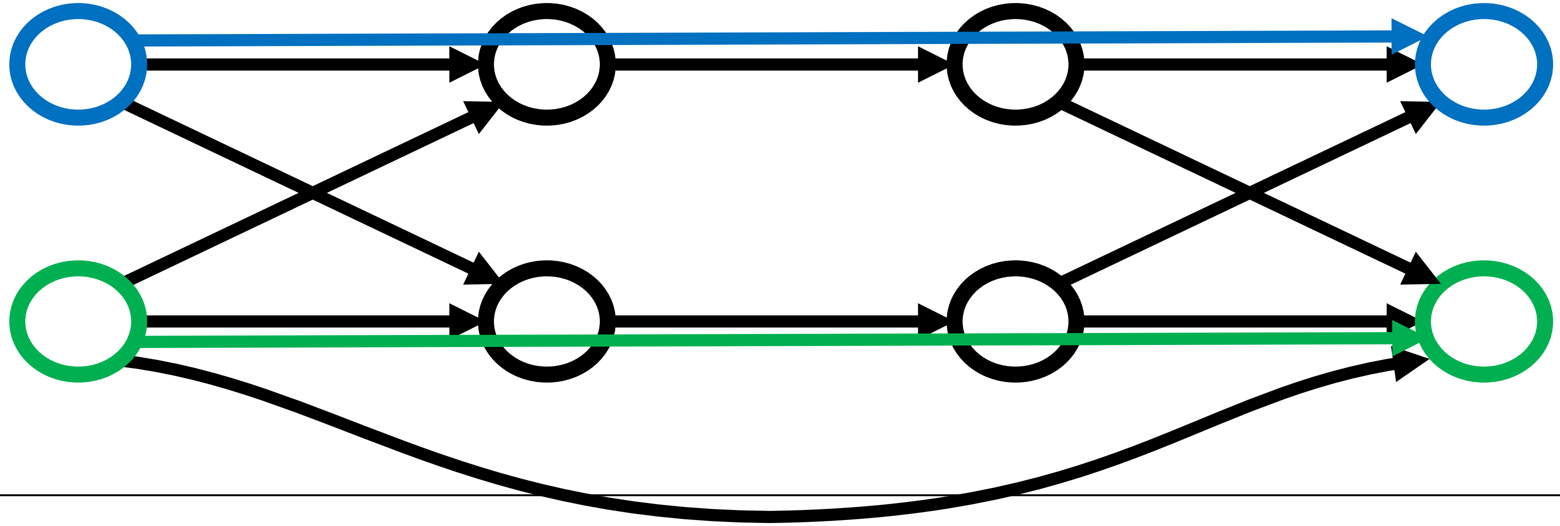
1. $x \coloneqq b$, 2. b $\coloneqq g$, 3. $g \coloneqq x$



Consistent Migration of Splittable Flows

SWAN: LP-approach with binary search

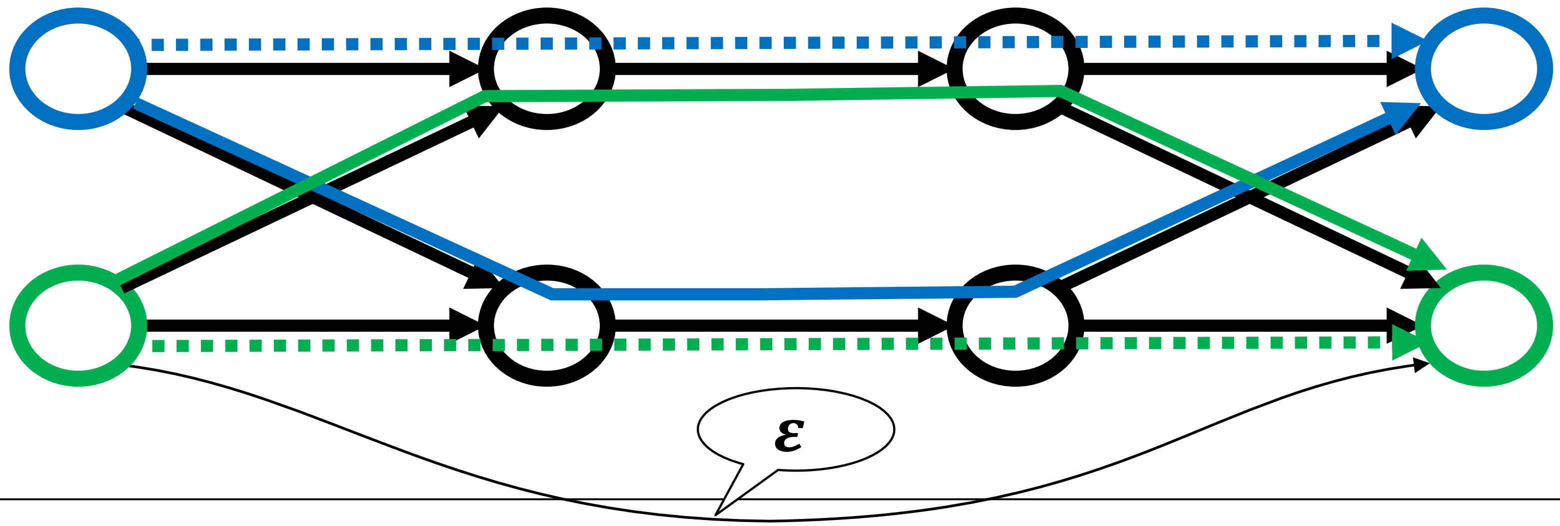
1 update? 2 updates? 4 updates? …



Consistent Migration of Splittable Flows

SWAN: LP-approach with binary search

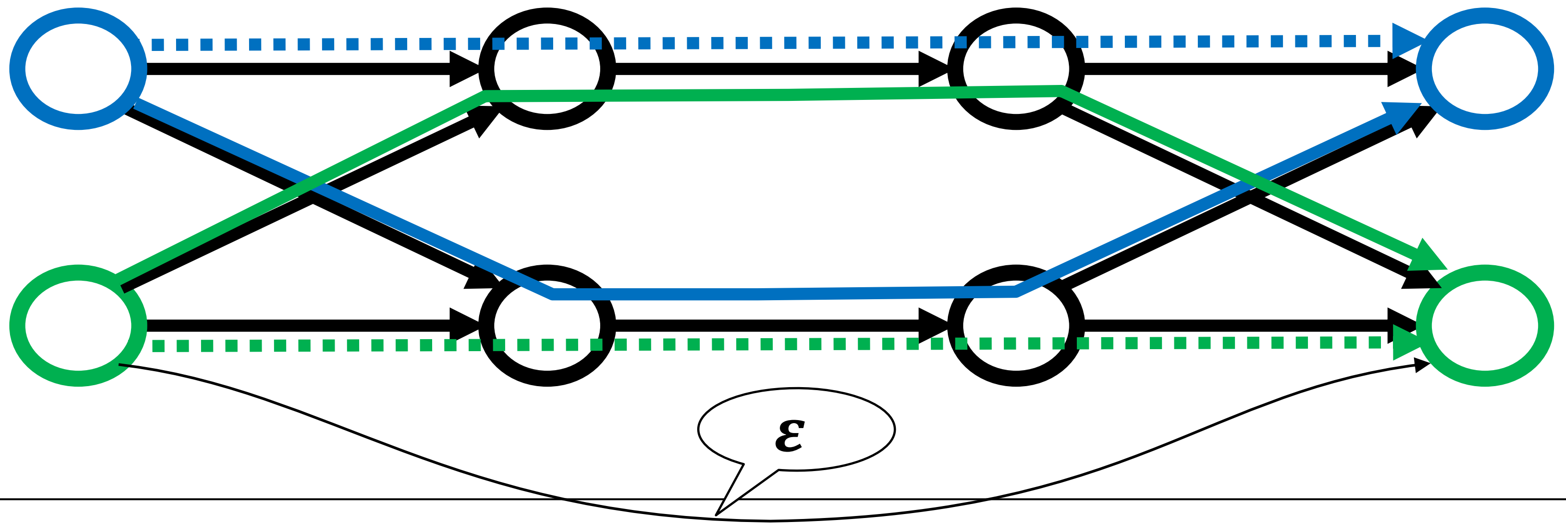
1 update? 2 updates? 4 updates? …



Consistent Migration of Splittable Flows

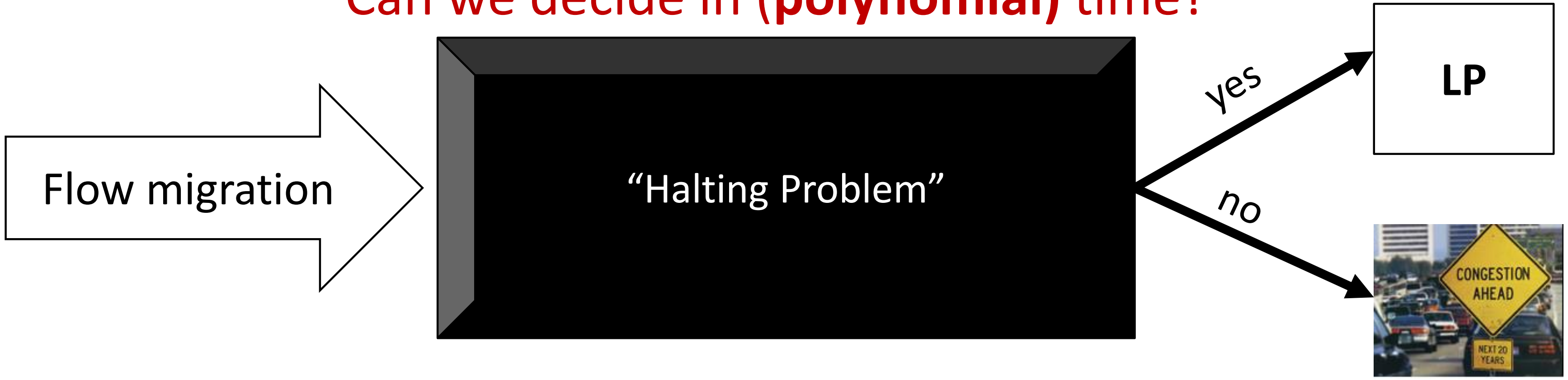
SWAN: LP-approach with binary search

$\Theta(1/\varepsilon)$ updates ☹



Consistent Migration of Splittable Flows

Can we decide in (**polynomial**) time?



To Slack or not to Slack?

Slack of x on all flow edges?

$\lceil 1/x \rceil - 1$ updates

To Slack or not to Slack?

What if not?

Try to create slack

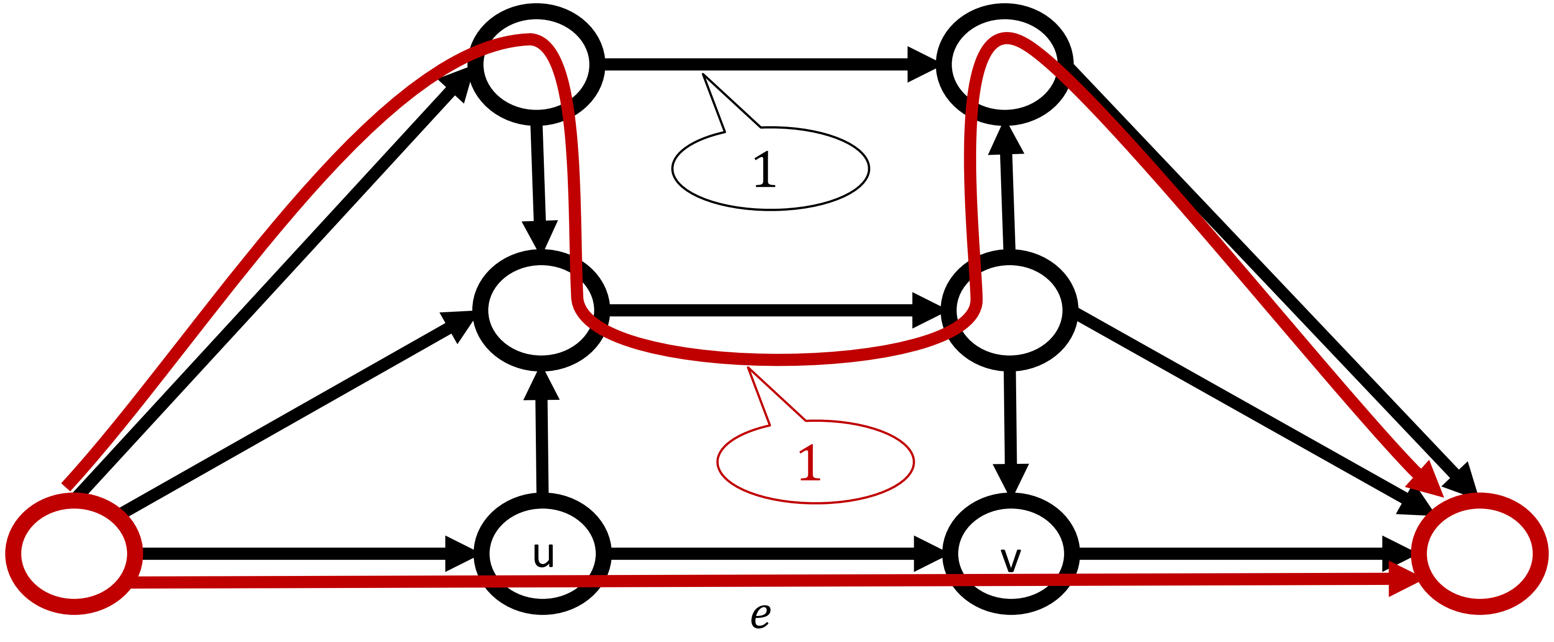
To Slack or not to Slack?

