The algorithmic challenges of local fast re-routing

Stefan Schmid (TU Berlin)

Kudos: Marco Chiesa

Communication Networks

Critical infrastructure of digital society

- Popularity of **datacentric applications**: health, business, entertainment, social networking, AI/ML, etc.
- Evident during ongoing **pandemic**: online learning, online conferences, etc.
- Much traffic especially to, from, and inside **datacenters**

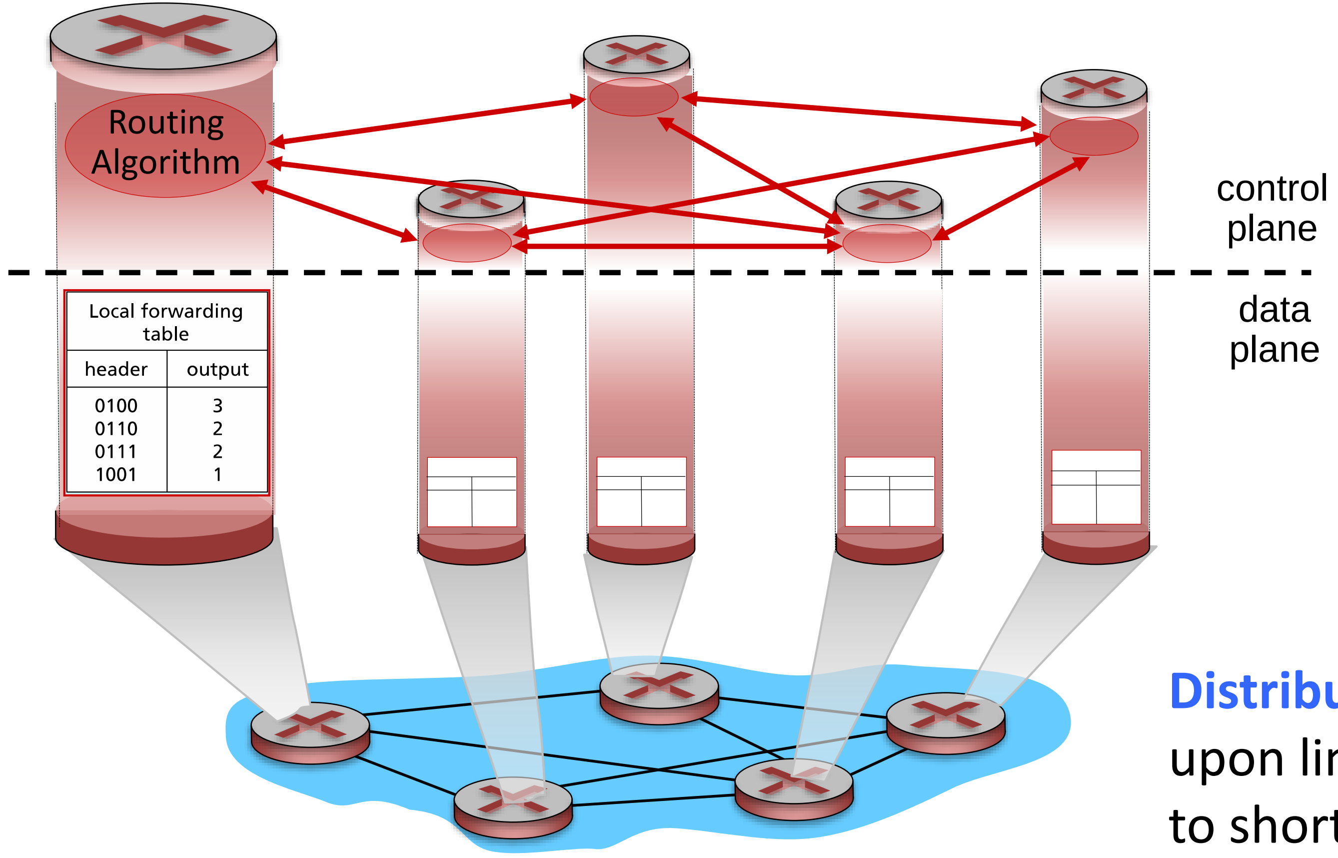
Facebook datacenter

Increasingly stringent dependability requirements!

Roadmap

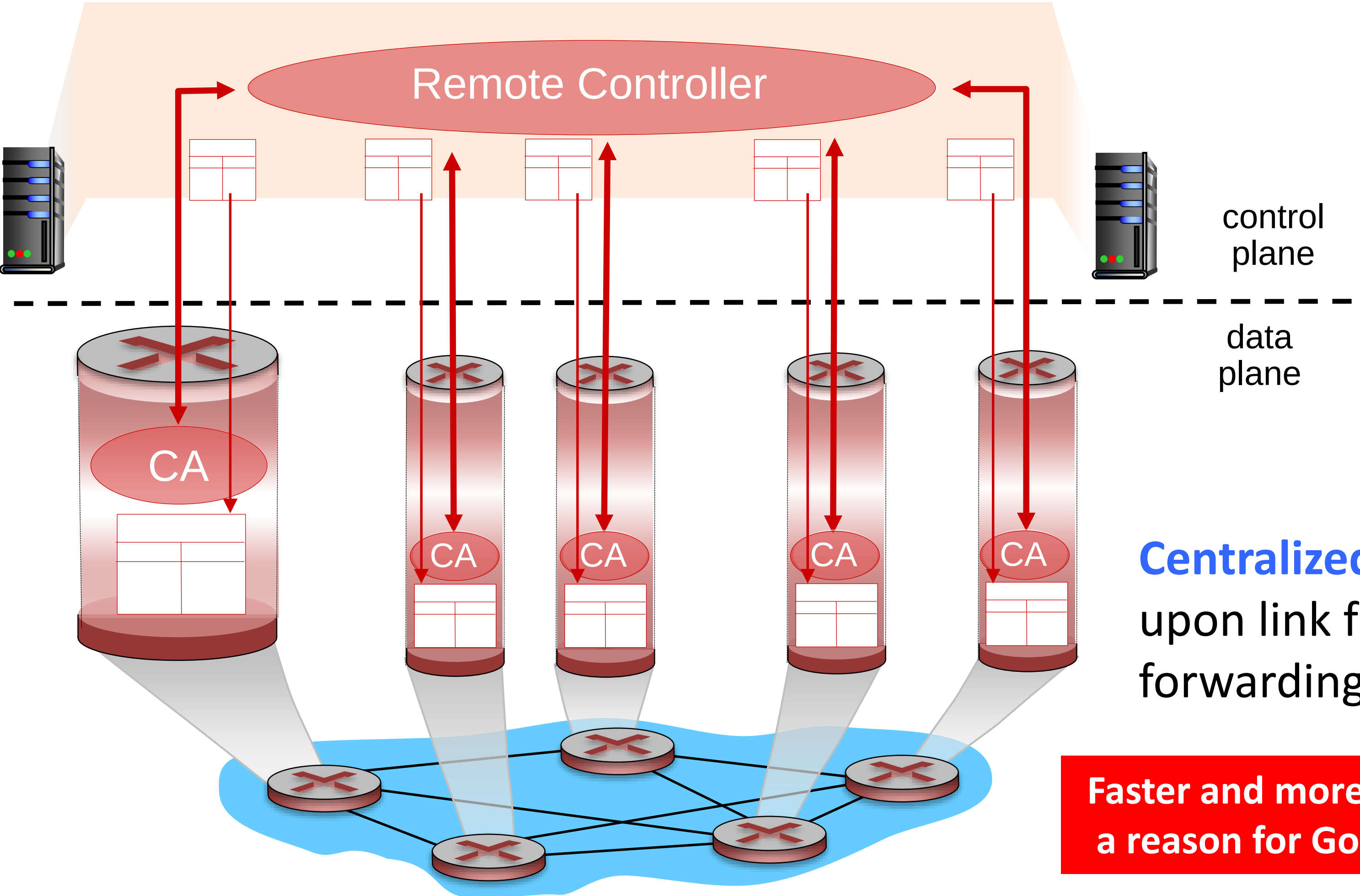
- **A Brief Background on Resilient Networking**
- Algorithms for Local Fast Re-Routing (FRR)

Traditional Networks



Distributed algorithms: upon link failure, ***reconverge*** to shortest paths

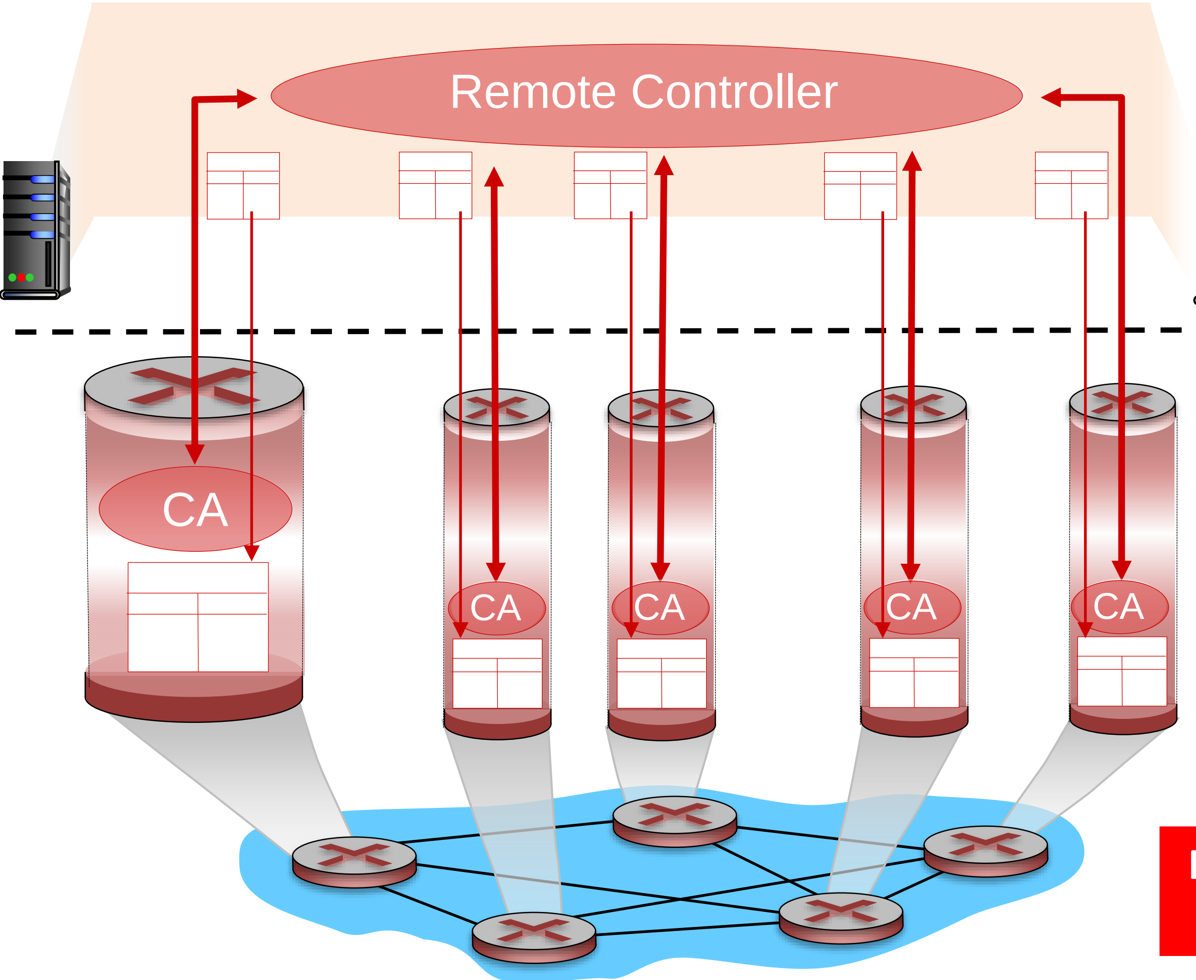
Software-Defined Networks (SDN)



Centralized algorithms: upon link failure, ***push*** new forwarding rules

Faster and more controlled reaction: a reason for Google's move to SDN!

Software-Defined Networks (SDN)



Centralized algorithms: upon link failure, ***push*** new forwarding rules

Faster and more controlled reaction: a reason for Google's move to SDN!

Restoration in control plane takes time -> packet drops!

Video shot taken from “Lemmings”
designed and developed by DMA Design

Failover: Control Plane vs Data Plane

- Slower reaction in the **control plane** than in the **data plane**

vs

Minister of Education

Teacher in the Classroom

Approaches for Failover

In Control Plane

- Distributed recomputation of shortest paths ("**re-convergence**")
- Centralized recomputation of paths (SDN)
- **Link-reversal** algorithms (e.g., Gafni et al.)

vs

In Data Plane

- Static forwarding table
- Rules pre-installed ***before*** failures are known

Approaches for Failover

Slow but "globally informed".

In C[...]rol Plane

- Distributed recomputation of shortest paths ("**re-convergence**")
- Centralized recomputation of paths (SDN)
- **Link-reversal** algorithms (e.g., Gafni et al.)

vs

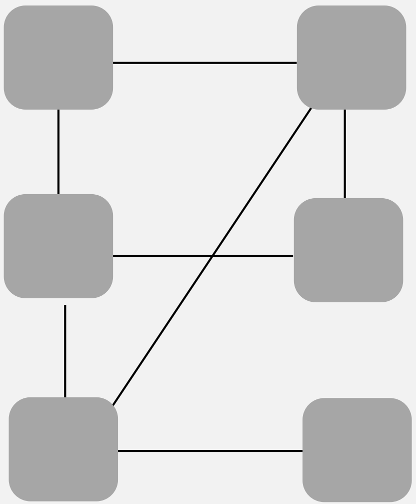
Fast but "local knowledge".

In D[...] Plane

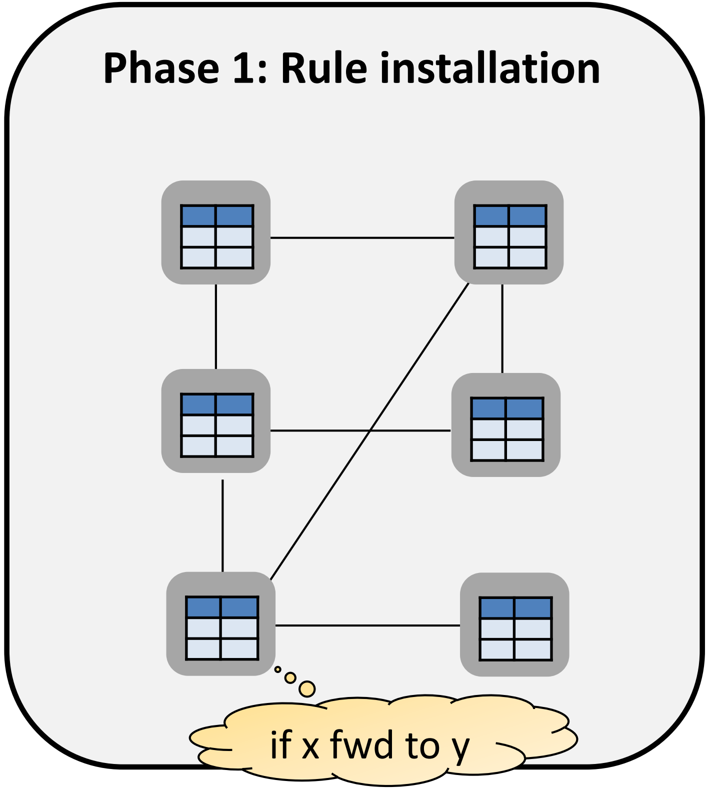
- Static forwarding table
- Rules pre-installed ***before*** failures are known