Combinatorial approach

Augmenting paths

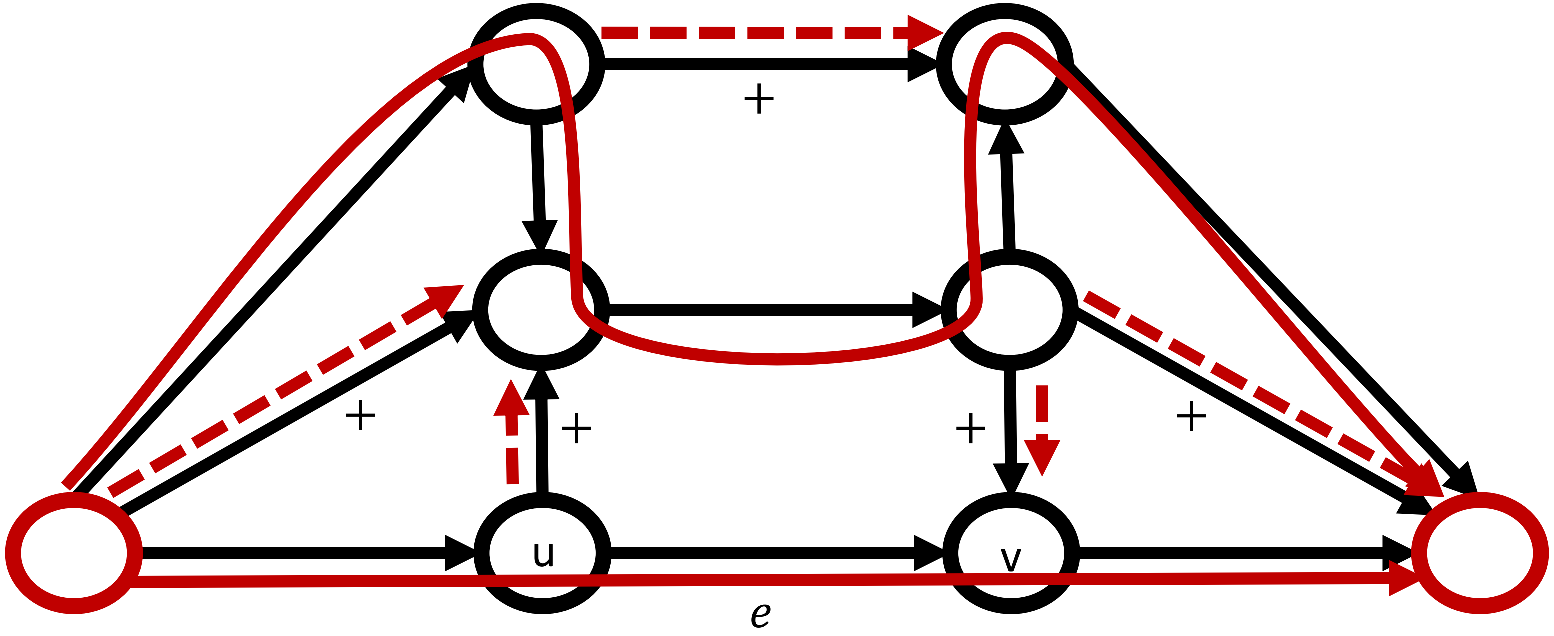
Combinatorial Approach

Move single commodities at a time



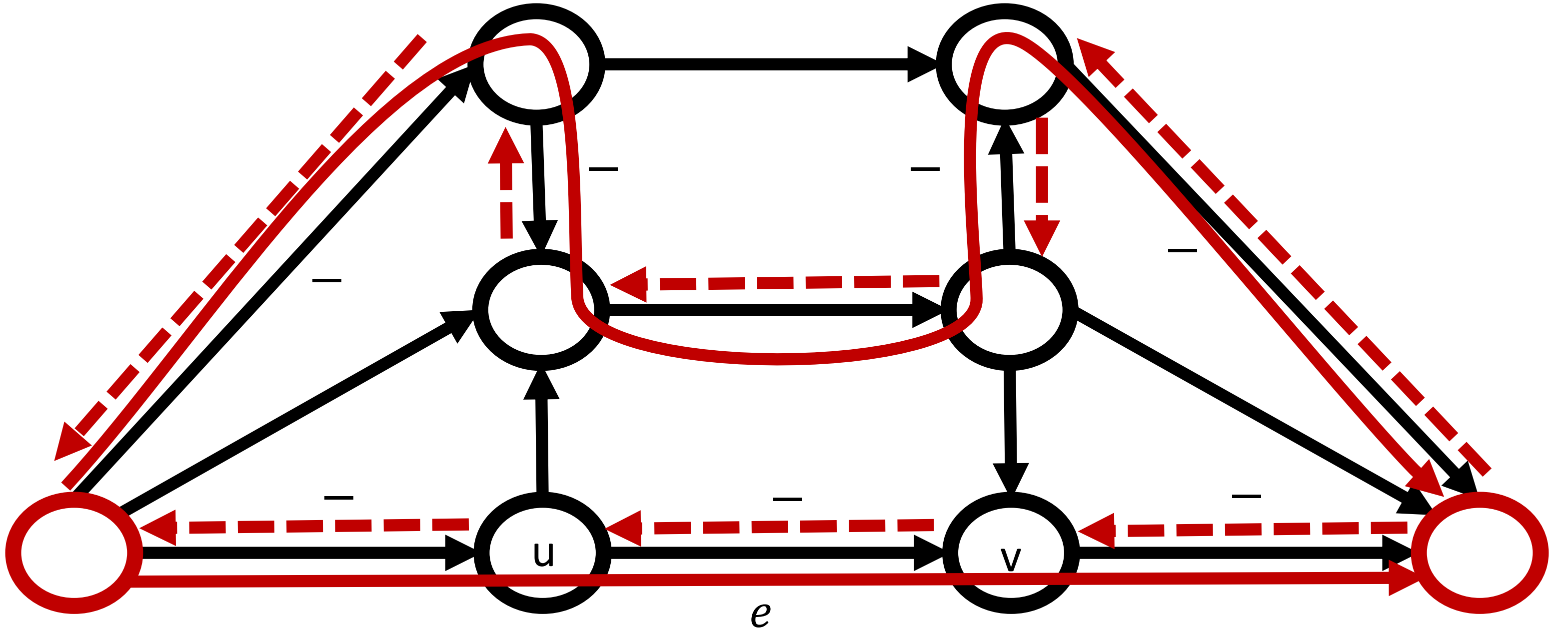
Combinatorial Approach

Where to increase flow?



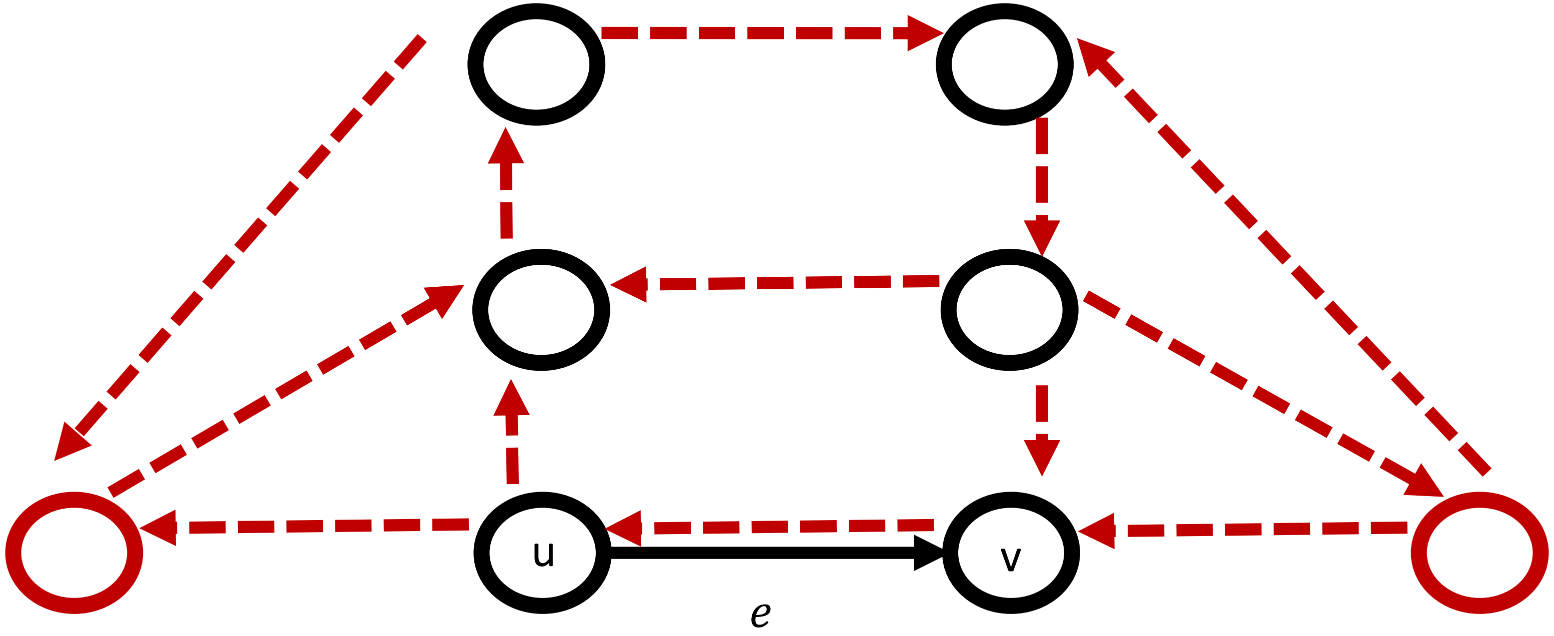
Combinatorial Approach

Where to push back flow?



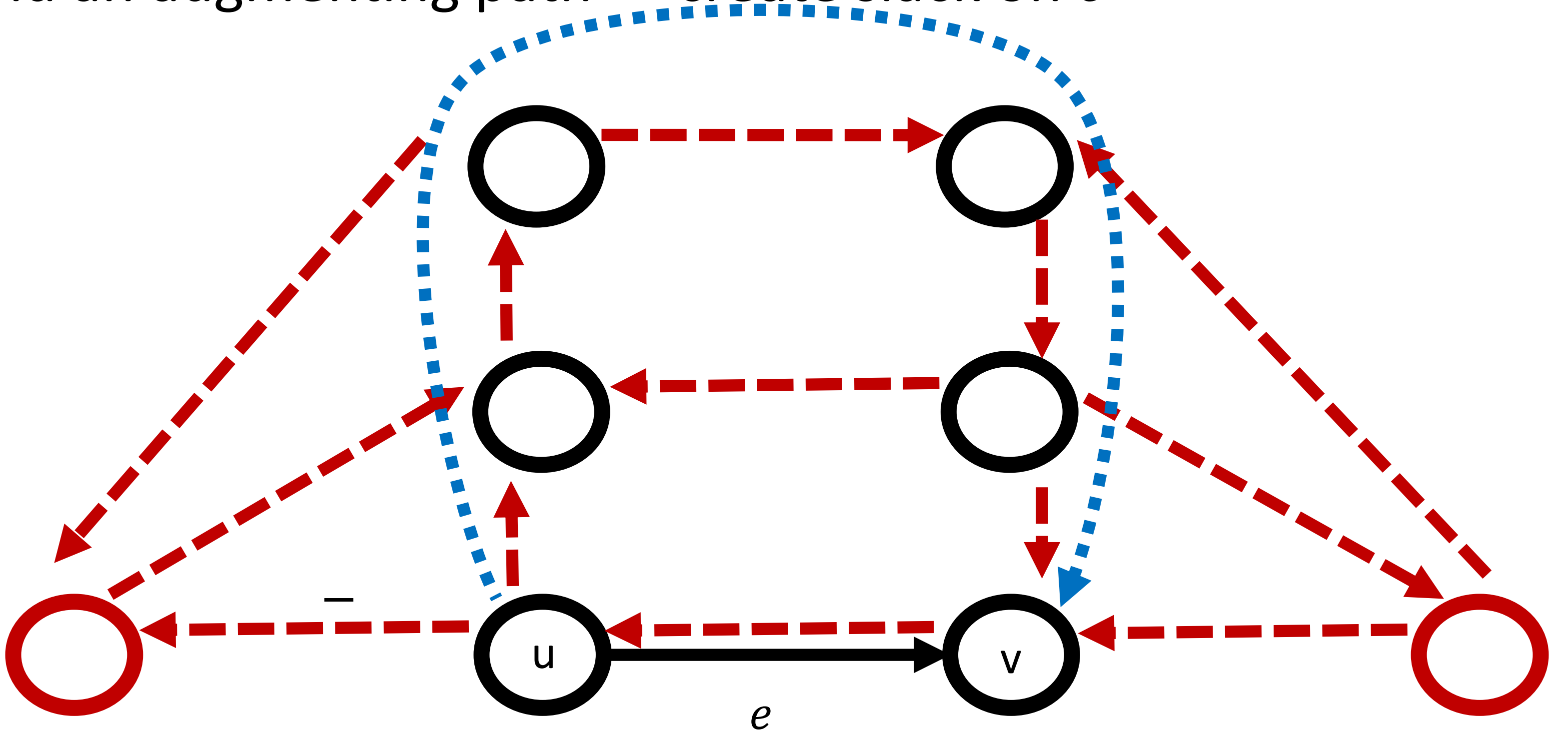
Combinatorial Approach

Resulting residual network



Combinatorial Approach

We found an augmenting path ⇒ create slack on e



High-level Algorithm Idea

- No slack on flow edges? Find augmenting paths
  - On both initial and desired state (updates can be performed in reverse)
  - Success? Use *SWAN* method to migrate

- Can't create slack on some flow edge?
  - Consistent migration impossible
    By contradiction (else augmenting paths would create slack)

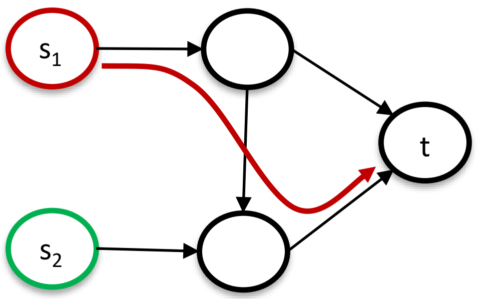
- Runtime: $O(Fm^3)$
  - (F being #commodities, m being #edges)

On Consistent Migration of Flows in SDNs. S. Brandt, K.-T. Foerster, R. Wattenhofer, INFOCOM 2016

Maybe surprisingly:
If the new flows fit in somehow,
we can migrate consistently!