The FRR Problem

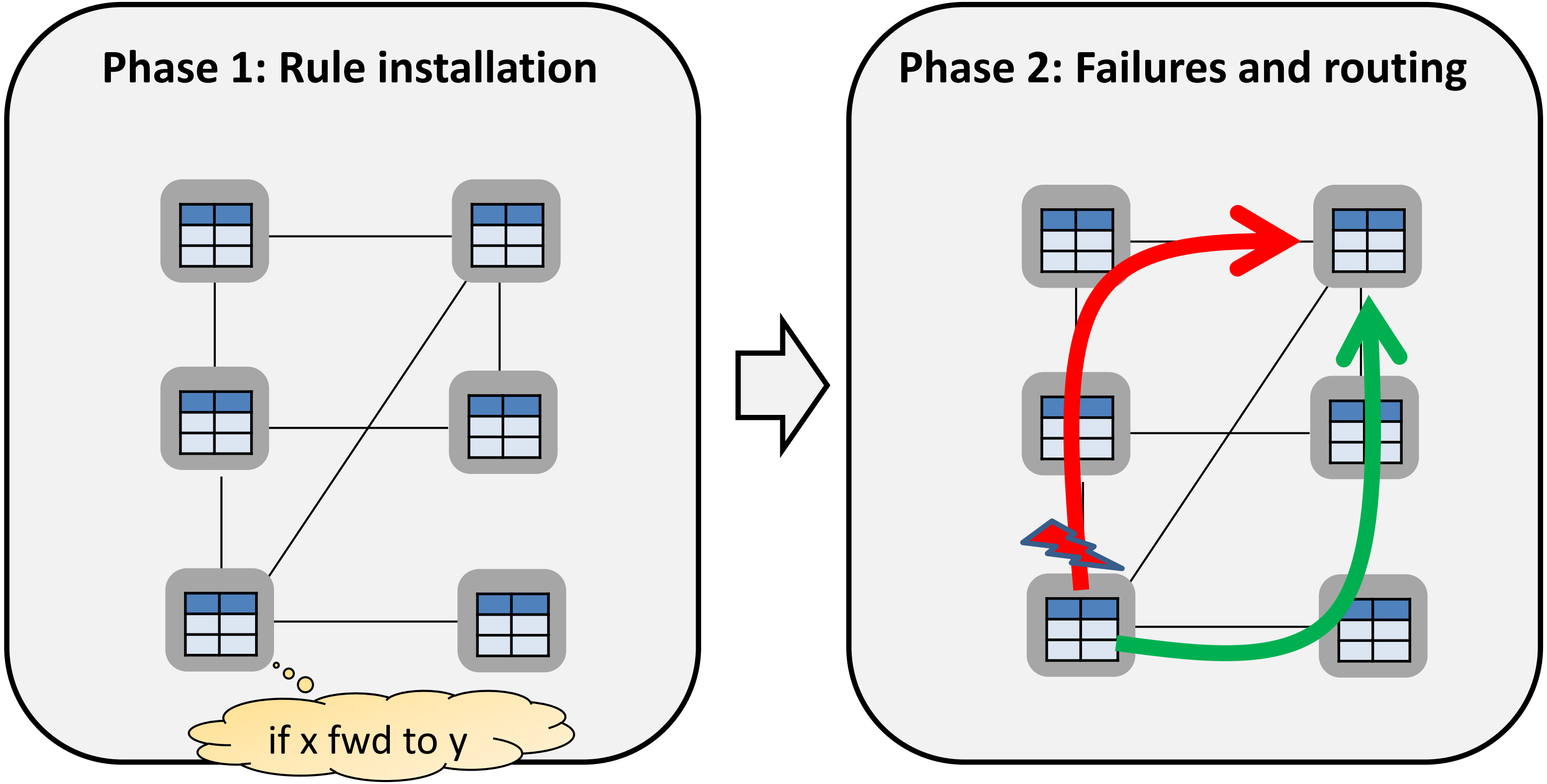
Phase 1: Rule installation



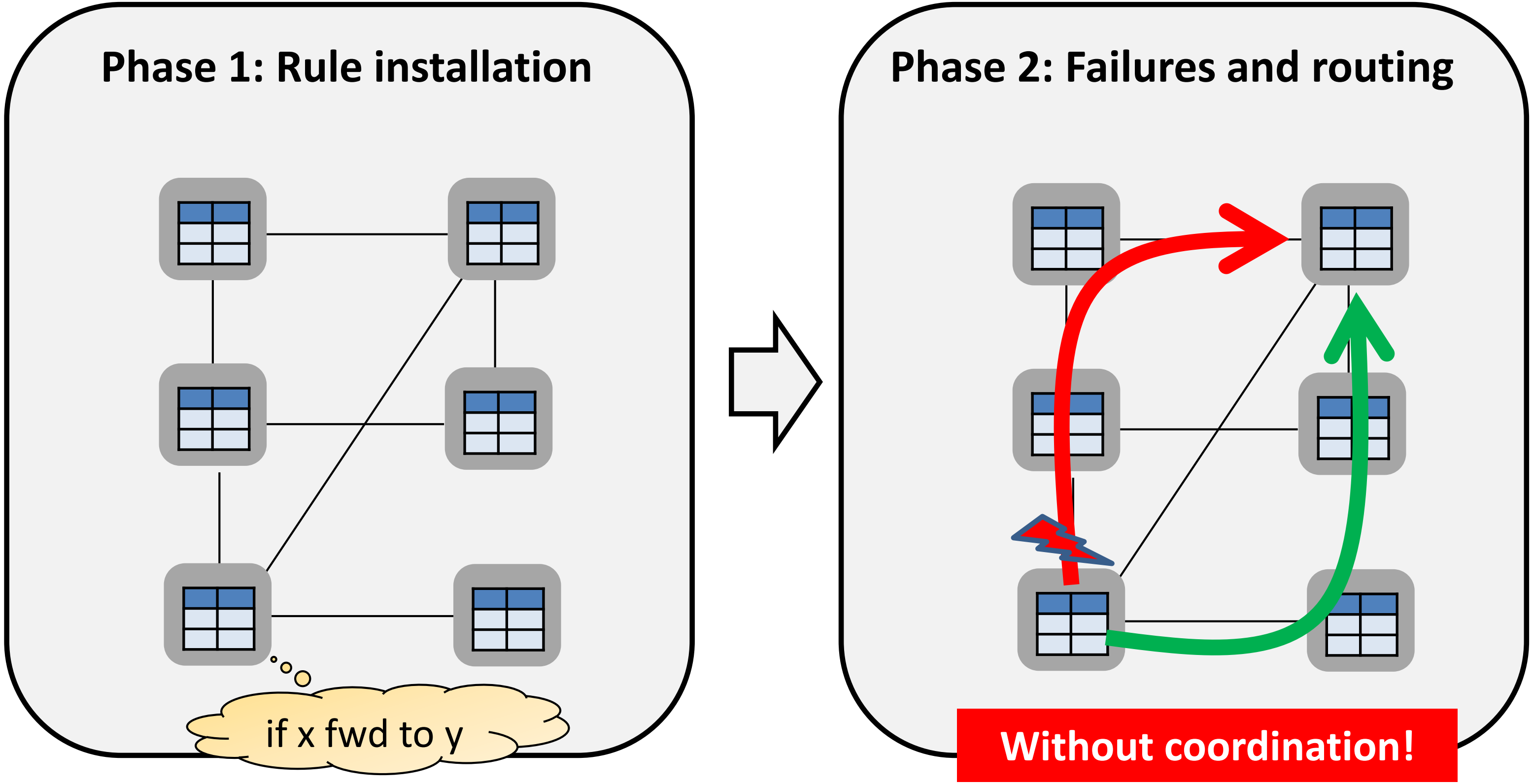
The FRR Problem



The FRR Problem



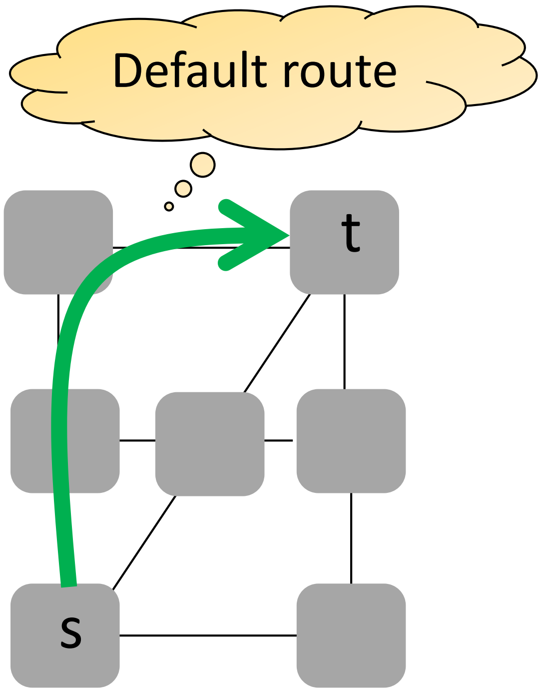
The FRR Problem



The FRR Problem

- **Pre-installed** local-fast failover rules
  - *Can depend on local failures and, e.g., destination, inport, source*
- **At runtime**, rules are just *"executed"*

Advantage: no need to wait for reconvergence.



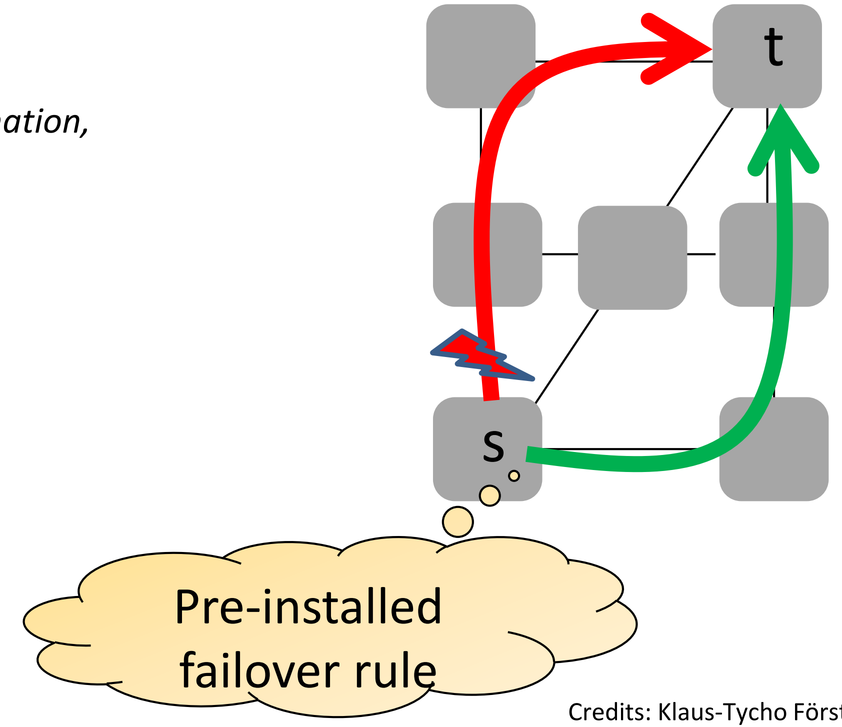
Credits: Klaus-Tycho Förster

The FRR Problem

- **Pre-installed** local-fast failover rules
  - *Can depend on local failures and, e.g., destination, inport, source*
- **At runtime**, rules are just *"executed"*

Advantage: no need to wait for reconvergence.

Good alternative under 1 failure!



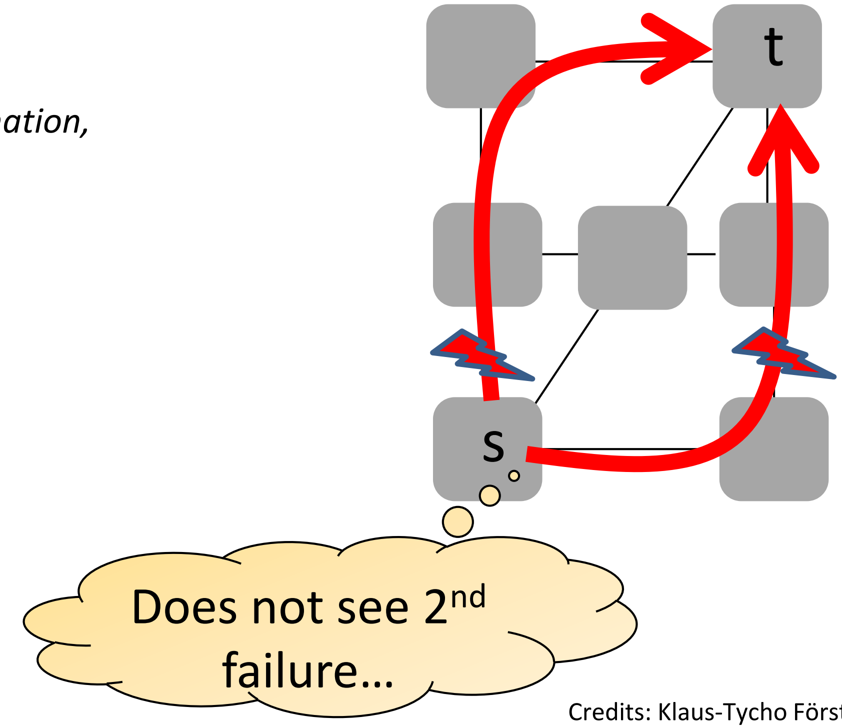
Credits: Klaus-Tycho Förster

The FRR Problem

- **Pre-installed** local-fast failover rules
  - *Can depend on local failures and, e.g., destination, inport, source*
- **At runtime**, rules are just *"executed"*

Advantage: no need to wait for reconvergence.

Good alternative under 1 failure!



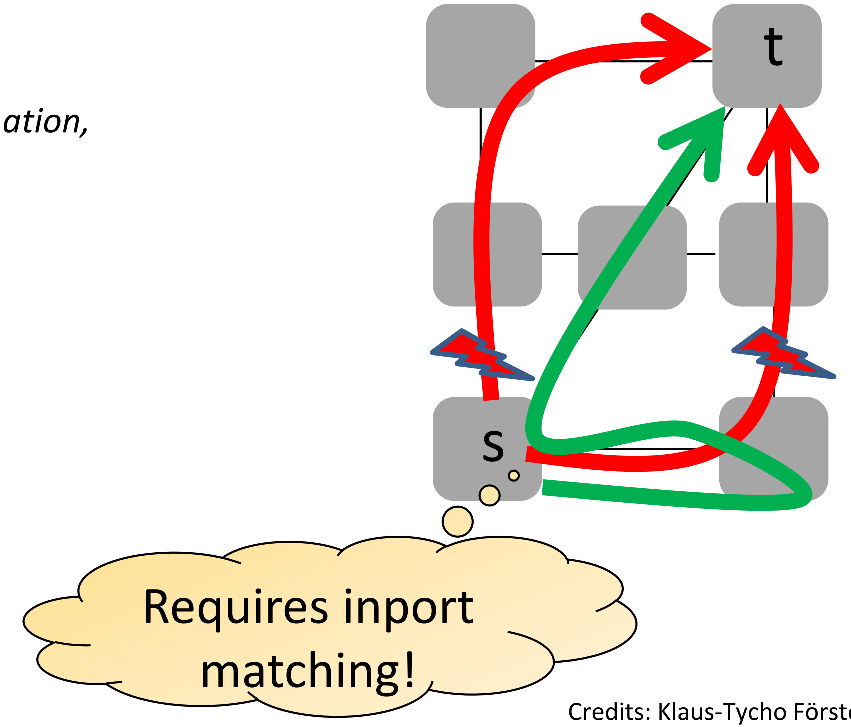
Credits: Klaus-Tycho Förster

The FRR Problem

- **Pre-installed** local-fast failover rules
  - *Can depend on local failures and, e.g., destination, inport, source*
- **At runtime**, rules are just *"executed"*

Advantage: no need to wait for reconvergence.

Can get complex under multiple failures..



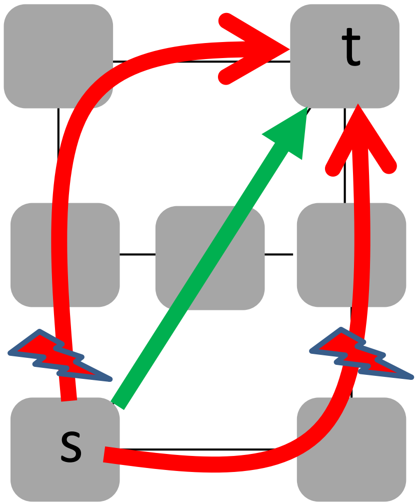
Credits: Klaus-Tycho Förster

The FRR Problem

- **Pre-installed** local-fast failover rules
  - *Can depend on local failures and, e.g., destination, inport, source*
- **At runtime**, rules are just *”executed”*

Advantage: no need to wait for reconvergence.