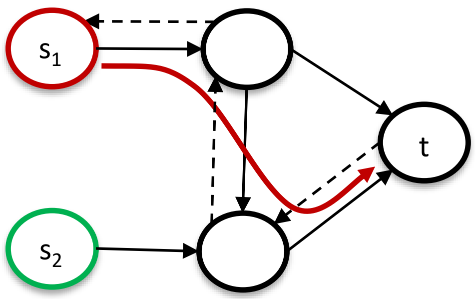
Open problems for scheduling flow migration

- What happens when we can pick the new paths?
  - Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

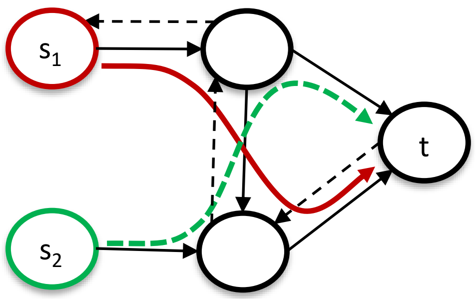


size of each flow: 1

capacity of links: 1

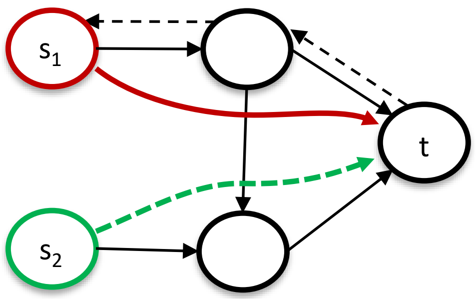


size of each flow: 1
capacity of links: 1



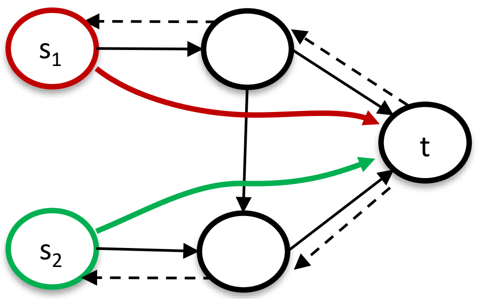
size of each flow: 1

capacity of links: 1



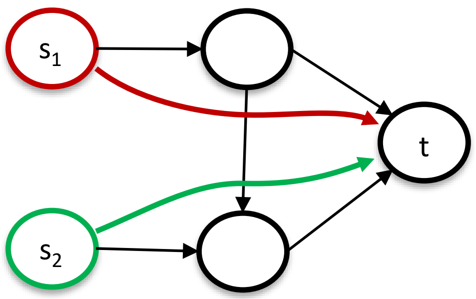
size of each flow: 1

capacity of links: 1



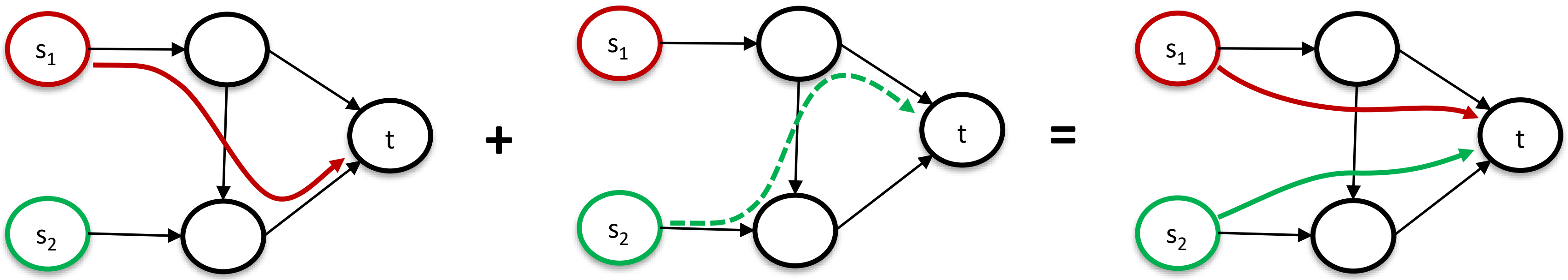
size of each flow: 1