With global knowledge: simpler!

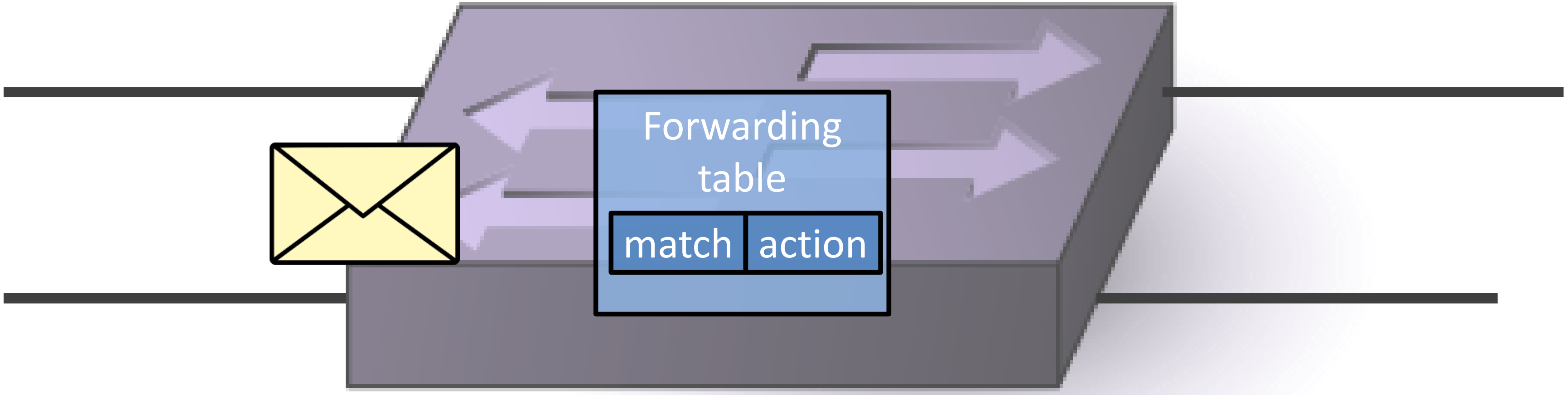


Credits: Klaus-Tycho Förster

What information is locally available in a switch for handling a packet?

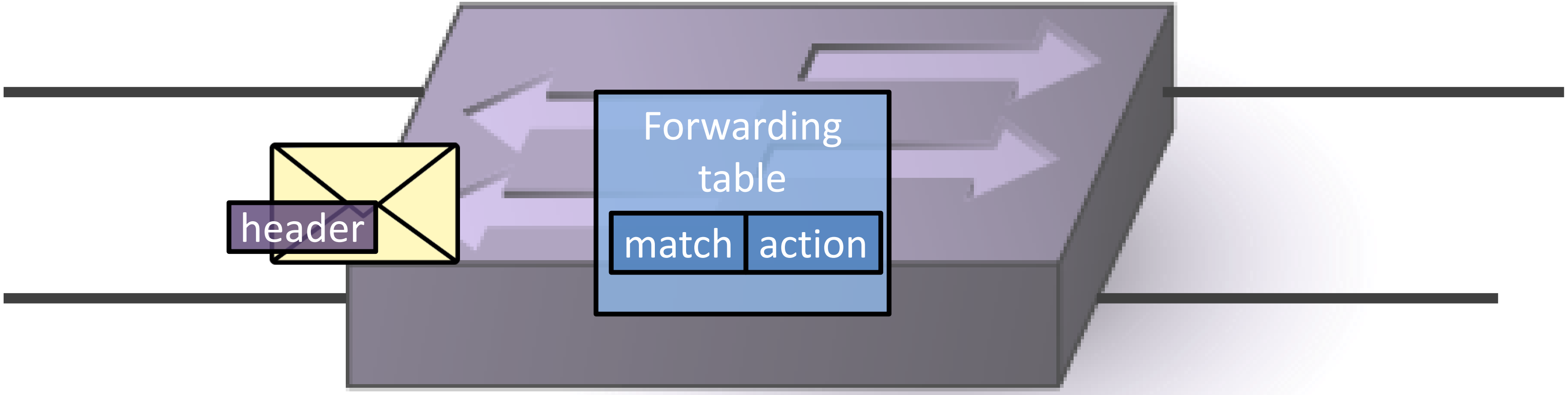
Credits: Marco Chiesa

Locally Available Information: The Forwarding Table: Match -> Action



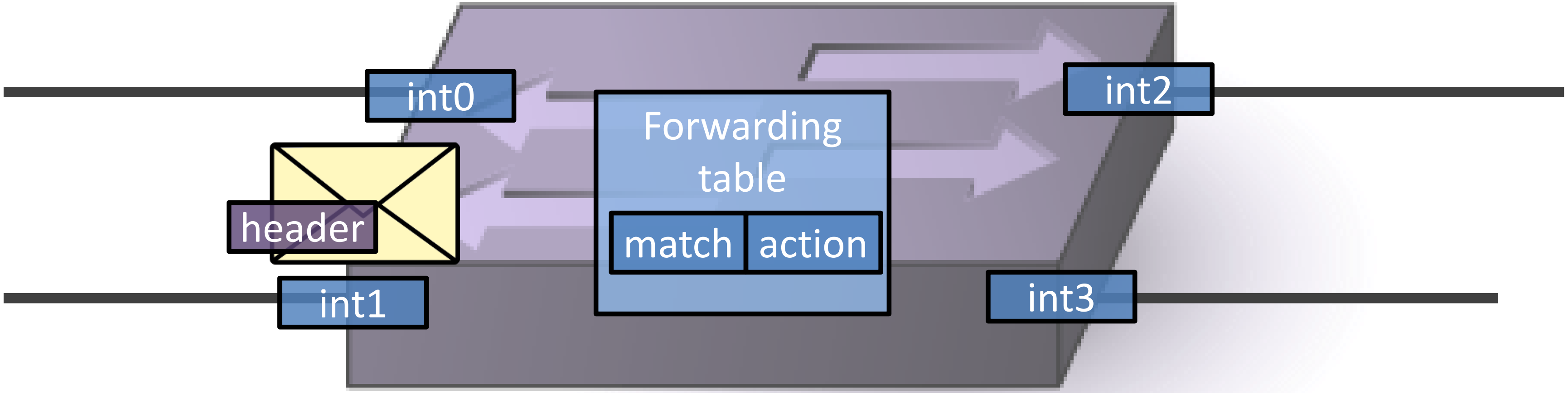
Credits: Marco Chiesa

Locally Available Information:
The Packet Header



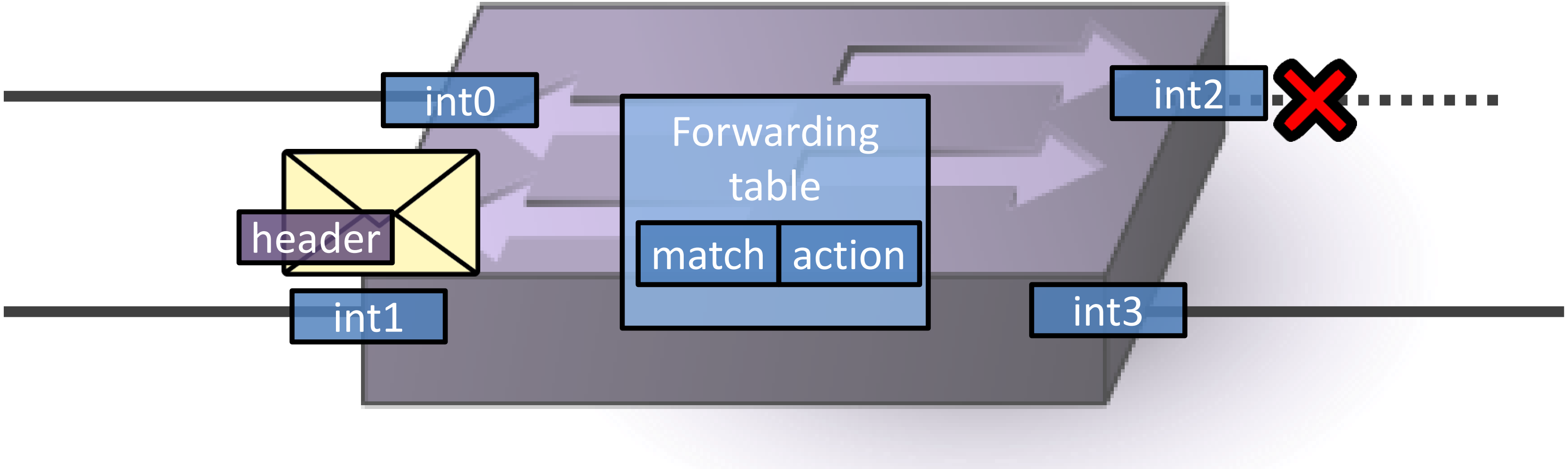
Credits: Marco Chiesa

Locally Available Information: The Inport of the Received Packet



Credits: Marco Chiesa

Locally Available Information: The Outgoing Port Depends on Failed Links



Credits: Marco Chiesa

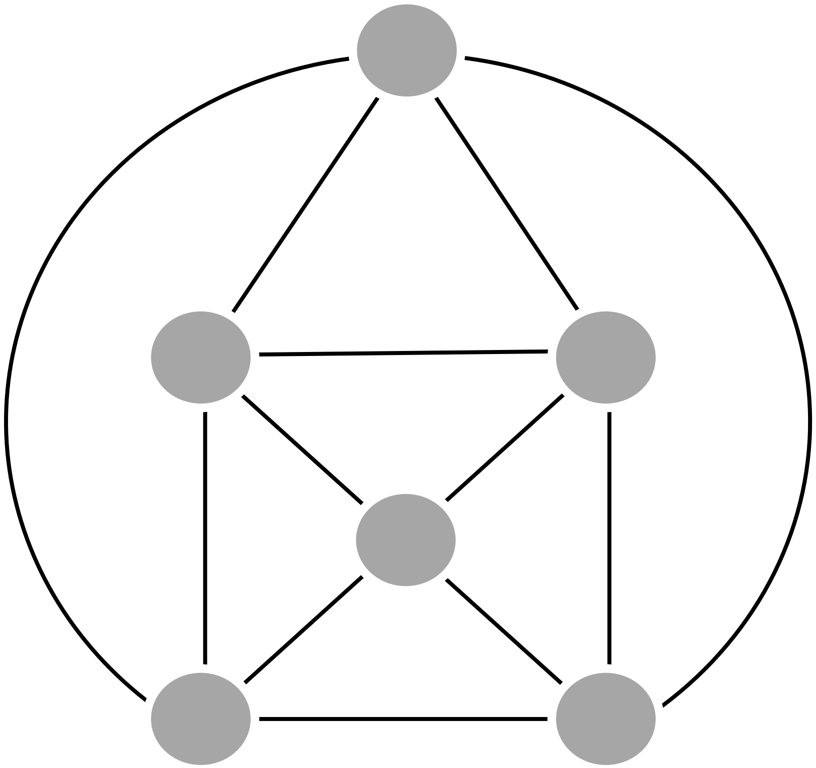
Raises an Interesting Question

Can we pre-install local fast failover rules which ensure reachability under multiple failures? *In particular: How many failures* can be tolerated by static forwarding tables?

Roadmap

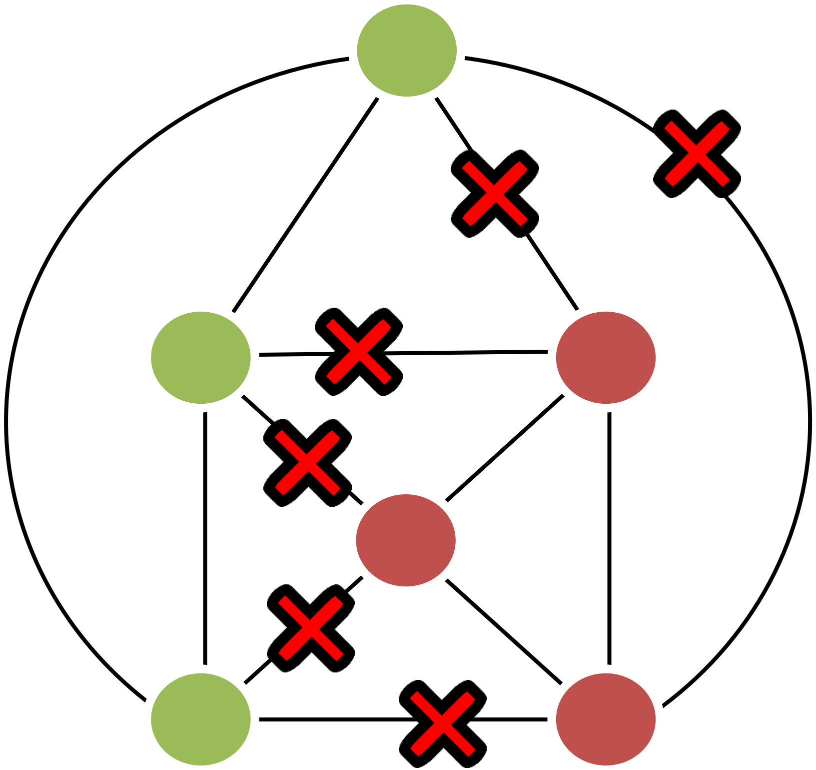
- A Brief Background on Resilient Networking
- **Algorithms for Local Fast Re-Routing (FRR)**

So: How many failures can be tolerated by static forwarding tables?



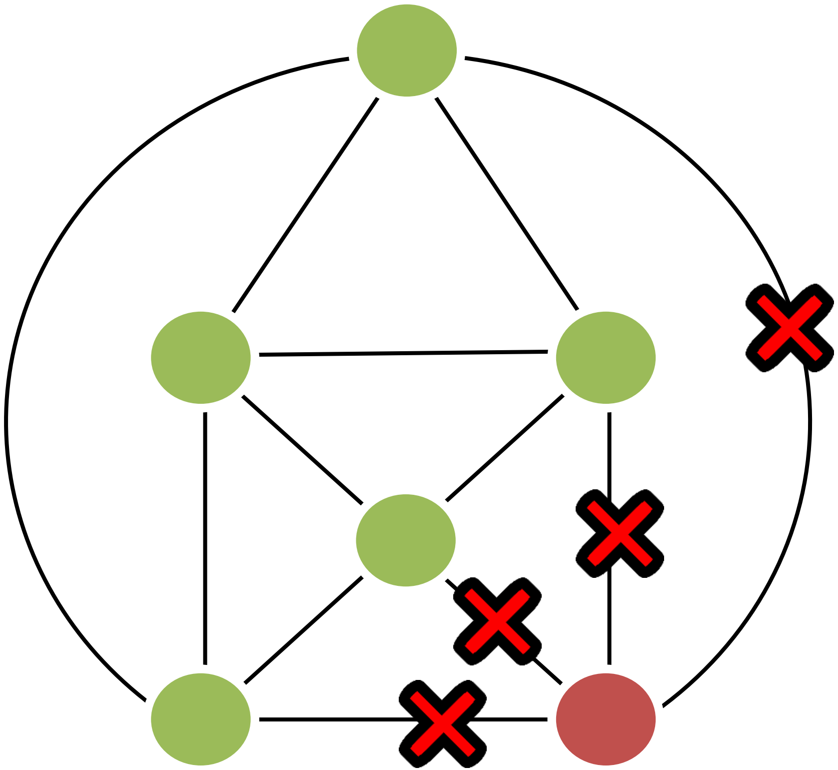
Credits: Marco Chiesa

If we partition the network, there is not much to do



Credits: Marco Chiesa

The connectivity k of a network N: the minimum number of link deletions that partitions N



The connectivity of this network is *four*

Credits: Marco Chiesa

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

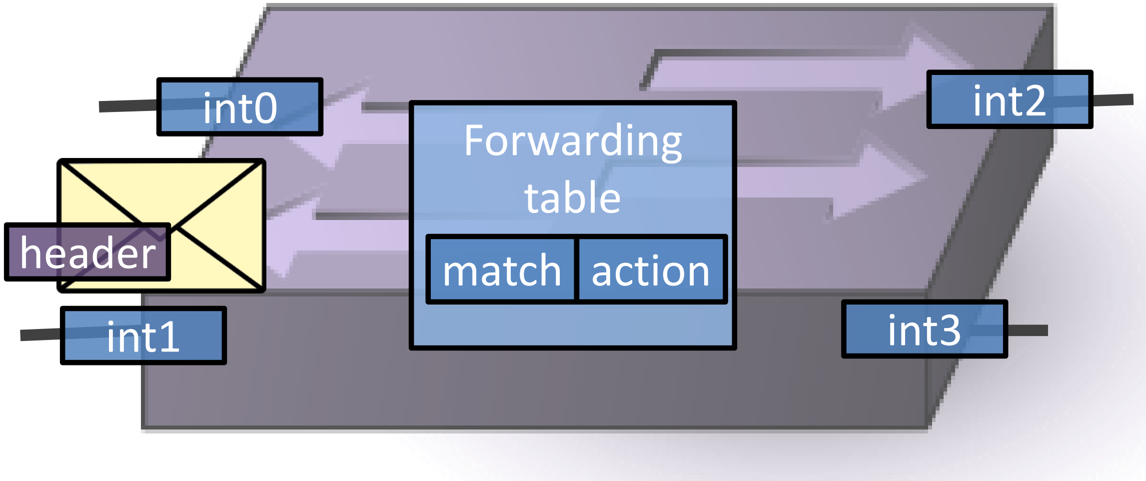
Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

Spectrum of Models

Recall our switch model:



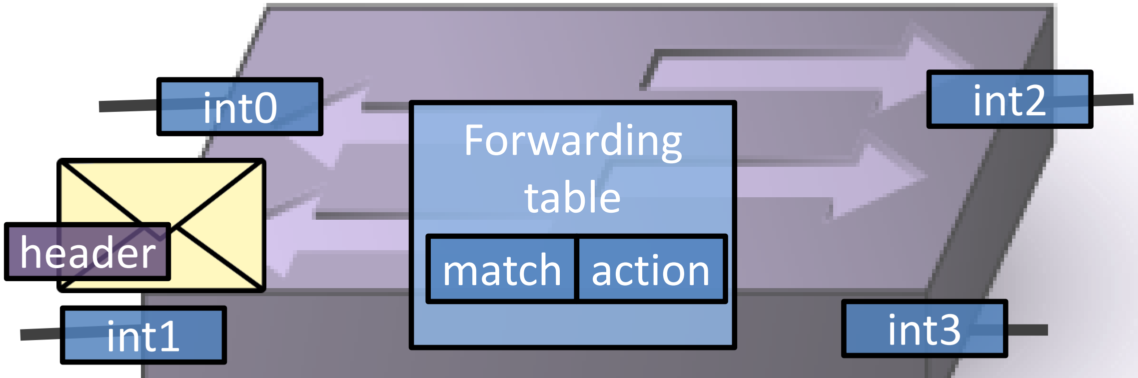
Achievable resilience depnds on *what can be matched*:

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting

Credits: Marco Chiesa

Spectrum of Models

Recall our switch model:



Can carry global information, but often undesirable

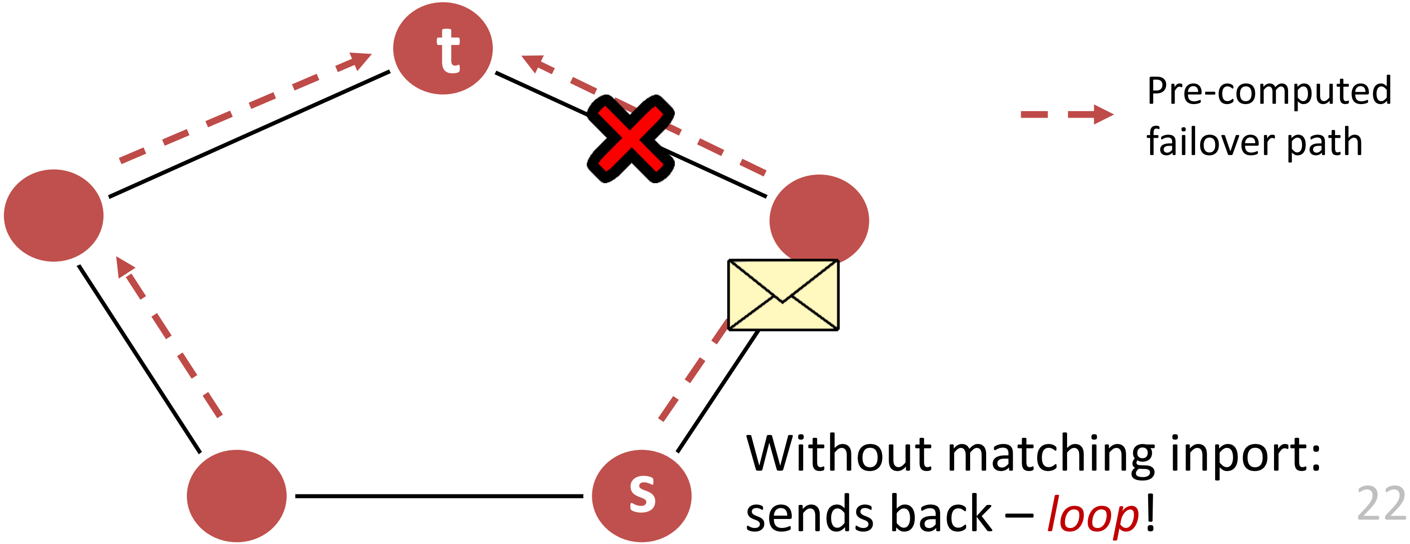
Achievable resilience depnds on

Per-destination	**Per source**	**Incoming port**	**Probabilistic forwarding**	**Packet header rewriting**

Credits: Marco Chiesa

Per-destination routing *cannot cope* with *even one* link failure

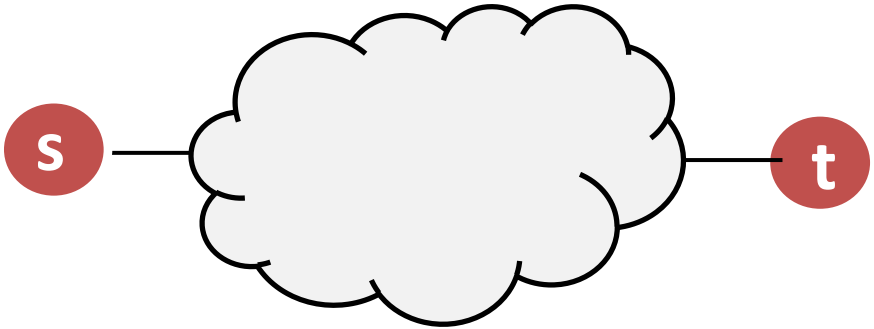
Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X					0



Without matching inport: sends back – *loop*!

Can we achieve k – 1 resiliency in k-connected graph here?

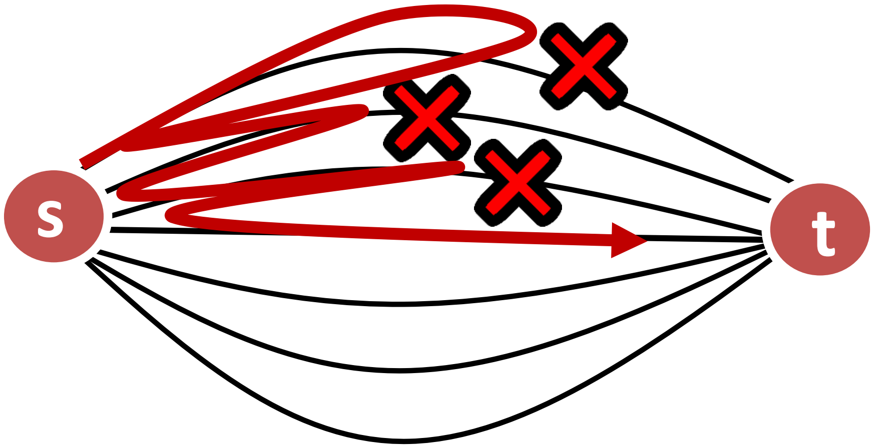
Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X	X	X			?



Credits: Marco Chiesa

Can we achieve k – 1 resiliency in k-connected graph here?

Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X	X	X			Yes



k disjoint paths: try one after the other, routing *back to source* each time.

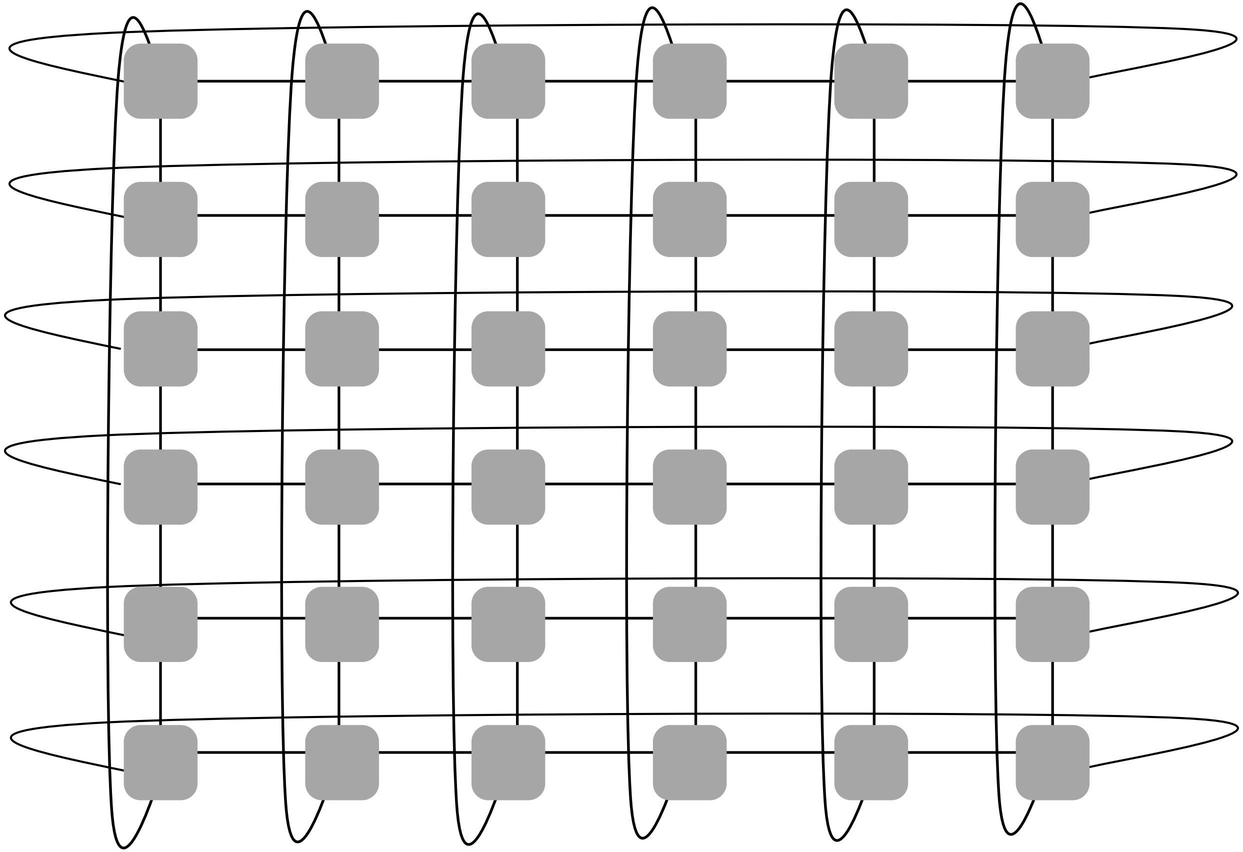
Credits: Marco Chiesa

Can we achieve k – 1 resiliency in k-connected graph here?

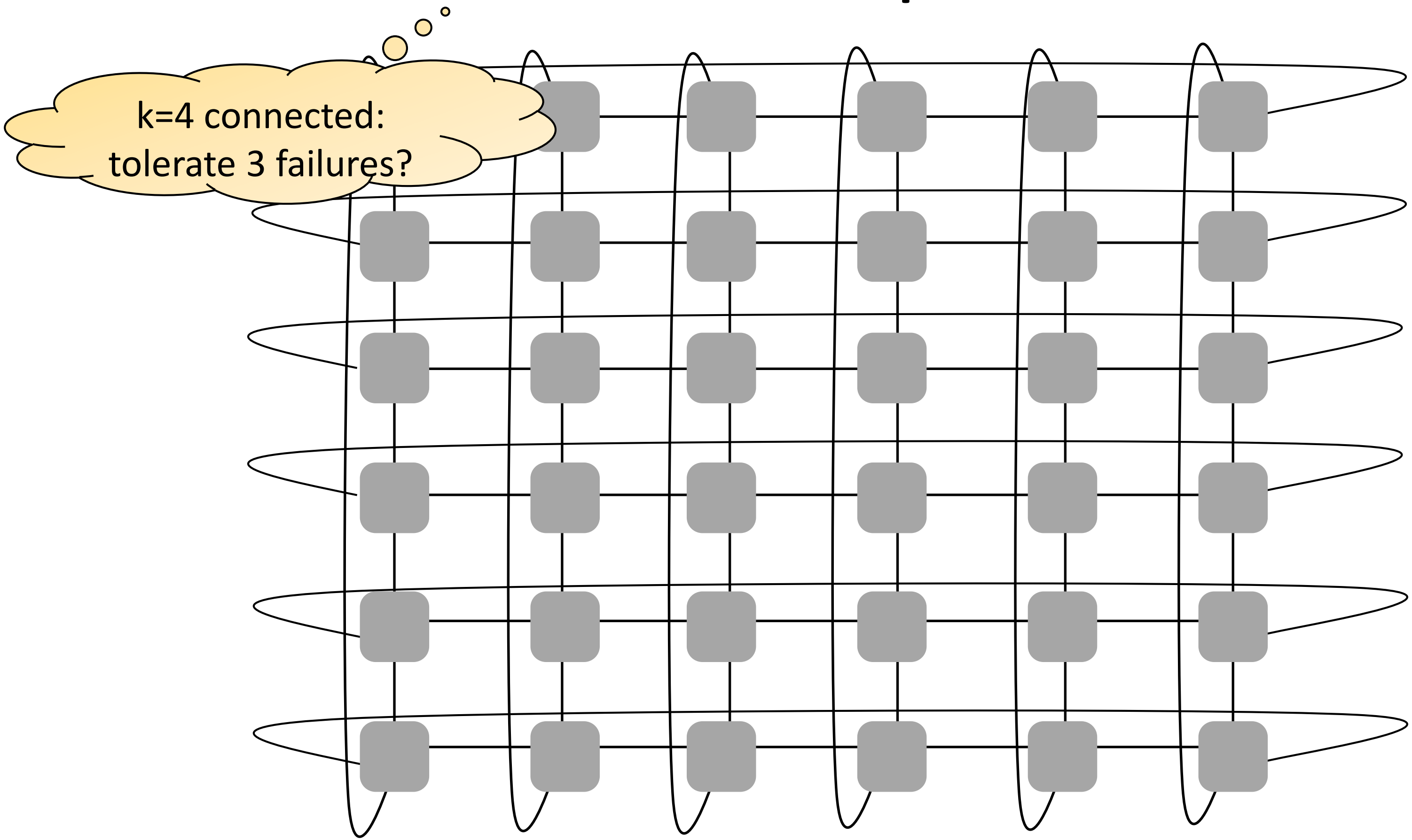
Per-destination	Per source	Incoming port	Probabilistic forwarding	Packet header rewriting	Resiliency
X		X			?

What about this scenario? Practically important. From now on called "ideal resilience".