capacity of links: 1



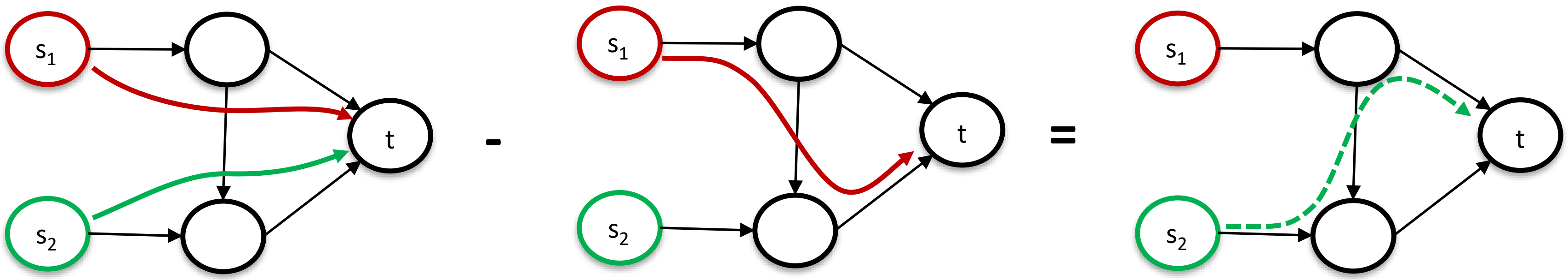
size of each flow: 1

capacity of links: 1



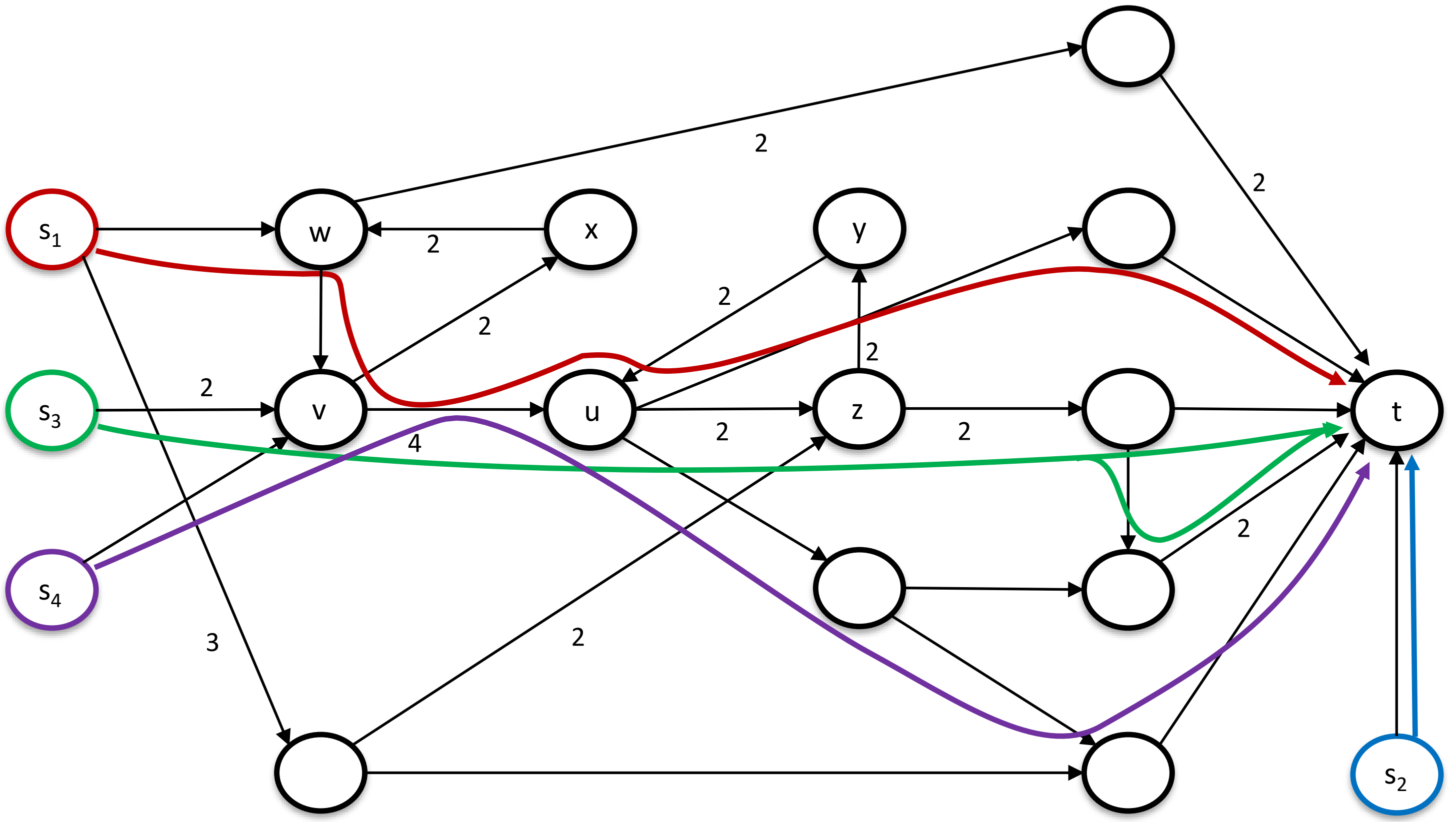
size of each flow: 1

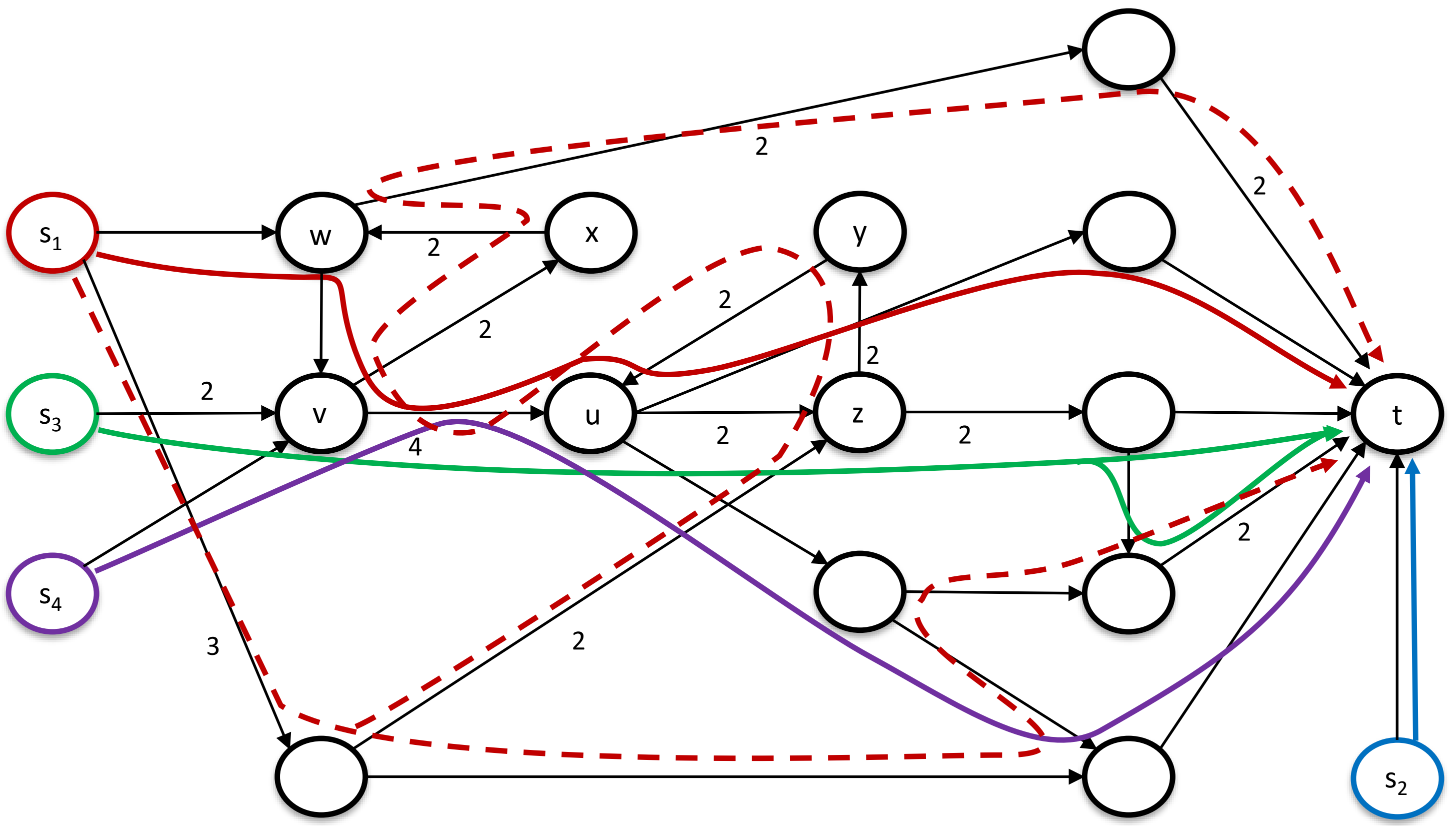
capacity of links: 1



size of each flow: 1

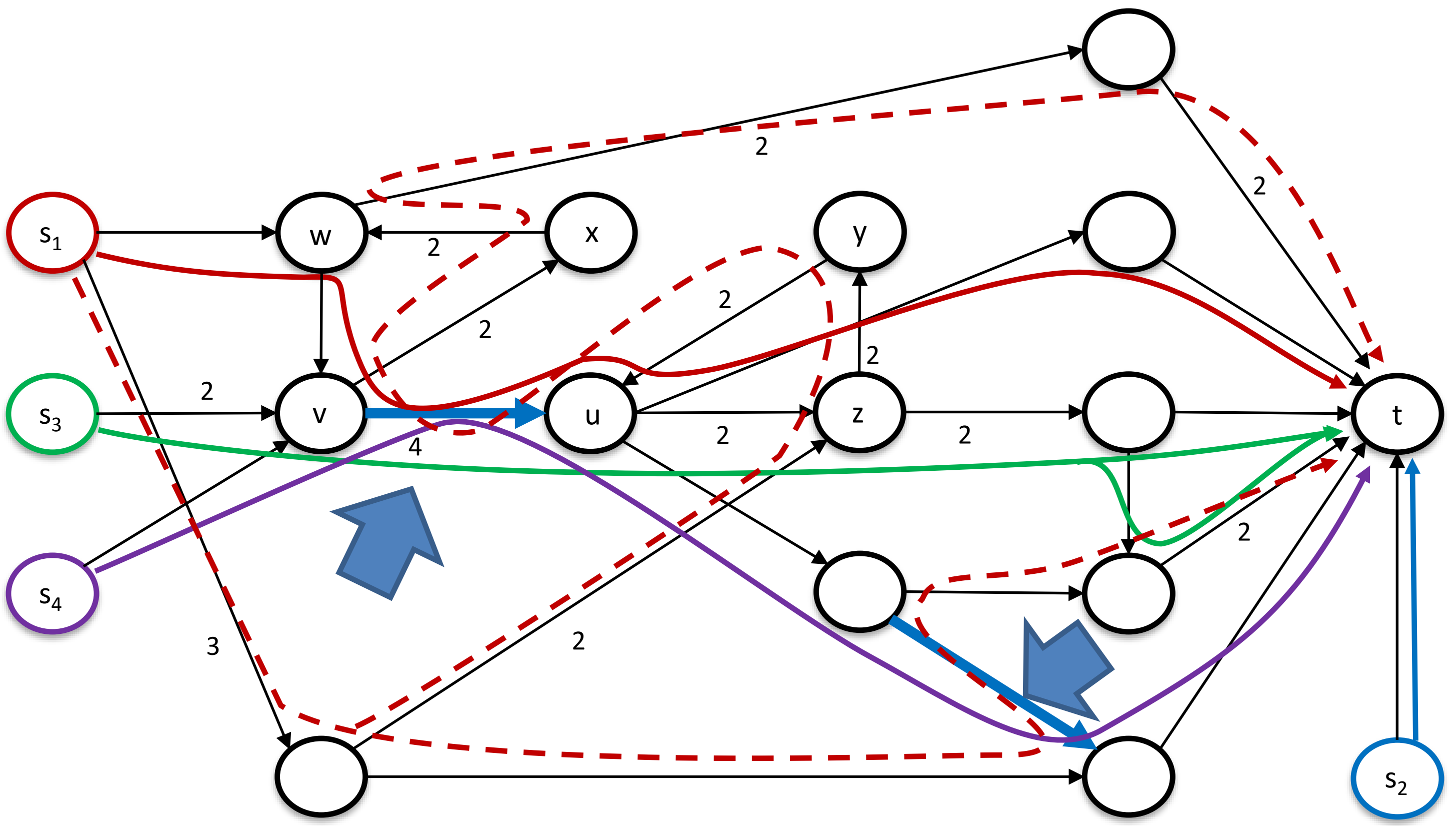
capacity of links: 1


s1
w
x
y
v
u
z
t
s3
s4
s2
2
2
2
2
2
2
2
2
2
4
2
2
3
2
2



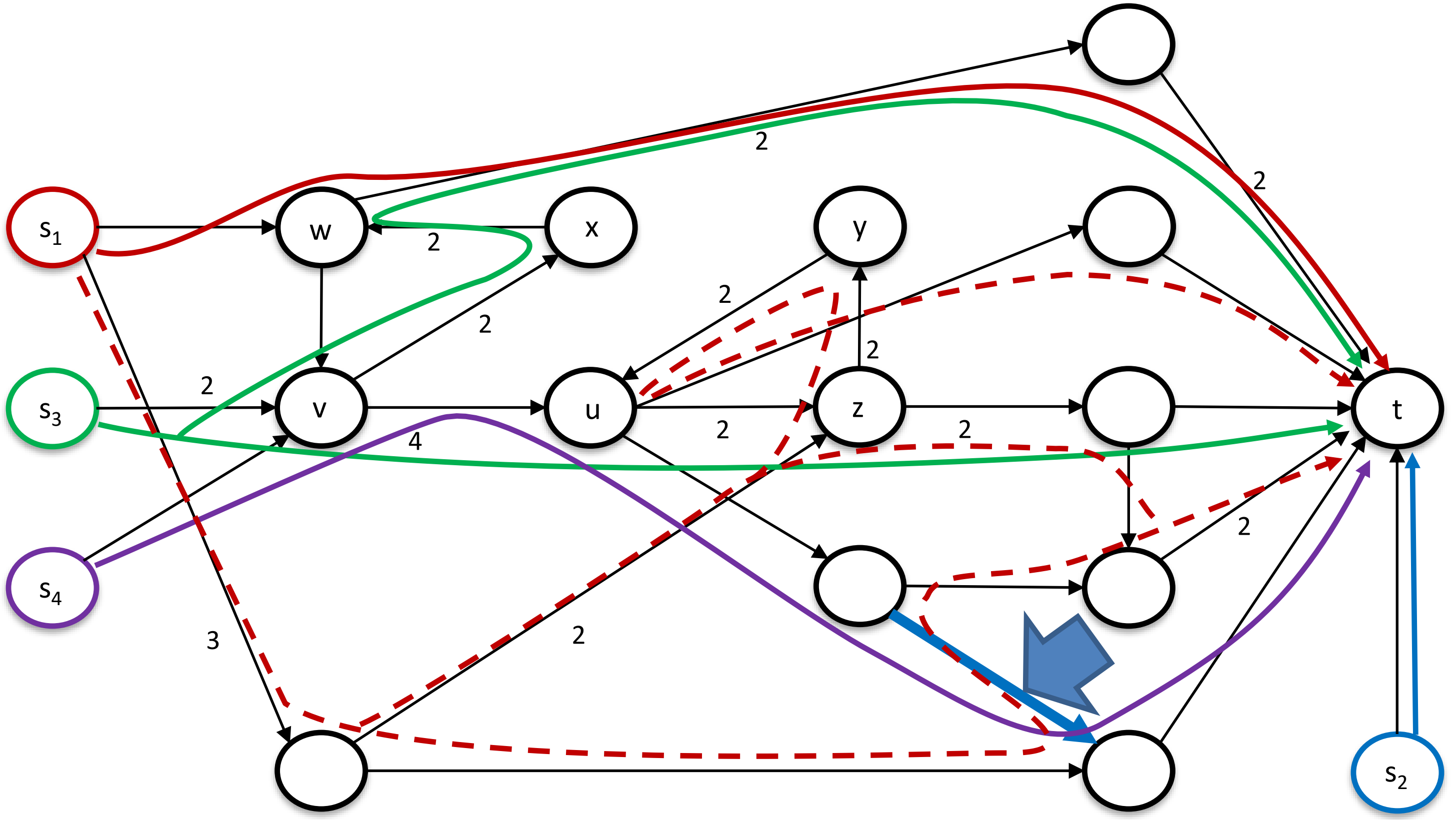
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)



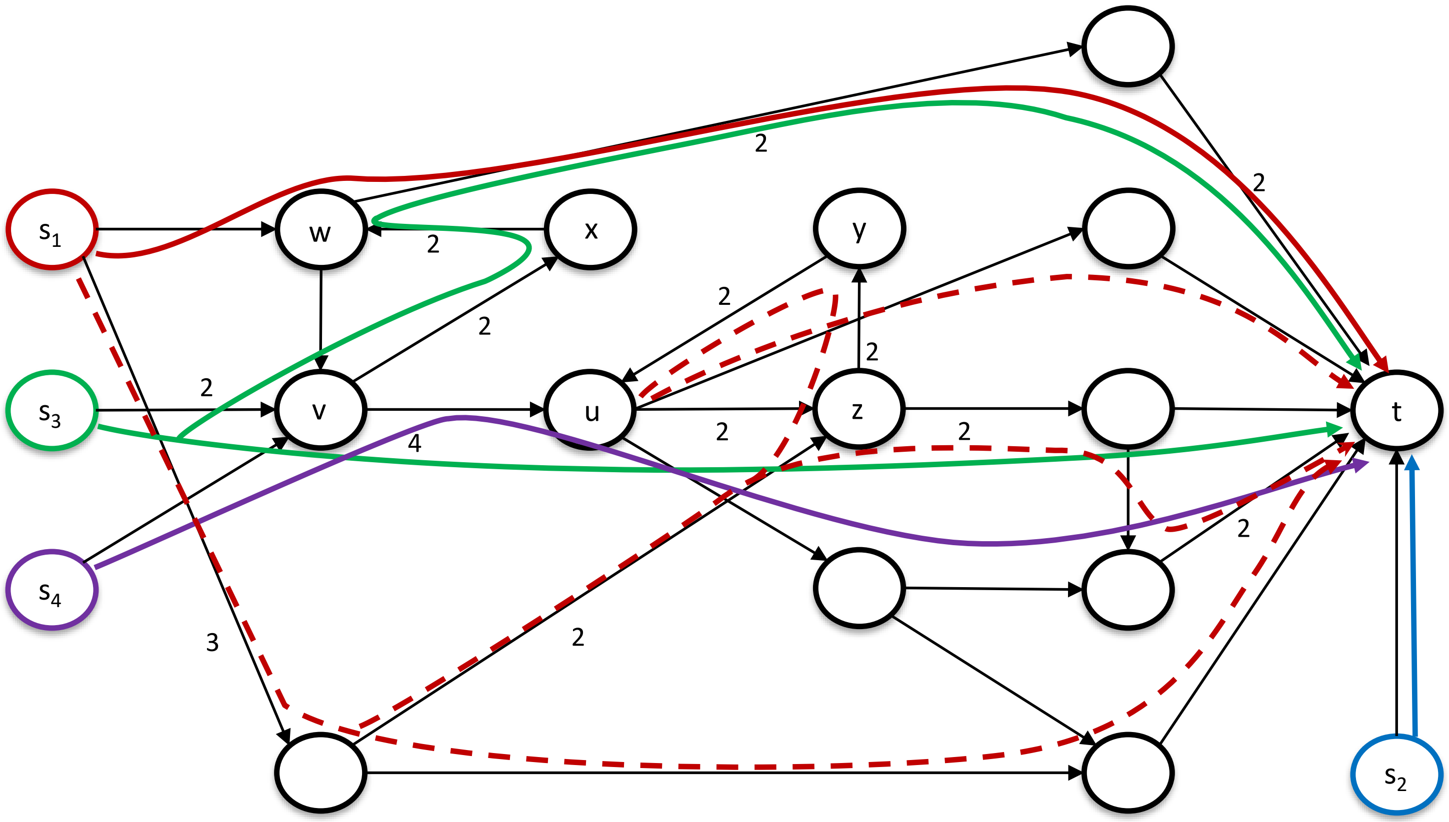
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)