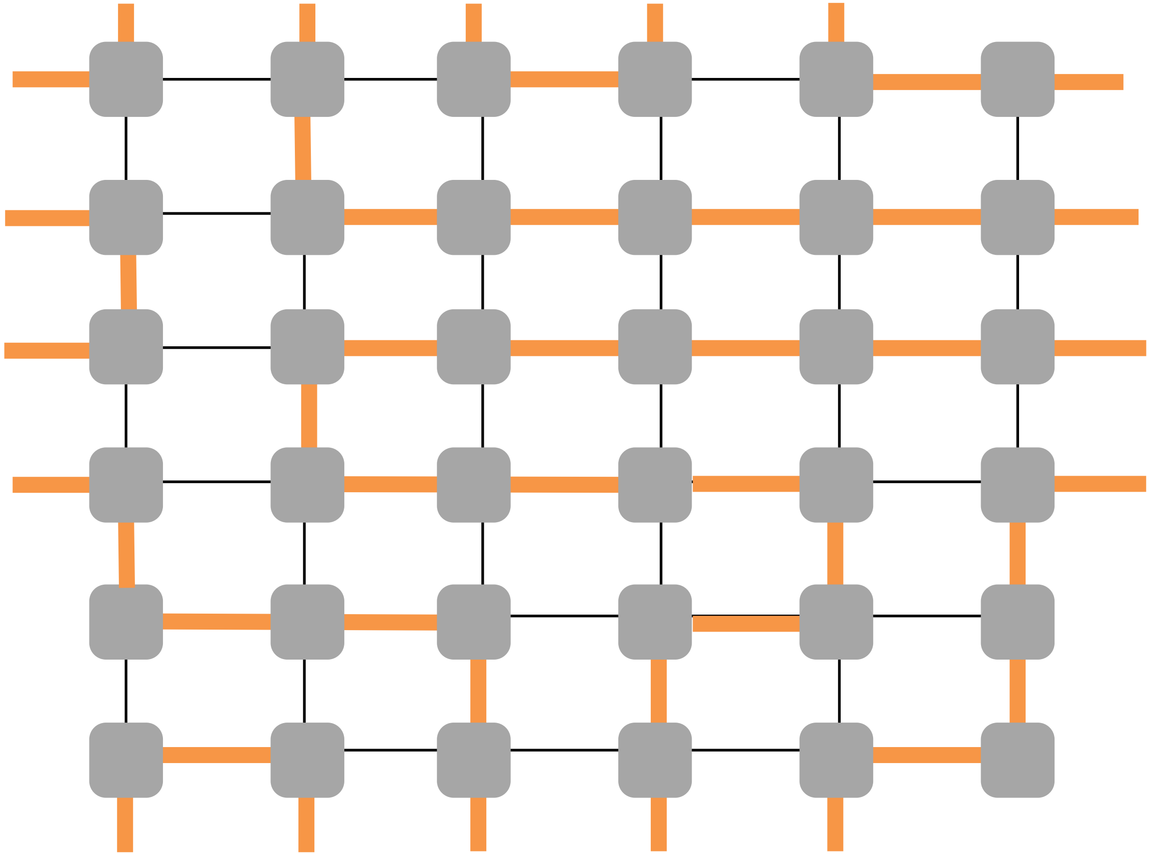
Ideal Resilience: Example 2-dim Torus?



Ideal Resilience: Example 2-dim Torus?



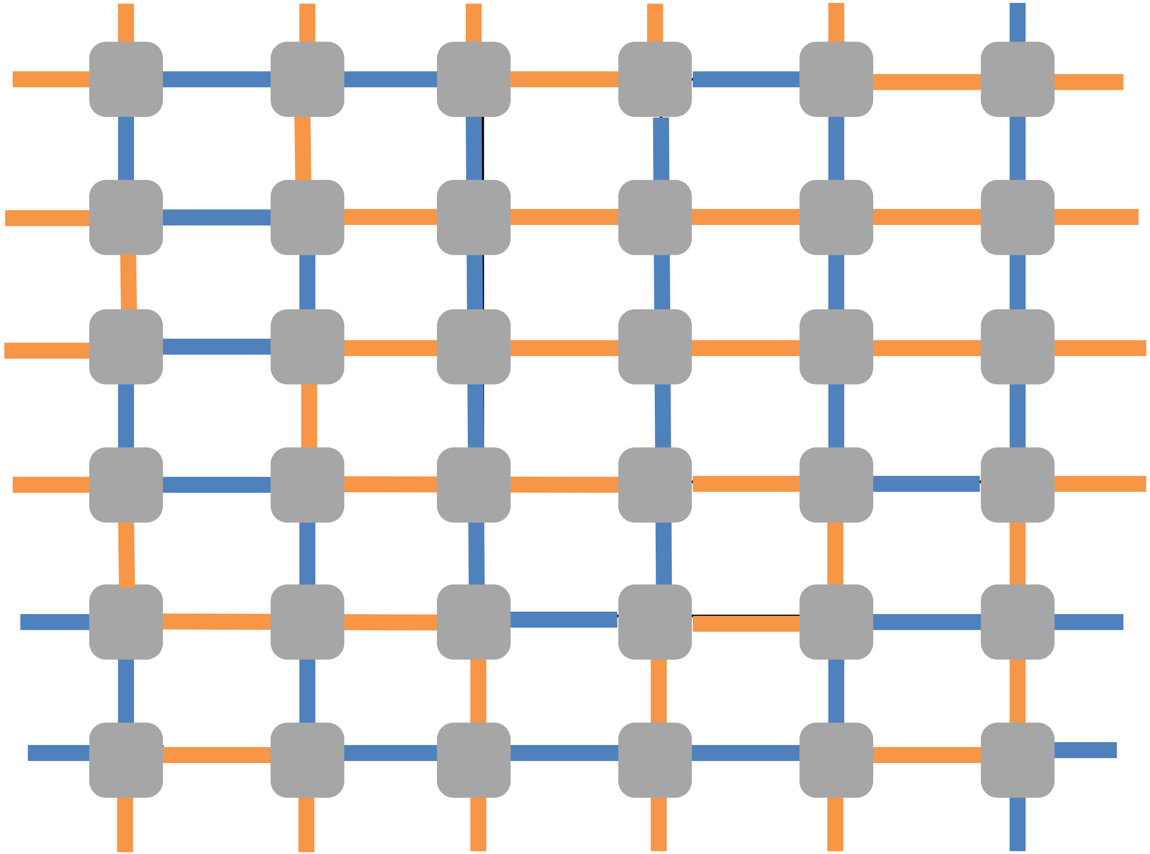
Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)

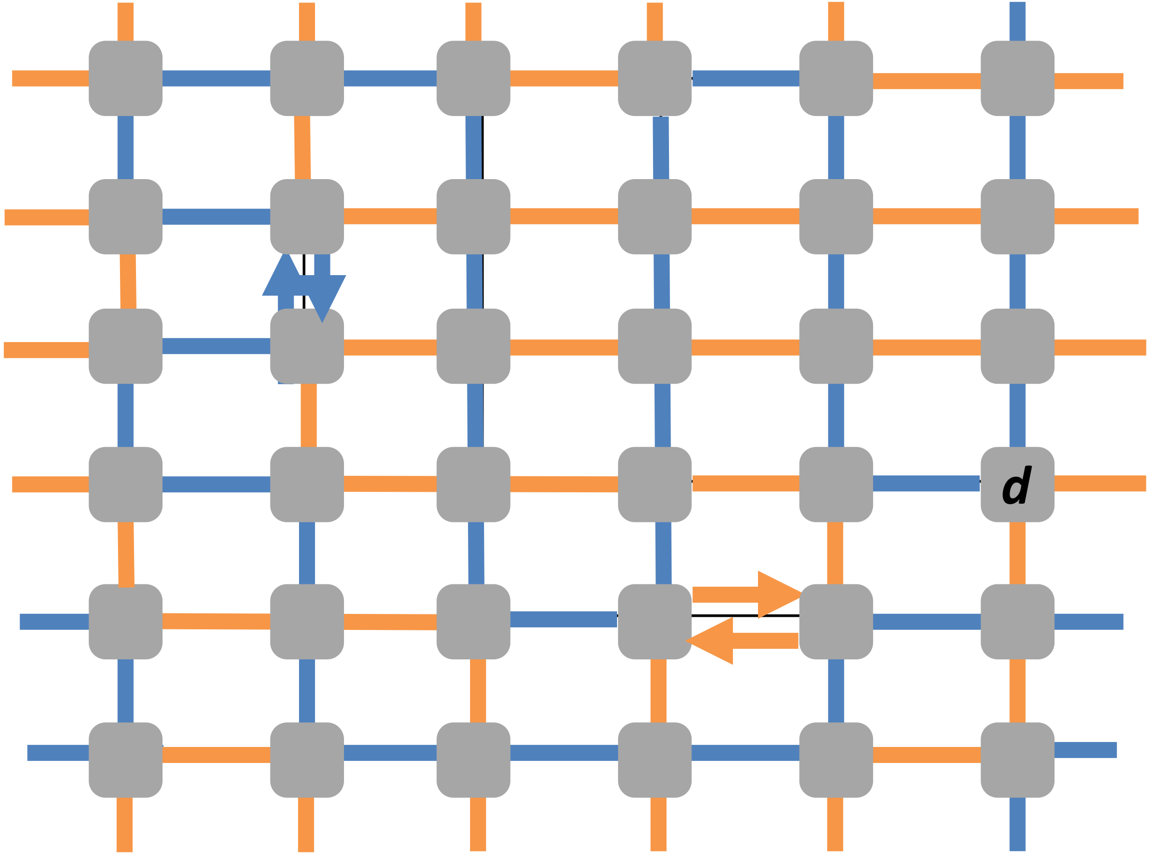
1st Hamilton cycle

Idea: Decomposition into Hamilton Cycles



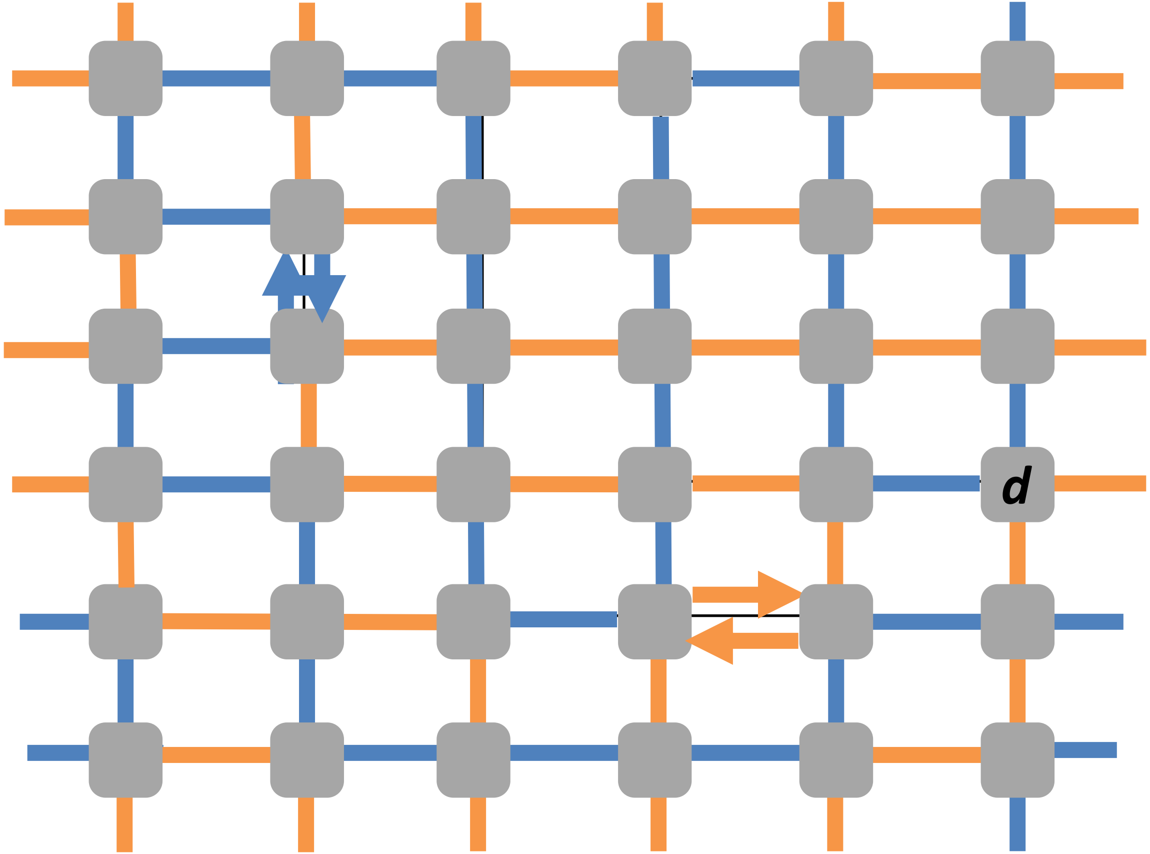
- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)

Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)
- Can route in both directions: *4-arc-disjoint* HCs

Idea: Decomposition into Hamilton Cycles



- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)
- Can route in both directions: *4-arc-disjoint* HCs

3-resilient routing to destination d:

- go along *1st directed HC*, if hit failure, *reverse* direction
- if again failure switch to *2nd HC*, if again failure *reverse direction*
- No more failures possible!

Ideal Resilience with Hamilton Cycles

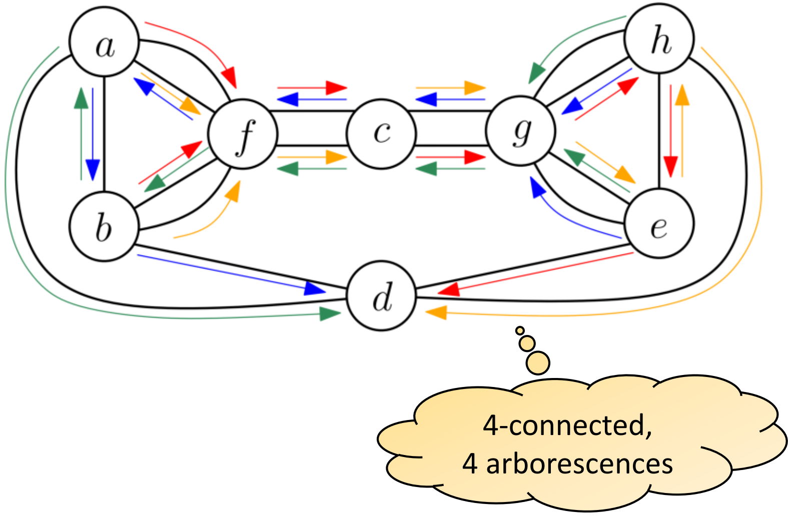
Chiesa et al.: if k-connected graph has k arc disjoint Hamilton Cycles, k-1 resilient routing can be constructed!

What about graphs which cannot be decomposed into Hamilton cycles?

Chiesa et al. **On the Resiliency of Static Forwarding Tables.** IEEE/ACM Transactions on Networking (ToN), 2017.