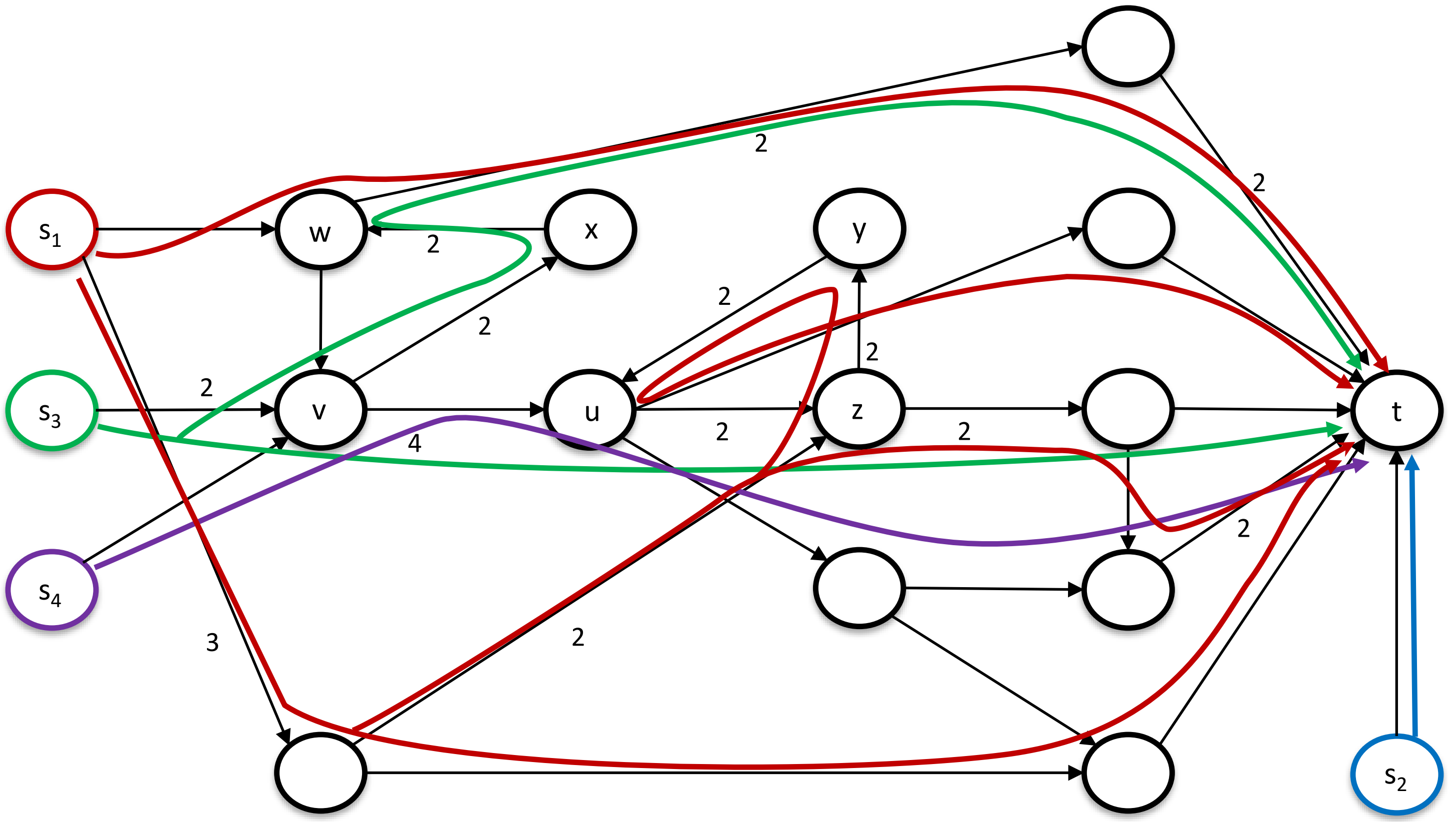
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)



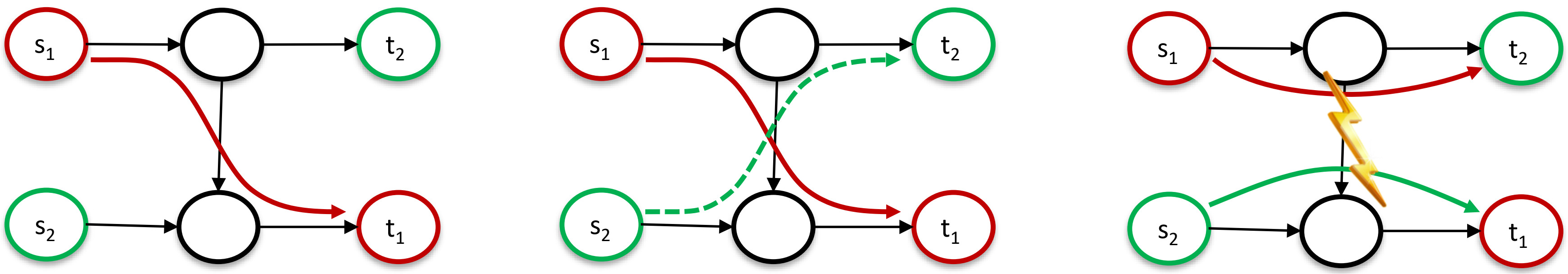
size of flows: 1, 2, 1, 1

capacity of links: 1 (or marked)

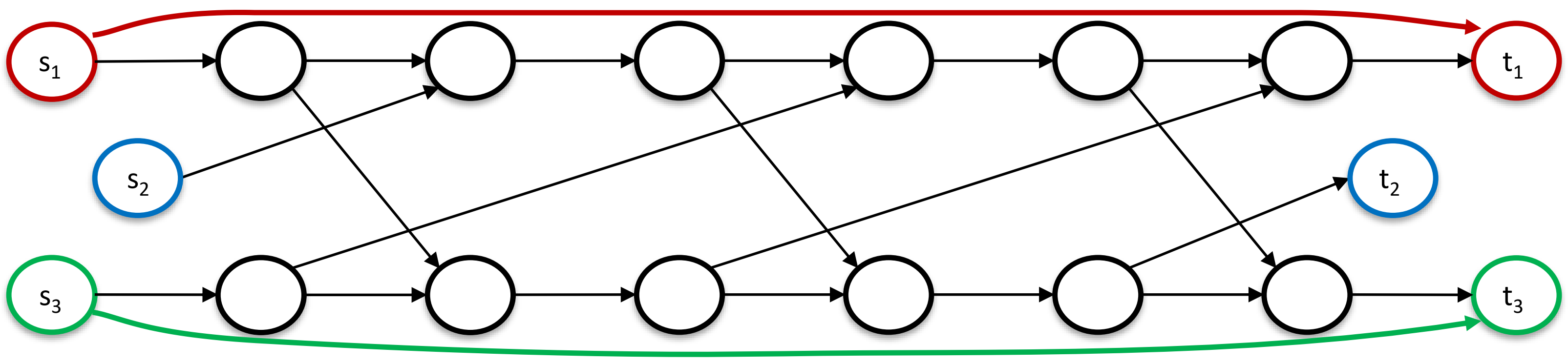


size of flows: 1+3=4, 2, 1, 1

capacity of links: 1 (or marked)

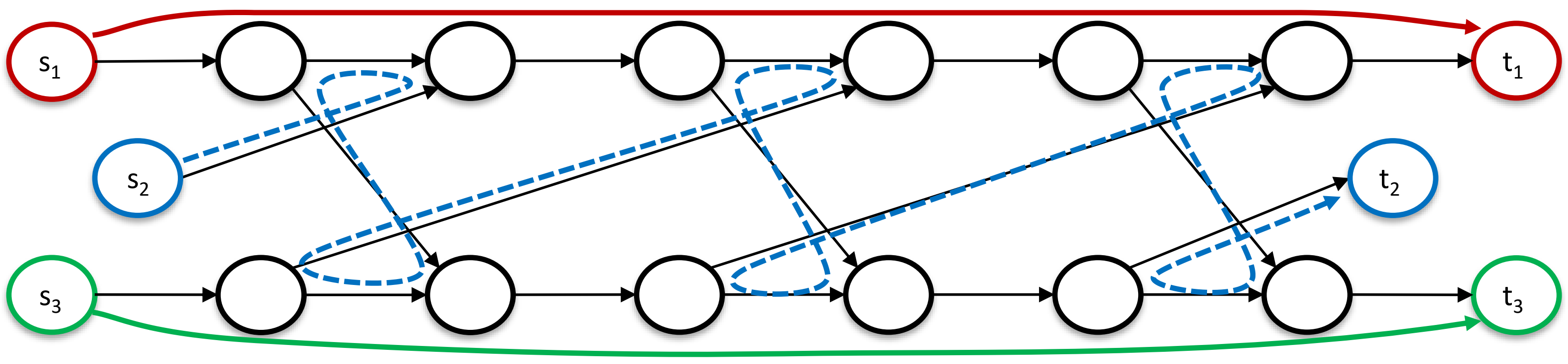


- Flows end up at the wrong destination!
- So let’s stick with augmenting flows that don’t mix destinations



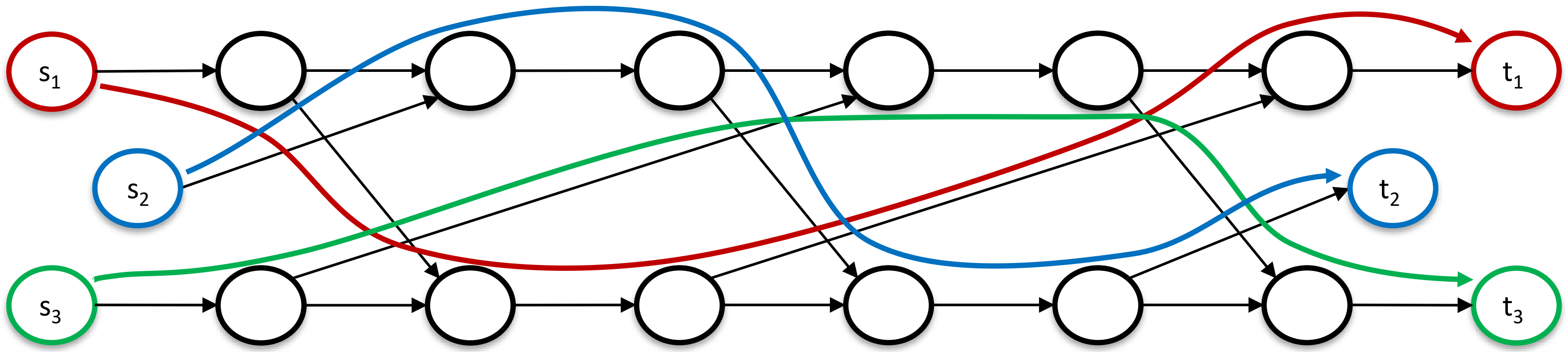
size of each flow: 1

capacity of each links: 1



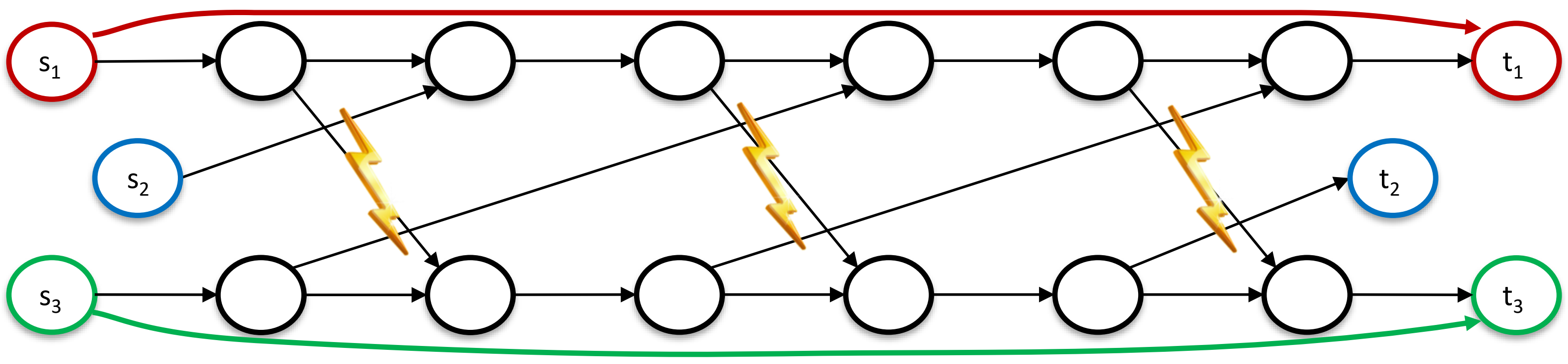
size of each flow: 1

capacity of each links: 1



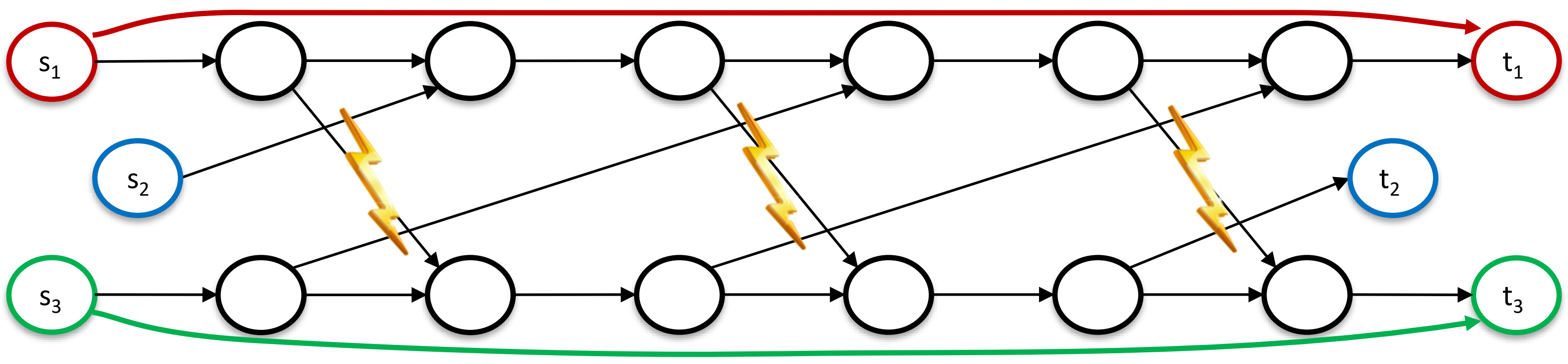
size of each flow: 1

capacity of each links: 1



size of each flow: 1

capacity of each links: 1



"it is unlikely that similar techniques can be developed for constructing multicommodity flows"

[Hu, 1963]

size of each flow: 1

capacity of each links: 1

universität wien

Maybe surprisingly:
If the new flows fit in somehow,
we can migrate consistently!

Open Problems for scheduling flow migration

- What happens when we can pick the new paths?
  - Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

Maybe further development
needs better understanding of
augmenting flows?

- Unsplittable flow migration:
  - In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  - ”Interesting“ polynomial cases?

universität wien

Maybe surprisingly:
If the new flows fit in somehow,
we can migrate consistently!

Open Problems for scheduling flow migration

- What happens when we can pick the new paths?
  - Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

- Unsplittable flow migration:
  - In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  - ”Interesting“ polynomial cases?

Maybe further development
needs better understanding of
augmenting flows?

More open questions and specifics:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019

universität wien

Maybe surprisingly:
If the new flows fit in somehow,
we can migrate consistently!

Open Problems for scheduling flow migration

- What happens when we can pick the new paths?
  - Idea: Fit the flows in, does not matter where
    - Only studied so far for a single destination and multiple sources [Brand, Foerster, Wattenhofer, PMC 2017]

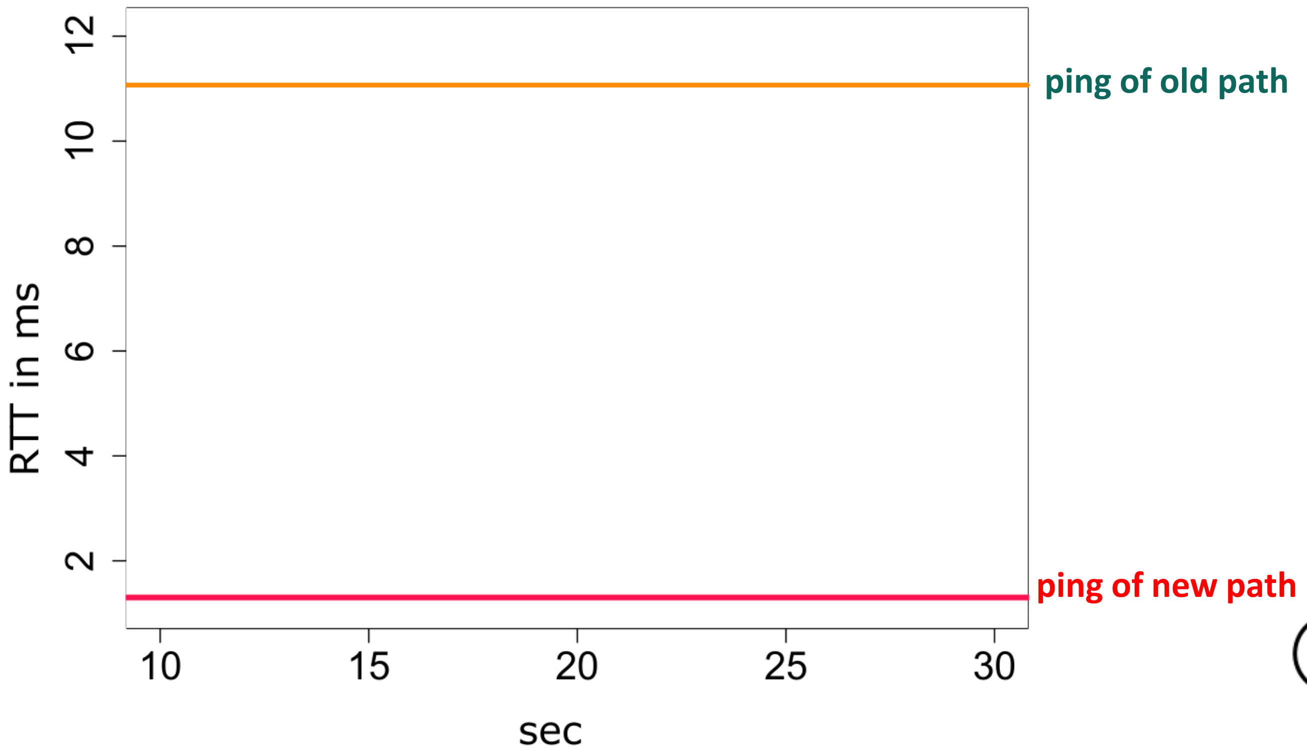
Maybe further development
needs better understanding of
augmenting flows?

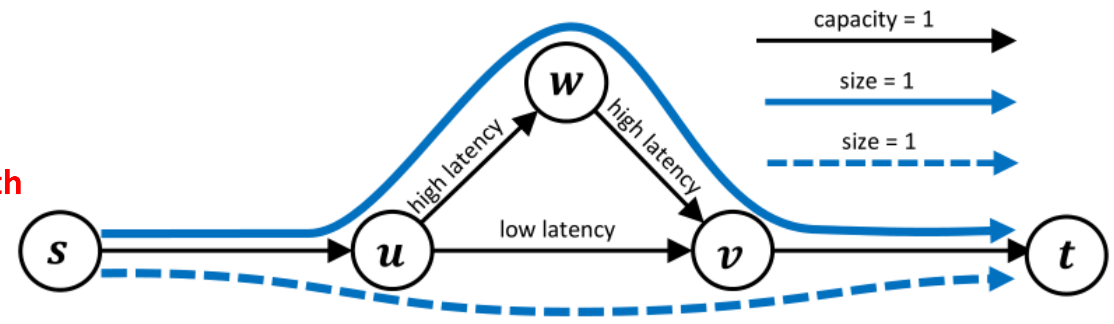
- Unsplittable flow migration:
  - In general: NP-, PSPACE-, or EXPTIME-complete?
    - (recall: flows might need to switch back and forth repeatedly)
  - ”Interesting“ polynomial cases?

- What happens when considering **Link Latency**?

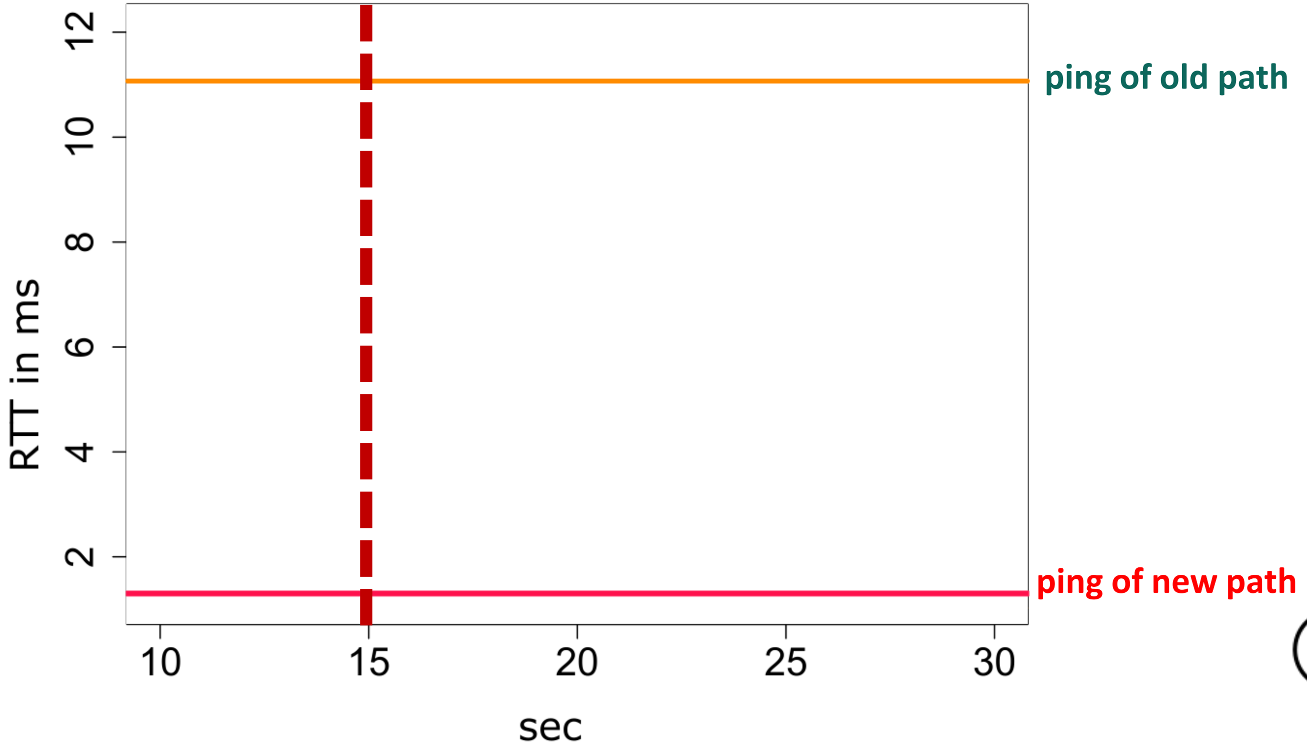
More open questions and specifics:
Survey of Consistent Software-Defined Network Updates
Klaus-Tycho Foerster, Stefan Schmid, Stefano Vissicchio
IEEE Communications Surveys & Tutorials, 21(2), 2019

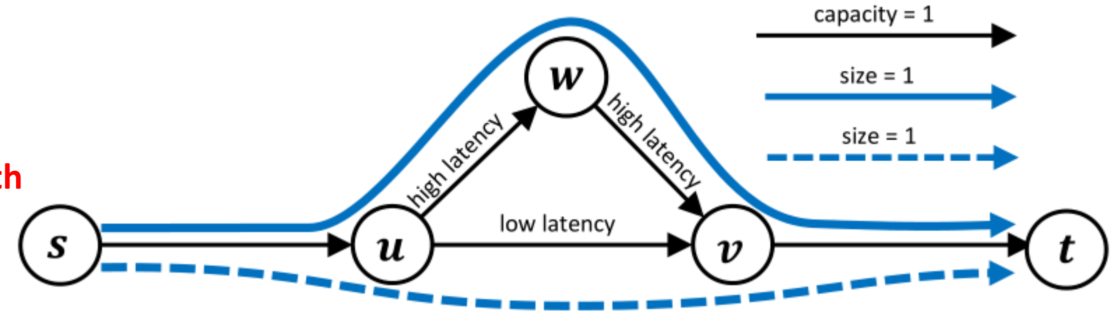
The Impact of Latency (in Testbed)



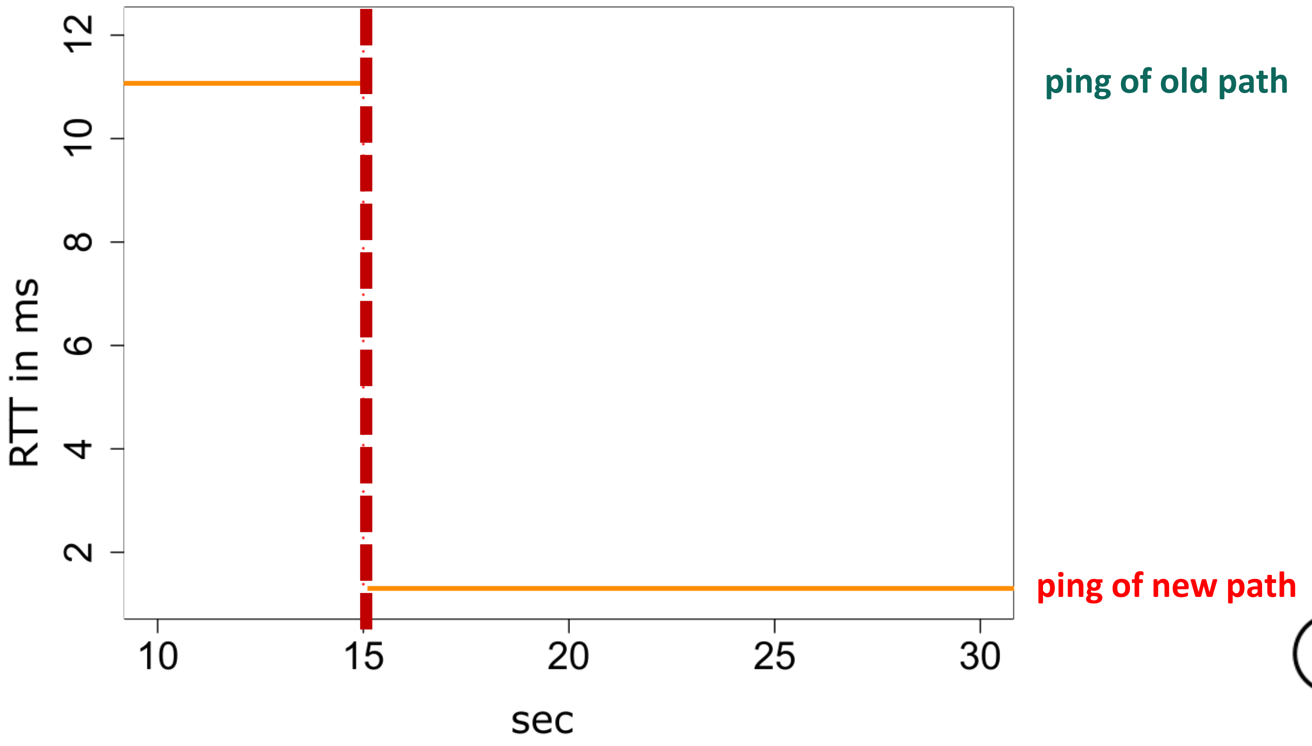


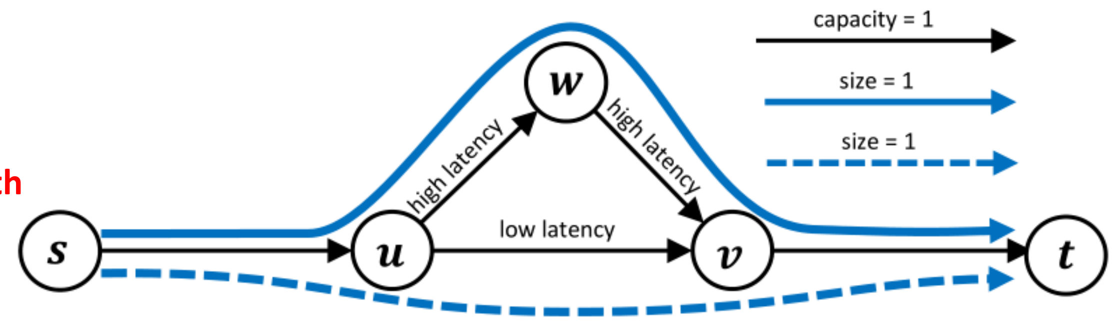
The Impact of Latency (in Testbed)