Ideal Resilience in General k-Connected Graphs

- Use directed trees (i.e. *arborescences*) instead of Hamilton cycles
  - *Arc-disjoint*, spanning, and *rooted* at destination
- Classic result: k-connectivity guarantees k-arborescence decomposition



Basic idea:

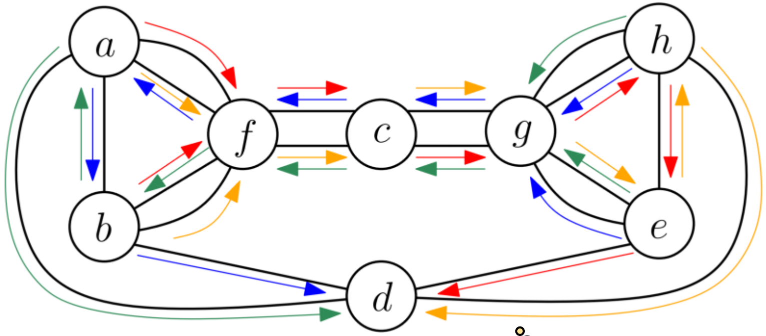
- Idea: route towards root on one arborescence
- After failure: change arborescence (e.g. in circular fashion)
- Incoming port defines current arborescence
- After k-1 failures: At least one arborescence intact

J. Edmonds, **Edge-disjoint branchings**. Combinatorial Algorithms, 1972.

Ideal Resilience in General k-Connected Graphs

- Use directed trees (i.e. *arborescences*) instead of Hamilton cycles
  - *Arc-disjoint*, spanning, and *rooted* at destination

- Classic result: k-connectivity guarantees k-arborescence decomposition



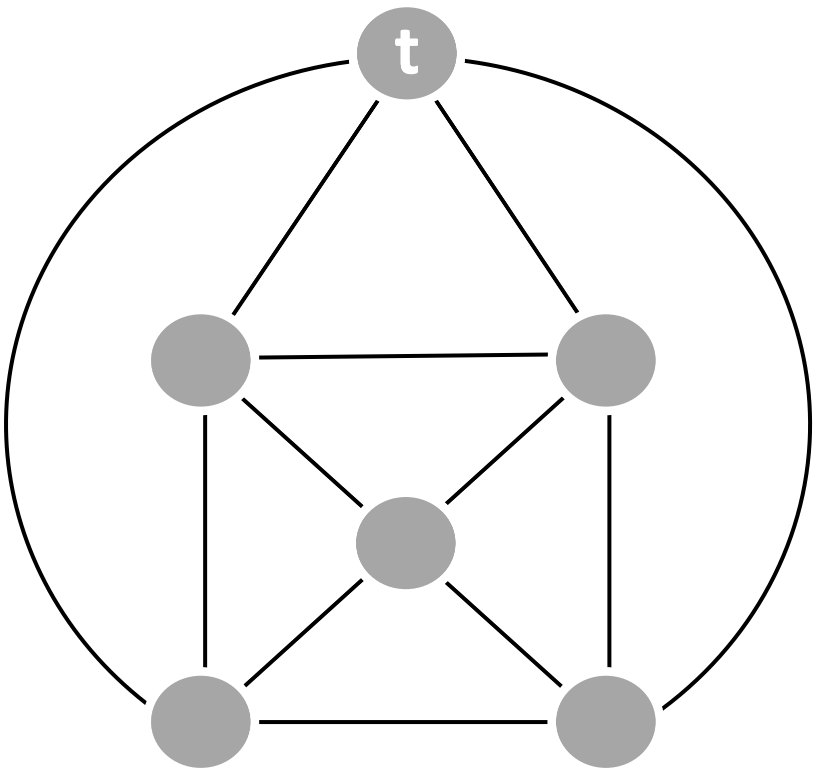
4-connected,
4 arborescences

Basic idea:
- Idea: route towards root on one arborescence
- After failure: change arborescence (e.g. in circular fashion)
- Incoming port defines current arborescence
- After k-1 failures: At least one arborescence intact

The challenge: how to avoid earlier tree?

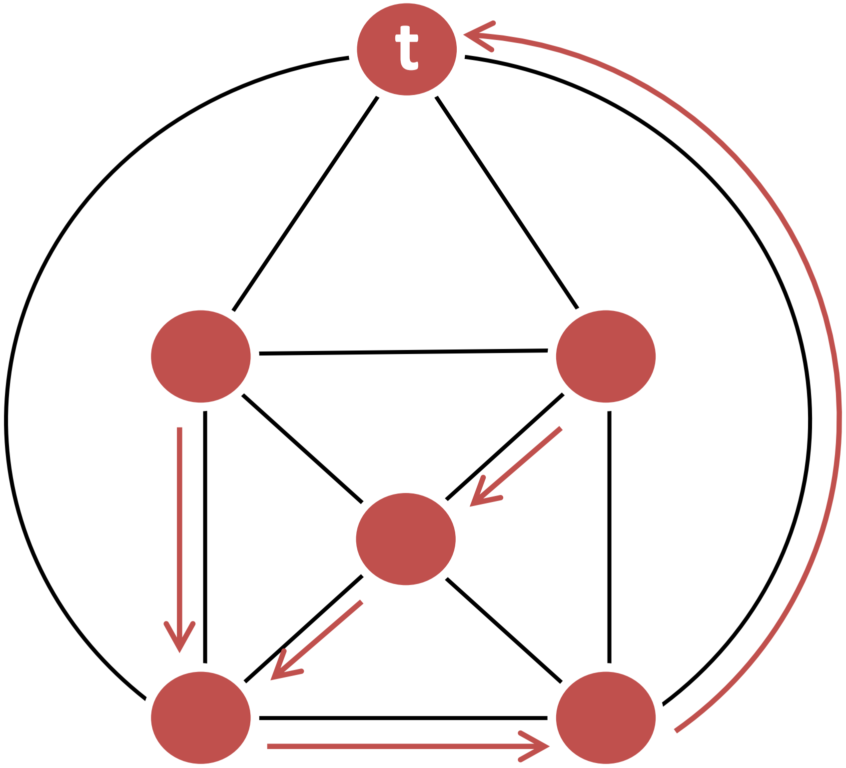
J. Edmonds, **Edge-disjoint branchings**. Combinatorial Algorithms, 1972.

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



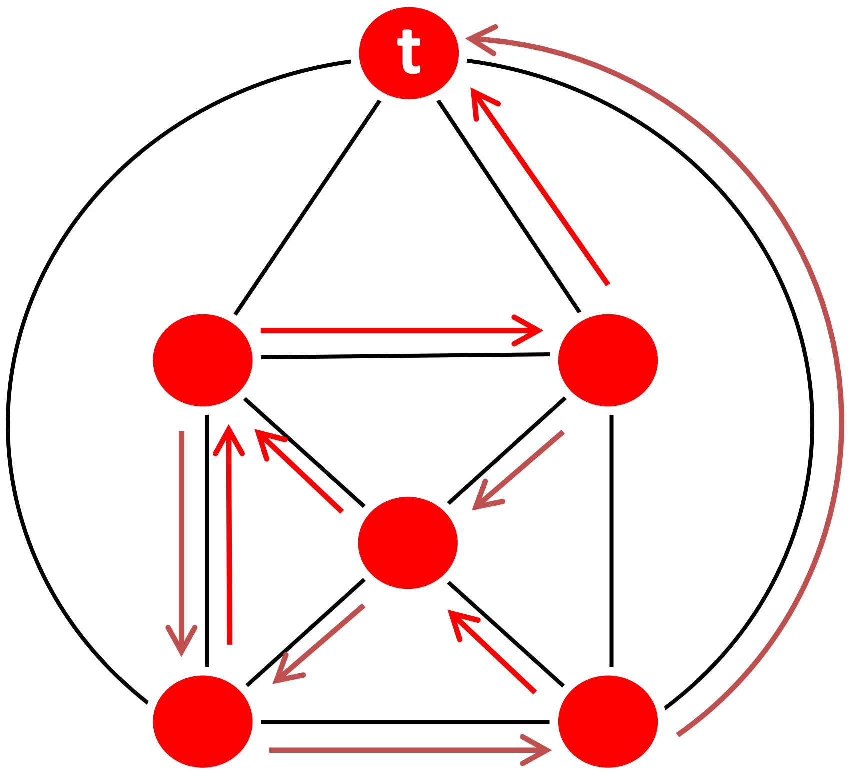
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



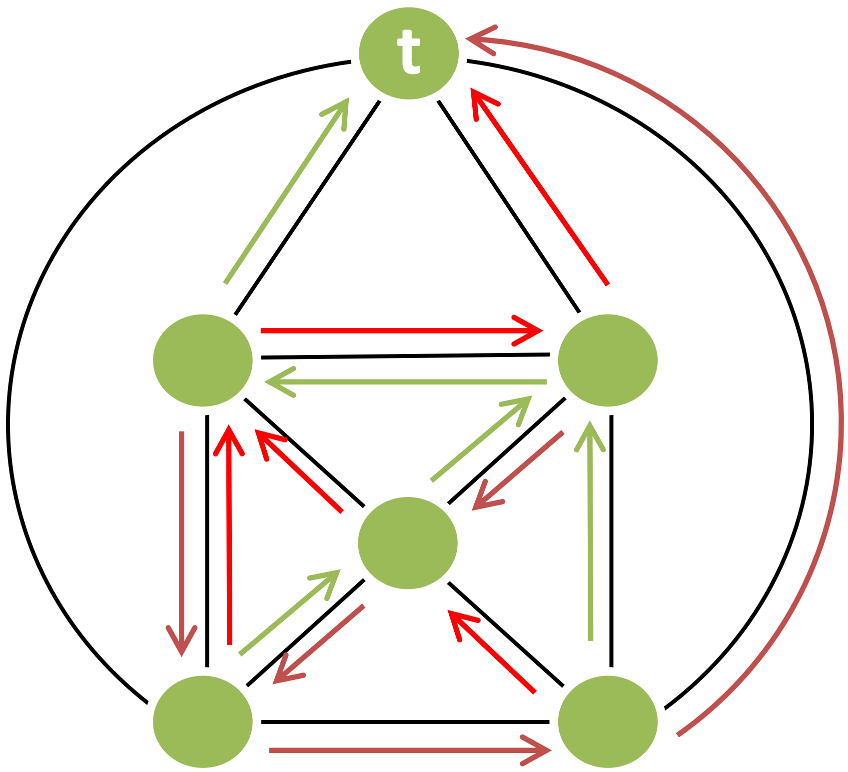
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



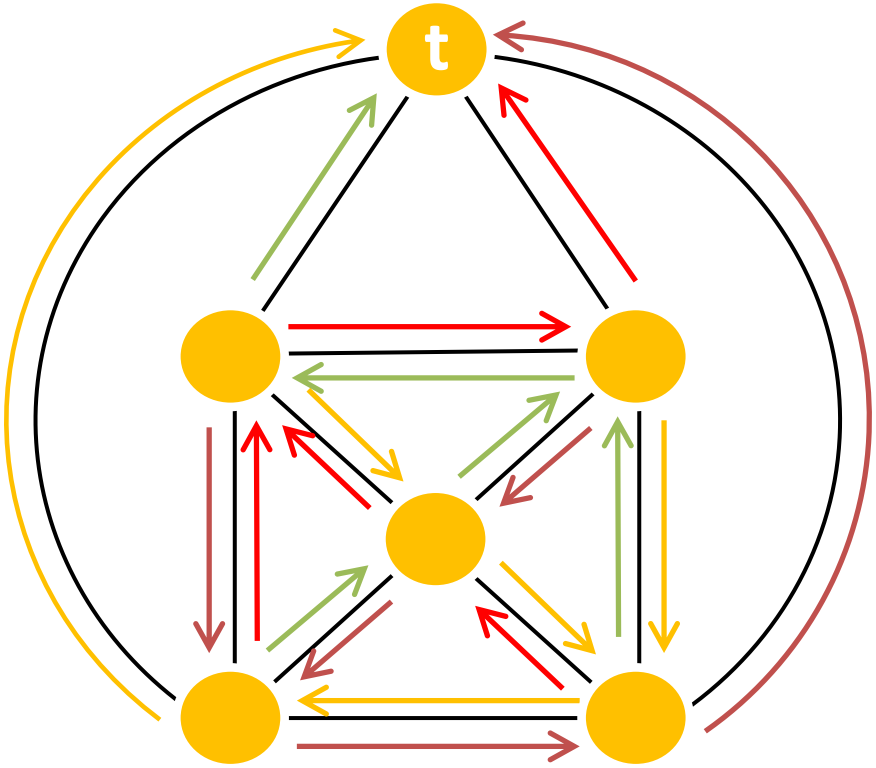
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



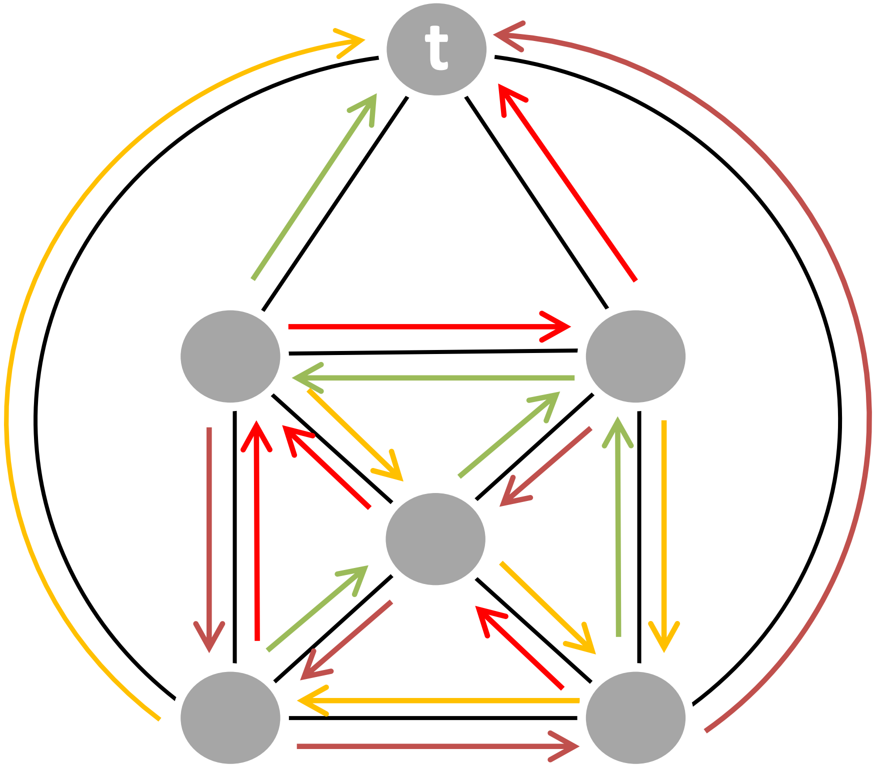
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



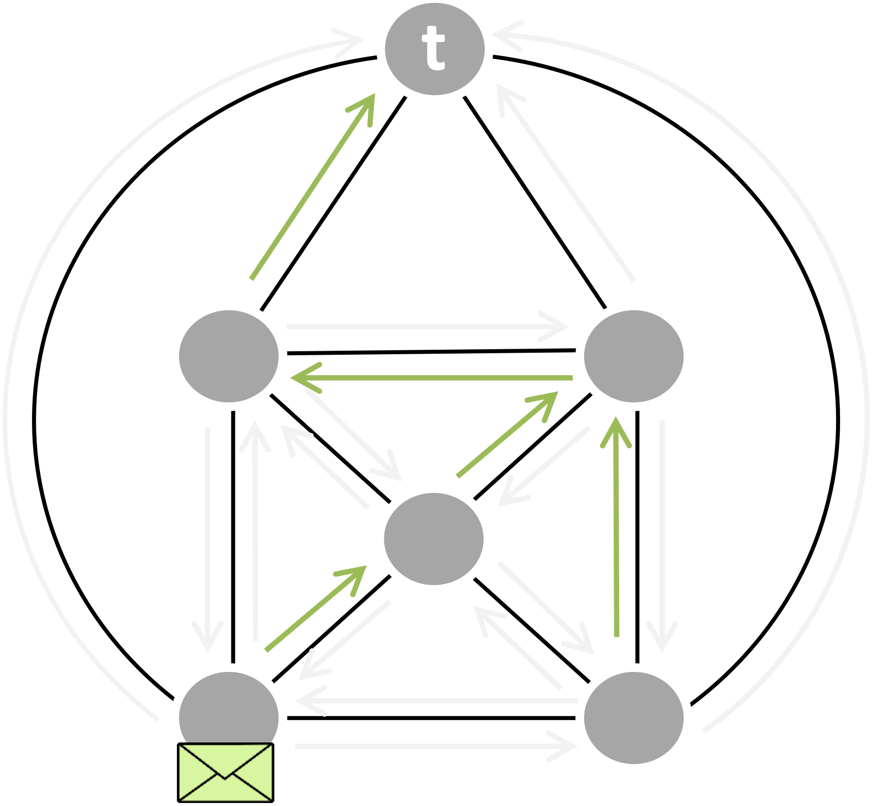
Credits: Marco Chiesa

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]