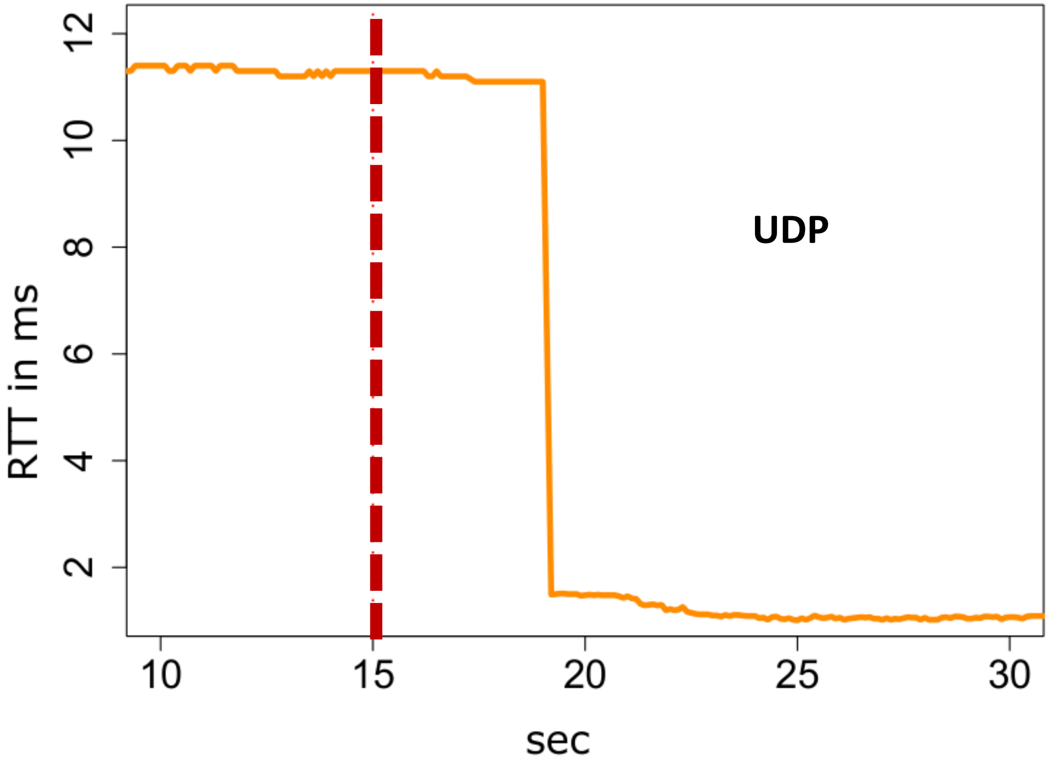


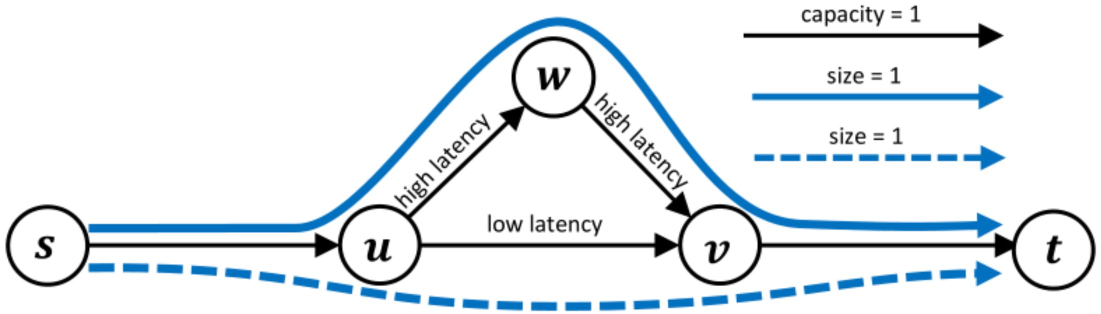
The Impact of Latency (in Testbed)



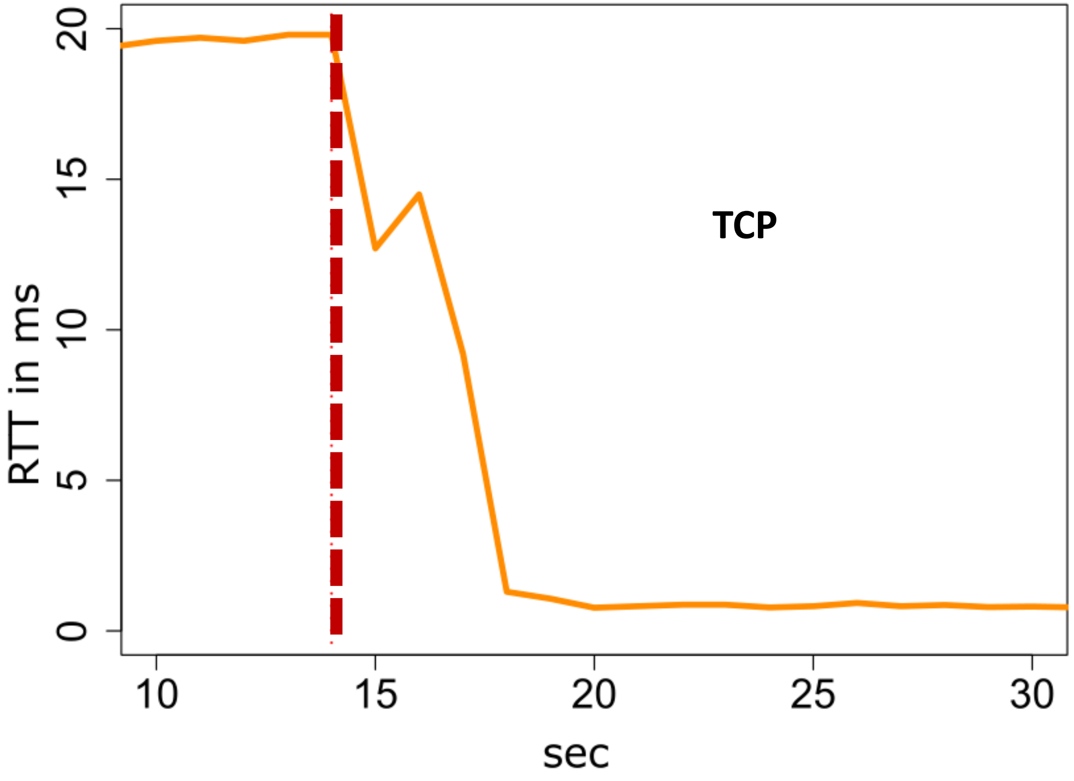


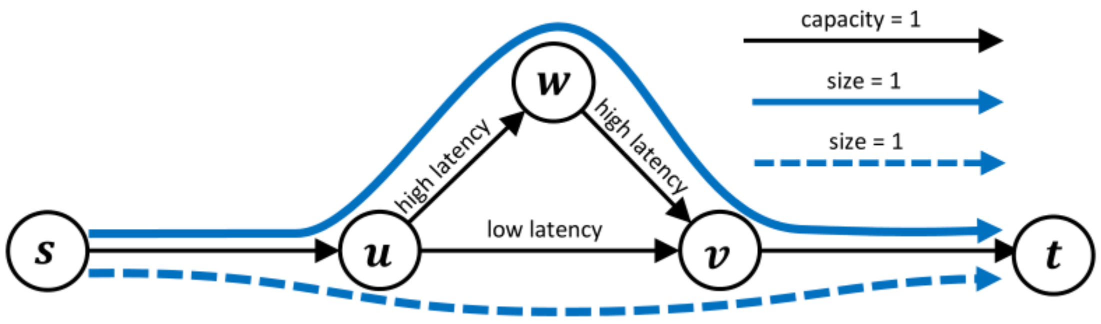
The Impact of Latency (in Testbed)





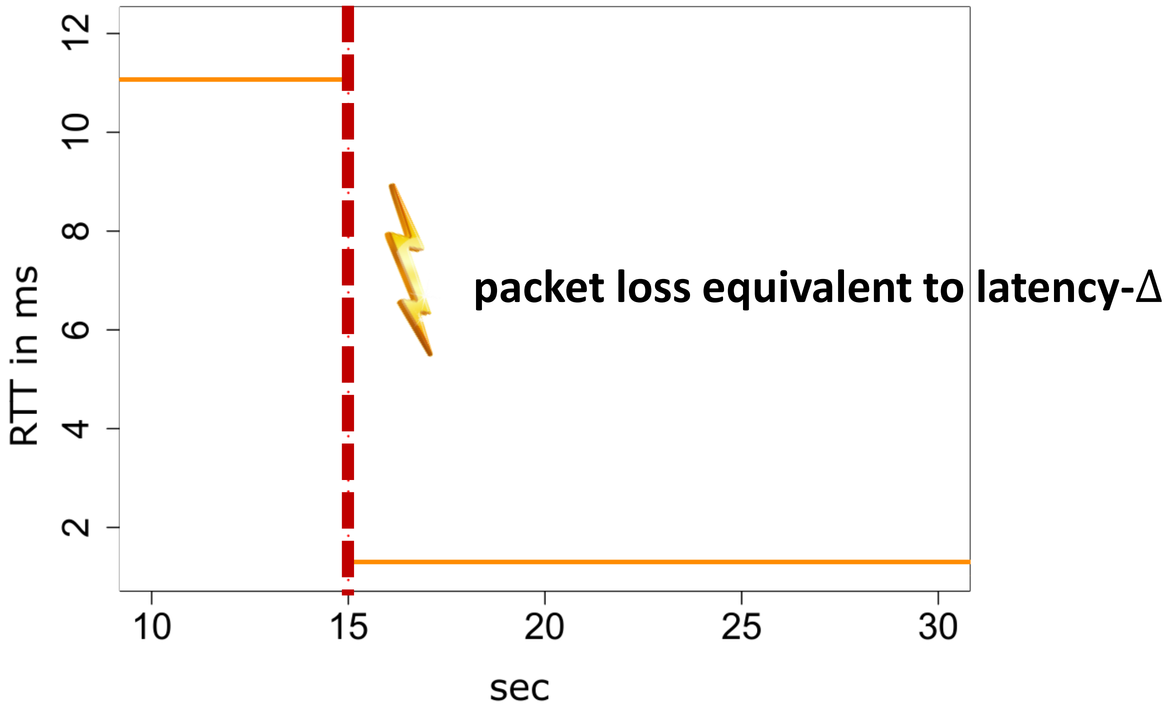
The Impact of Latency (in Testbed)

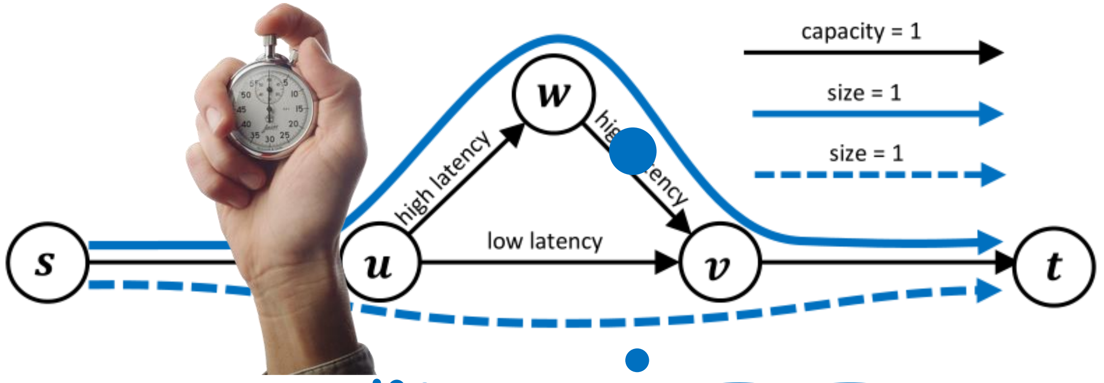




The Impact of Latency (in Testbed)