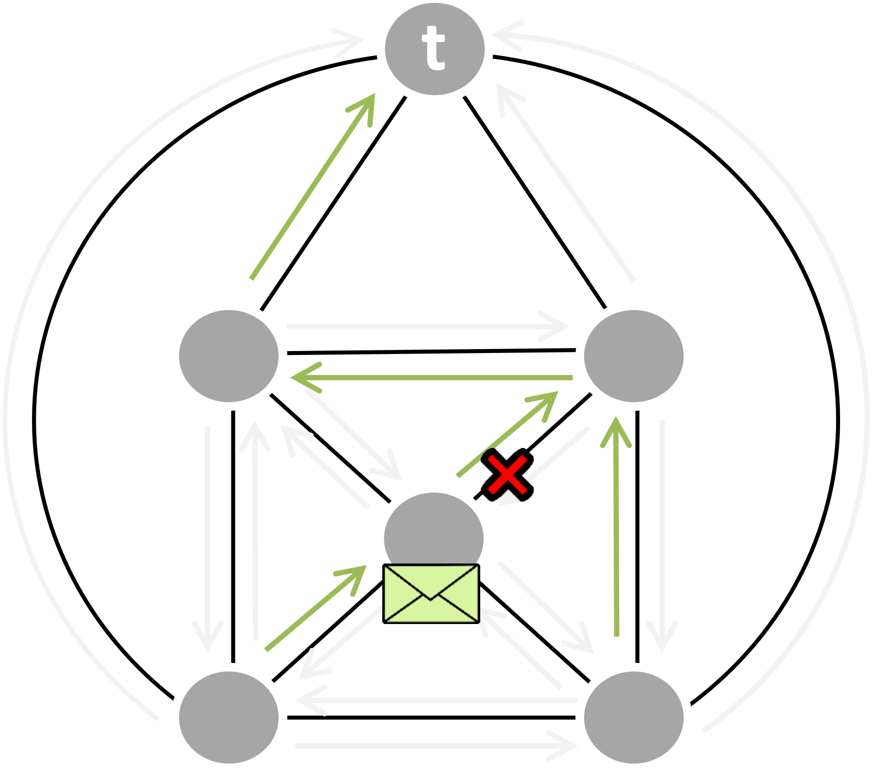
Credits: Marco Chiesa

General technique: routing along the same tree



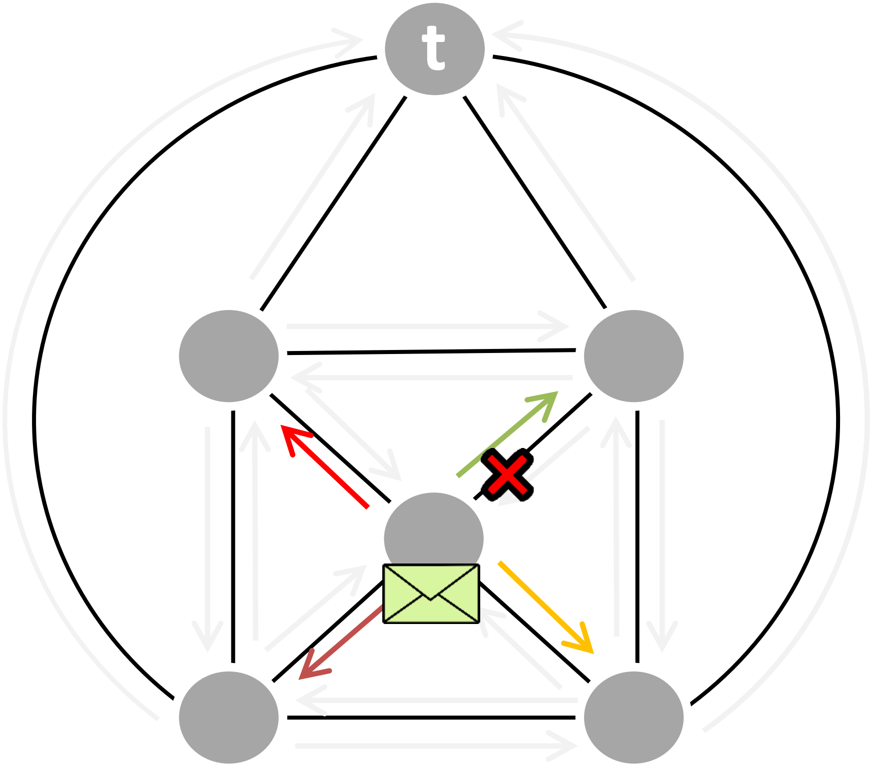
Credits: Marco Chiesa

When a failed link is hit…



Credits: Marco Chiesa

… how do we choose the next arborescence?



Credits: Marco Chiesa

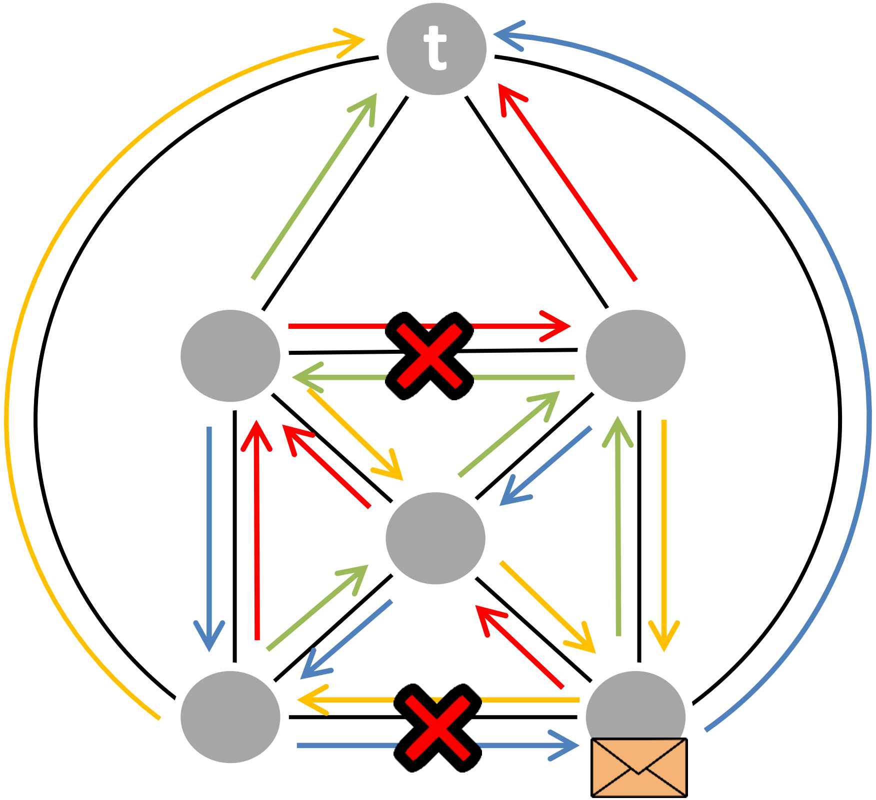
But how do we choose the next arborescence?

Circular-arborescence routing:

- compute an order of the arborescences
- switch to the next arborescence when hitting a failed link

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order



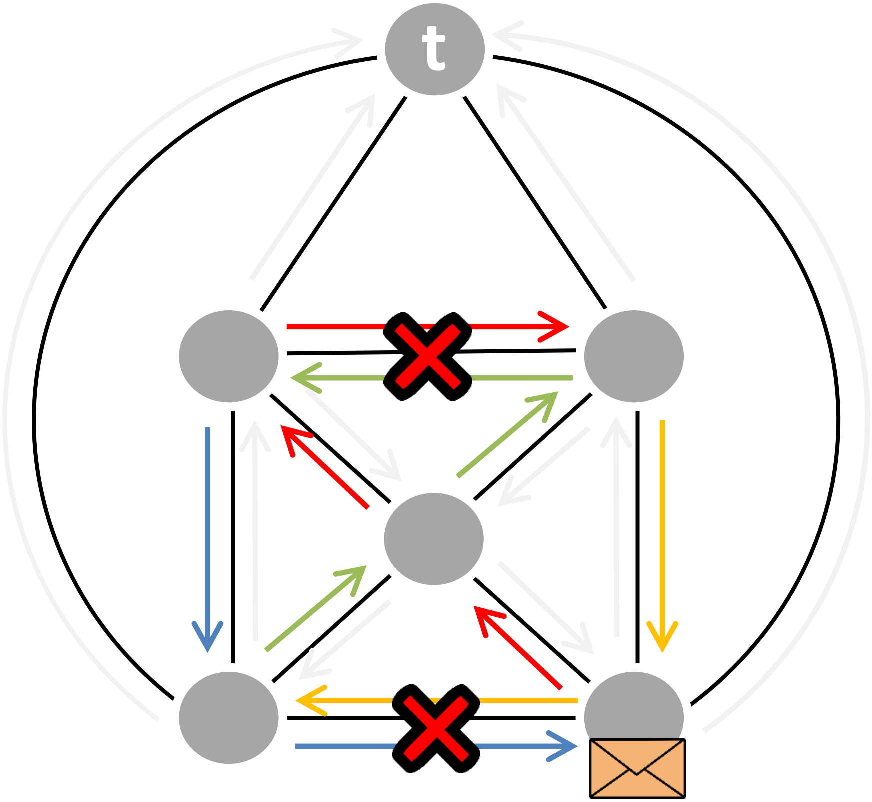
Intuition: each single failure may affect two arborescences

Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

Go along arborescence 1 to destination...



Intuition: each single failure may affect two arborescences

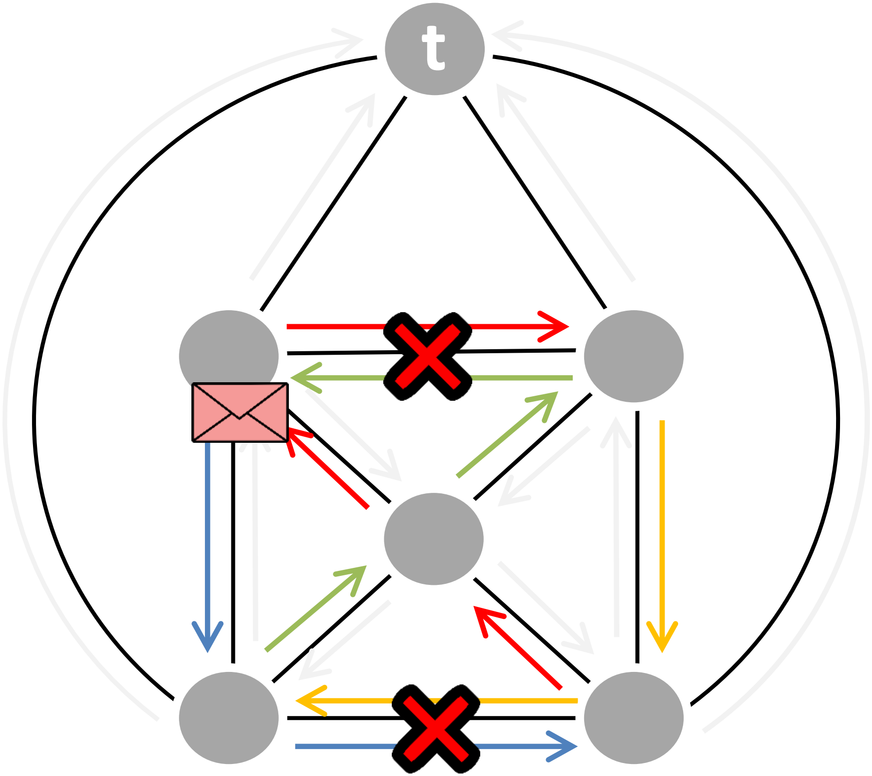
Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4

Go along arborescence 2 to destination...



Intuition: each single failure may affect two arborescences

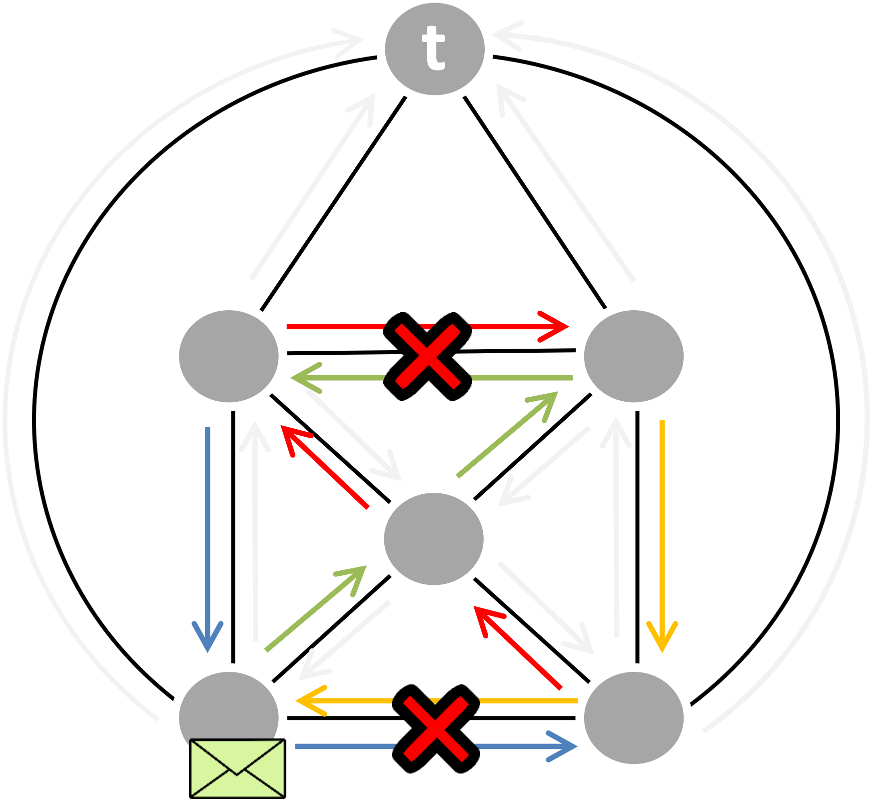
Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4

Go along arborescence 3 to destination...



Intuition: each single failure may affect two arborescences

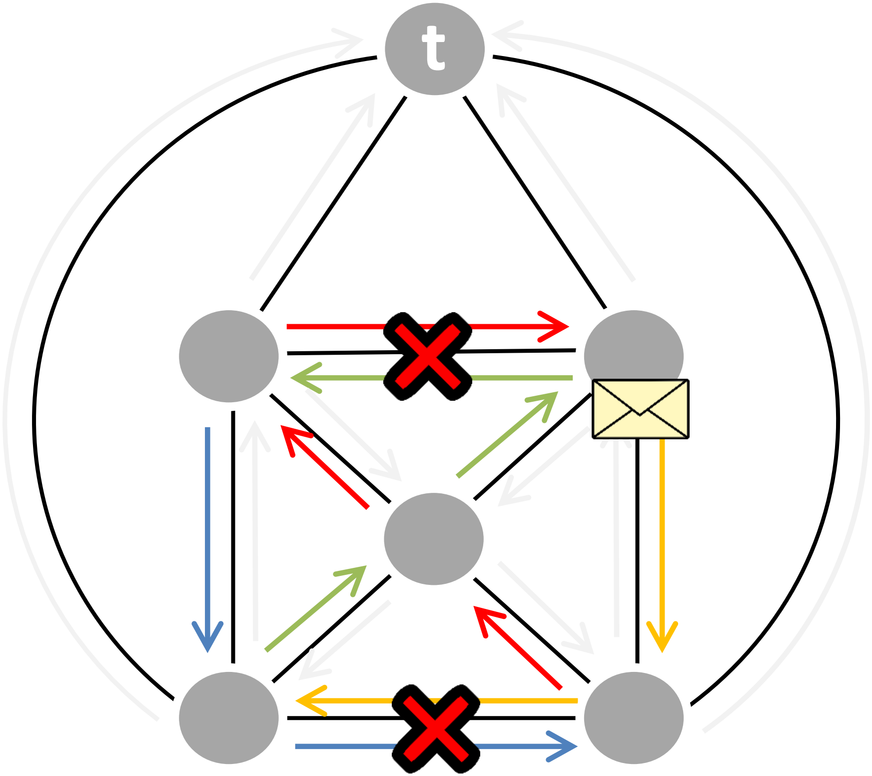
Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order

1	2	3	4

Go along arborescence 4 to destination...