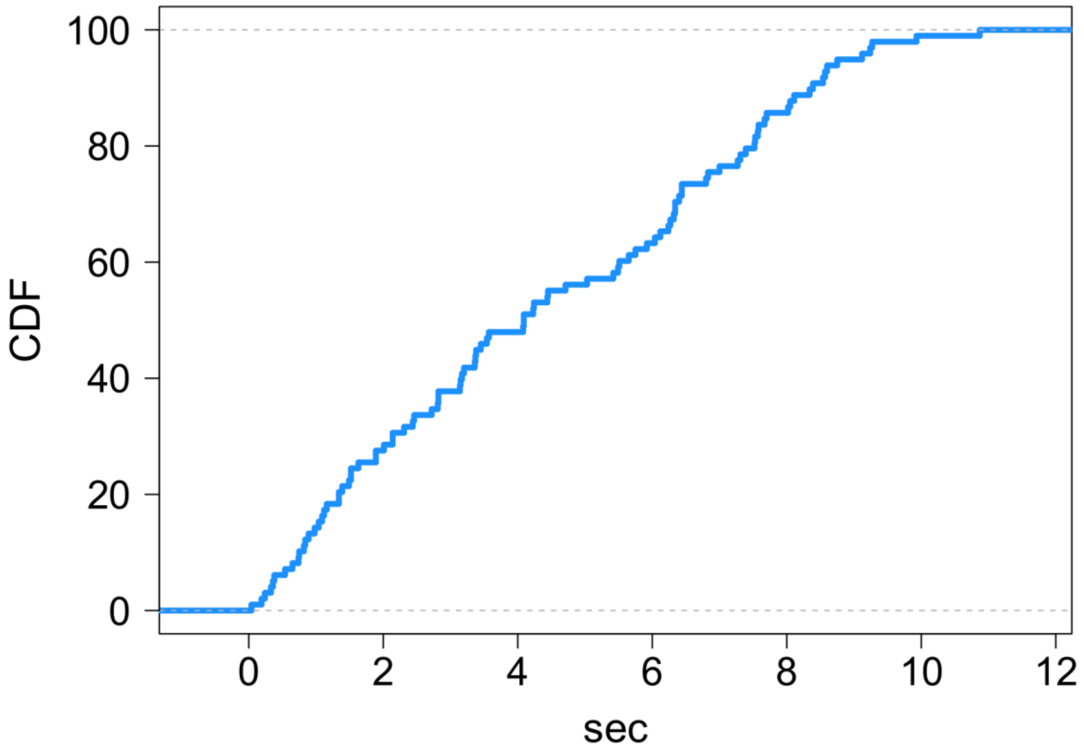
Because there is also work that focuses on better *time synchronization*, notably by *Mizrahi et al.*
https://sites.google.com/site/timedsdn/





Even holds without asynchrony

CDF of the Congestion Duration



Recap

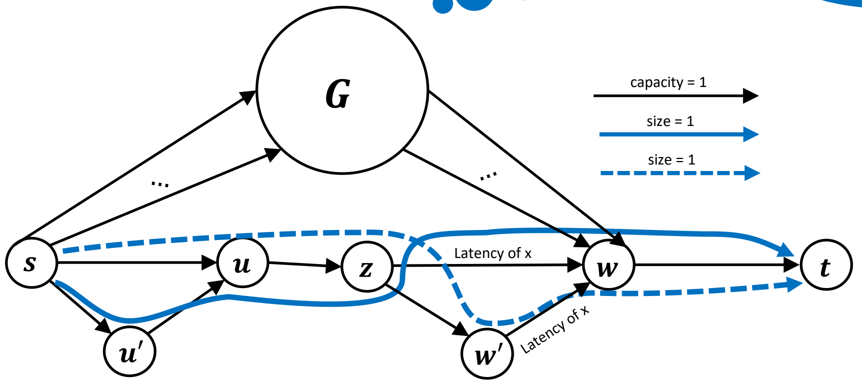
- Common (coarse-grained) model:
  - Sum for all flows: Max(**old flow rules** , **new flow rules**) does not violate capacity [SWAN, SIGCOMM’13]
  - Decidable in polynomial time [Brandt et al., INFOCOM’16]
    - For unsplittable flows: NP-hard already for 2 flows

- Does not capture congestion due to flows congesting themselves!
  - How hard?

How hard?

- Unit latencies and splittable flow of unit size:
  - Already NP-hard for a single flow!

Find a temporary path to offload parts of the flow



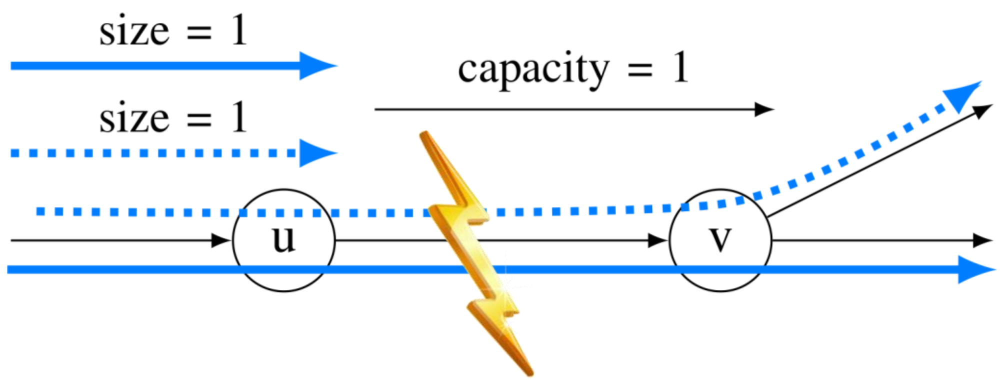
universität wien

Recap of the last few slides

- Common (coarse-grained) model:
  - Sum for all flows: Max(**old flow rules** , **new flow rules**) does not violate capacity [SWAN, SIGCOMM’13]
  - Decidable in polynomial time [Brandt et al., INFOCOM’16]
    - For unsplittable flows: NP-hard already for 2 flows
- Does not capture congestion due to flows congesting themselves!
  - How hard?
    - NP-hard for unit size/latency and splittable flows ☹
- How to fix?
  - Treat old and new flow rules as separate flows?

Old and New as Different Entities

- Idea: We can handle interplay between different flows
  - Handle old and new as different flows?
    - Prevents such congestion in popular approaches, eg SWAN, Dionysus, zUpdate etc.




Relax And Take it Easy!
MY BABY FLOAT
INTEX

Relax for Polynomial-Time Lossless Updates

- Idea: Relax the problem formulation
  - Be congestion-free for *any* set of latencies
    - (I.e., adversary may change latencies at any time)

- Now congestion-free intermediate steps become **reversible**

Achieved by spreading the network load

- Rough structure of the algorithm (for splittable flows):
  - Take old (new) state, reach intermediate state where critical set of edges have spare capacity
    - Not possible? No congestion-free migration possible.

Recap of the last few slides

How to extend beyond a single destination?

- Common (coarse-grained) model:
  - Sum for all flows: Max(**old flow rules** , **new flow rules**) does not violate capacity [SWAN, SIGCOMM'13]
  - Decidable in polynomial time [Brandt et al., INFOCOM'16]
    - For unsplittable flows: NP-hard already for 2 flows
- Does not capture congestion due to flows congesting themselves!
  - NP-hard for unit size/latency and splittable flows ☹

But requires non-fixed new flow paths

- By relaxing latency constraints:
  - Again polynomial-time decidable
- Interestingly: Augmenting flow idea still works **even without relaxing latency constraints!**

Open Problems and Outlook in General

- Various algorithmic and complexity questions for a centralized controller
  - See recent survey

- First connections to more classic distributed computing topics are made
  - *Proof-labeling*
    - Very basic right now, how to build more complex/efficient systems?

- Maybe the bigger question: How to properly distribute the centralized controller
  - Opportunity: The SUPPORTED model / *preprocessing*

universität
wien

Some References

- **Survey of Consistent Software-Defined Network Updates. Klaus-Tycho Foerster, Stefan Schmid, and Stefano Vissicchio. IEEE Communications Surveys and Tutorials (COMST), Volume 21, Issue 2, pp. 1435-1461, secondquarter 2019.**
- Brief Announcement: Does Preprocessing Help under Congestion? Klaus-Tycho Foerster, Janne Korhonen, Joel Rybicki, and Stefan Schmid. ACM Symposium on Principles of Distributed Computing (PODC), Toronto, Ontario, Canada, July 2019.
- On Polynomial-Time Congestion-Free Software-Defined Network Updates. Saeed Akhoondian Amiri, Szymon Dudycz, Mahmoud Parham, Stefan Schmid, and Sebastian Wiederrecht. IFIP Networking, Warsaw, Poland, May 2019.
- Latency and Consistent Flow Migration: Relax for Lossless Updates. Klaus-Tycho Foerster, Laurent Vanbever, and Roger Wattenhofer. 18th IFIP Networking Conference (IFIP Networking), Warsaw, Poland, May 2019.
- On the Power of Preprocessing in Decentralized Network Optimization. Klaus-Tycho Foerster, Juho Hirvonen, Stefan Schmid, and Jukka Suomela. 39th IEEE International Conference on Computer Communications (INFOCOM), Paris, France, April 2019.
- RADWAN: Rate Adaptive Wide Area Network. Rachee Singh, Manya Ghobadi, Klaus-Tycho Foerster, Mark Filer, and Phillipa Gill. Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM), Budapest, Hungary, August 2018.
- Congestion-Free Rerouting of Flows on DAGs. Saeed Akhoondian Amiri, Szymon Dudycz, Stefan Schmid, and Sebastian Wiederrecht. 45th International Colloquium on Automata, Languages, and Programming (ICALP), Prague, Czech Republic, July 2018.
- Loop-Free Route Updates for Software-Defined Networks. Klaus-Tycho Foerster, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. IEEE/ACM Transactions on Networking (ToN), Volume 26, Issue 1, pp. 328-341, February 2018.
- Efficient Loop-Free Rerouting of Multiple SDN Flows. Arsany Basta, Andreas Blenk, Szymon Dudycz, Arne Ludwig, and Stefan Schmid. IEEE/ACM Transactions on Networking (ToN), 2018.
- Local Checkability, No Strings Attached: (A)cyclicity, Reachability, Loop Free Updates in SDNs. Klaus-Tycho Foerster, Thomas Luedi, Jochen Seidel, and Roger Wattenhofer. Theoretical Computer Science (TCS), Volume 709, pp. 48-63, January 2018.
- On the Consistent Migration of Unsplittable Flows: Upper and Lower Complexity Bounds. Klaus-Tycho Foerster. 16th IEEE International Symposium on Network Computing and Applications (NCA), Cambridge, MA, USA, November 2017.
- Augmenting Flows for the Consistent Migration of Multi-Commodity Single-Destination Flows in SDNs. Sebastian Brandt, Klaus-Tycho Foerster, and Roger Wattenhofer. Pervasive and Mobile Computing (PMC), Volume 36, pp. 134-150, April 2017.
- Optimal Consistent Network Updates in Polynomial Time. Pavol Cerný, Nate Foster, Nilesh Jagnik, Jedidiah McClurg. DISC 2016
- The Power of Two in Consistent Network Updates: Hard Loop Freedom, Easy Flow Migration. Klaus-Tycho Foerster and Roger Wattenhofer. 25th International Conference on Computer Communication and Networks (ICCCN), Waikoloa, Hi, USA, August 2016.
- Transiently Consistent SDN Updates: Being Greedy is Hard. Saeed Akhoondian Amiri, Arne Ludwig, Jan Marcinkowski, and Stefan Schmid. 23rd International Colloquium on Structural Information and Communication Complexity (SIROCCO), Helsinki, Finland, July 2016.
- Consistent Updates in Software Defined Networks: On Dependencies, Loop Freedom, and Blackholes. Klaus-Tycho Foerster, Ratul Mahajan, and Roger Wattenhofer. 15th IFIP Networking Conference (IFIP Networking), Vienna, Austria, May 2016.
- On Consistent Migration of Flows in SDNs. Sebastian Brandt, Klaus-Tycho Foerster, and Roger Wattenhofer. 36th IEEE International Conference on Computer Communications (INFOCOM), San Francisco, California, USA, April 2016.
- Exploiting Locality in Distributed SDN Control. Stefan Schmid and Jukka Suomela. ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN), Hong Kong, China, August 2013.
- Achieving High Utilization with Software-Driven WAN. Chi-Yao Hong, Srikanth Kandula, Ratul Mahajan, Ming Zhang, Vijay Gill, Mohan Nanduri and Roger Wattenhofer. Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM) 2013.
- Abstractions for network update. Mark Reitblatt, Nate Foster, Jennifer Rexford, Cole Schlesinger, David Walker. Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM) 2012.
- Fast Distributed Approximations in Planar Graphs : Andrzej Czygrinow, Michal Hanckowiak, Wojciech Wawrzyniak:.. DISC 2008: 78-92
- Multi-Commodity Network Flows. T. C. Hu. Operations Research 11(3):344-360, 1963.

Not all, if some are missing, should be listed on slides directly

universität wien

Central Control over Distributed Asynchronous Systems: A Tutorial on Software-Defined Networks and Consistent Network Updates

Klaus-T. Foerster