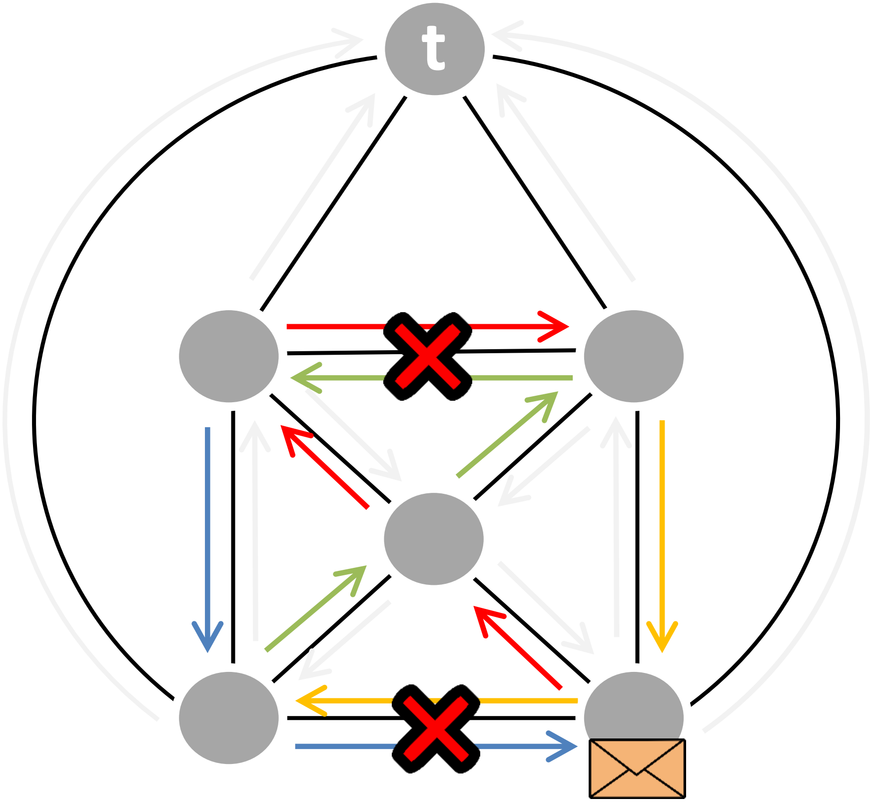


Intuition: each single failure may affect two arborescences

Credits: Marco Chiesa

Circular arborescence-routing is (k/2-1)-resilient

Arborescence order



Intuition: each single failure may affect two arborescences

All k=4 arborescences used (2 failures disconnected affected all four): LOOP!

Credits: Marco Chiesa

An Alternative Algorithm: Bouncing Arborescence

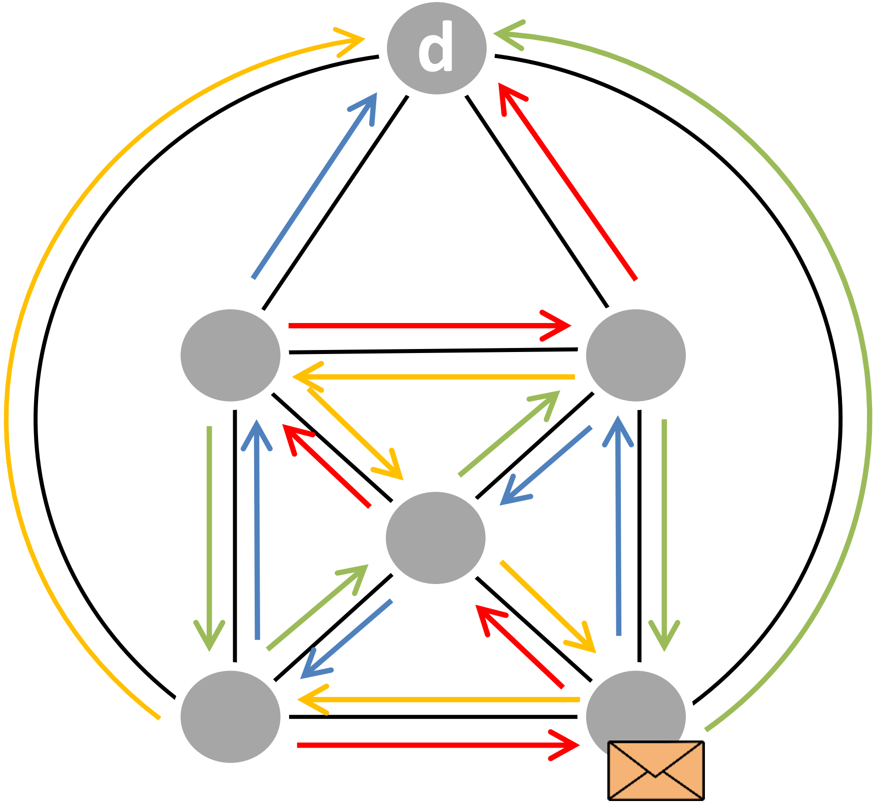
Bouncing-arborescence algorithm:

- Reroute on the tree that shares the failed link

This algorithm is ***1-resilient***.

Bouncing-Arborescence is 1-Resilient

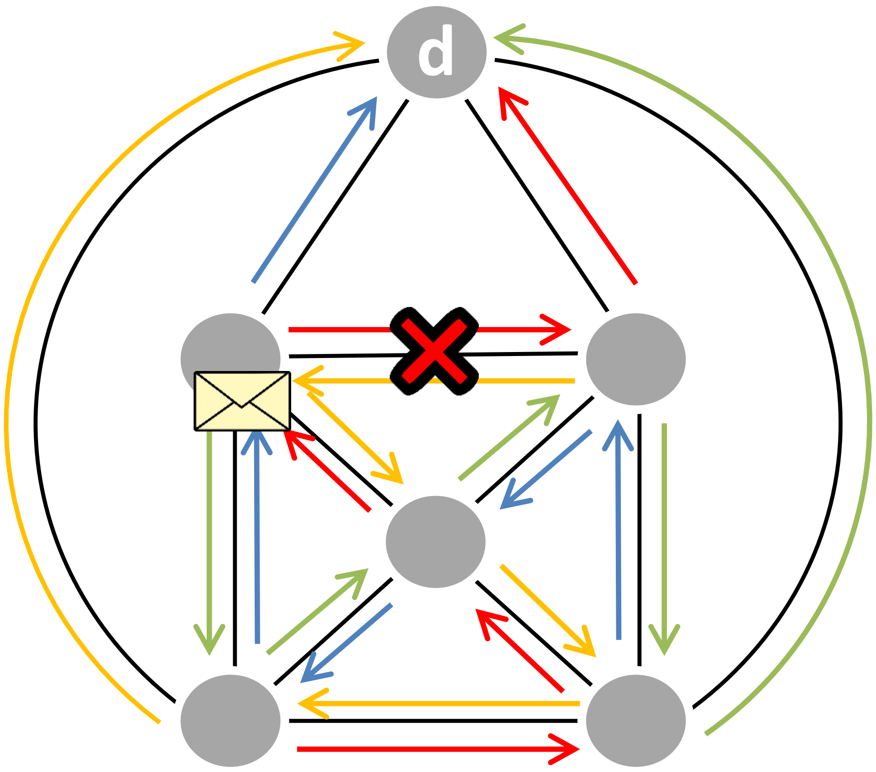
Start with red…



Credits: Marco Chiesa

Bouncing-Arborescence is 1-Resilient

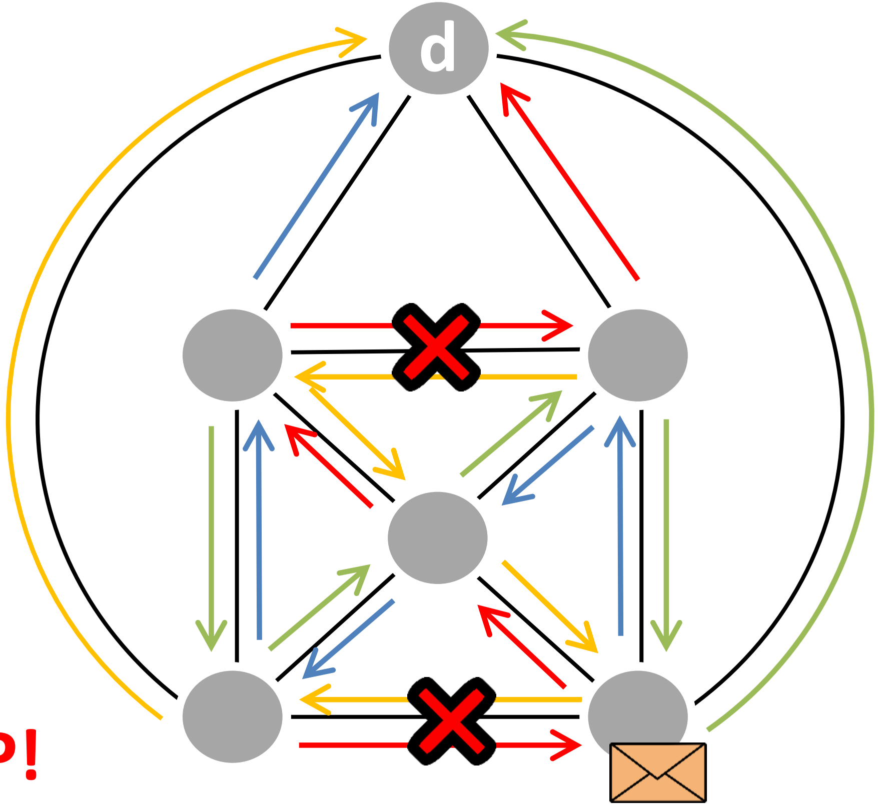
… bounce to yellow…



Credits: Marco Chiesa

Bouncing-Arborescence is 1-Resilient

… bounce to red (again!)…

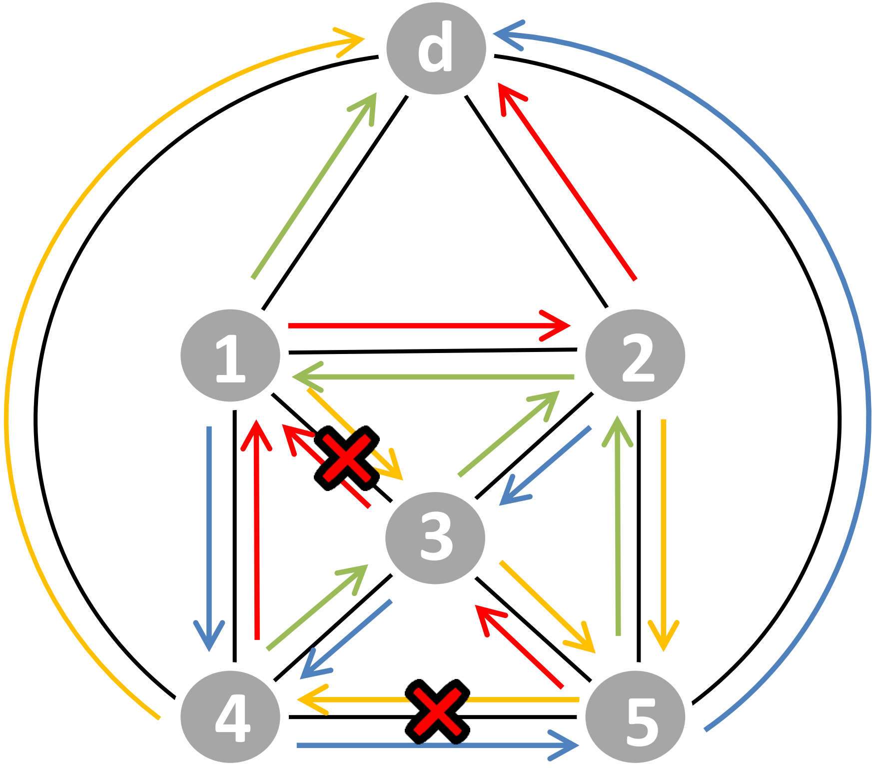


LOOP!

Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

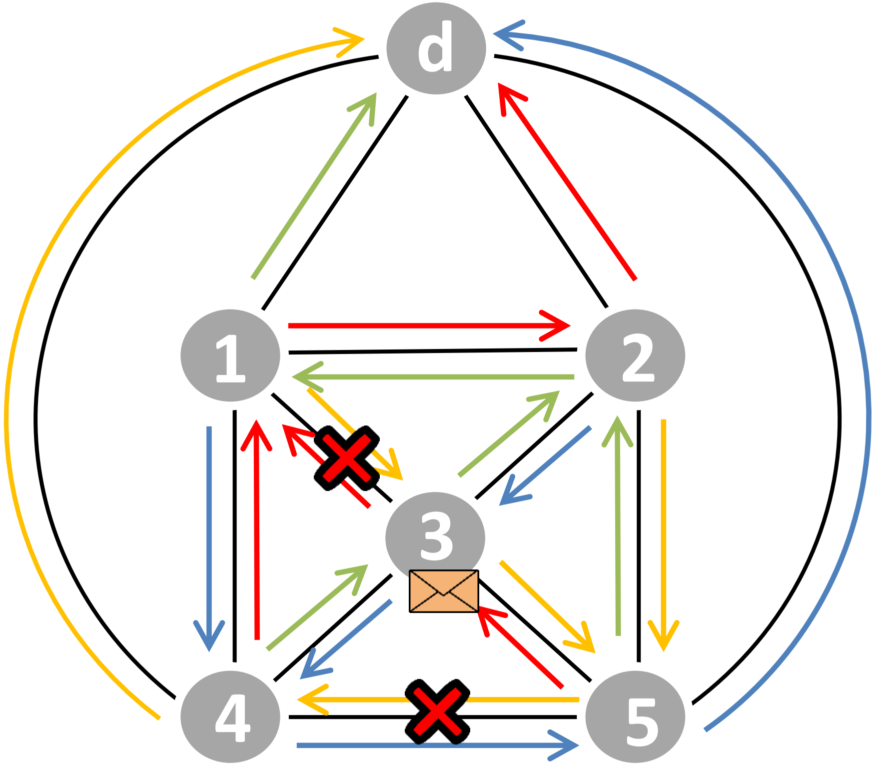
- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures



Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

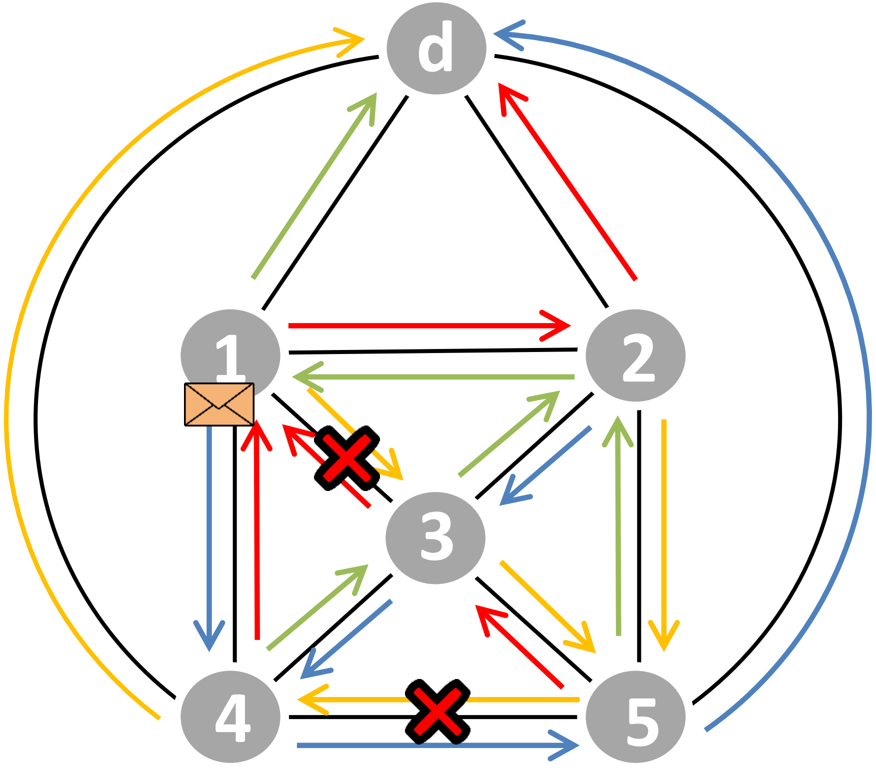
- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures
  - (3,1) is not well-bouncing



Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures
  - (3,1) is not well-bouncing
  - (1,3) is well-bouncing

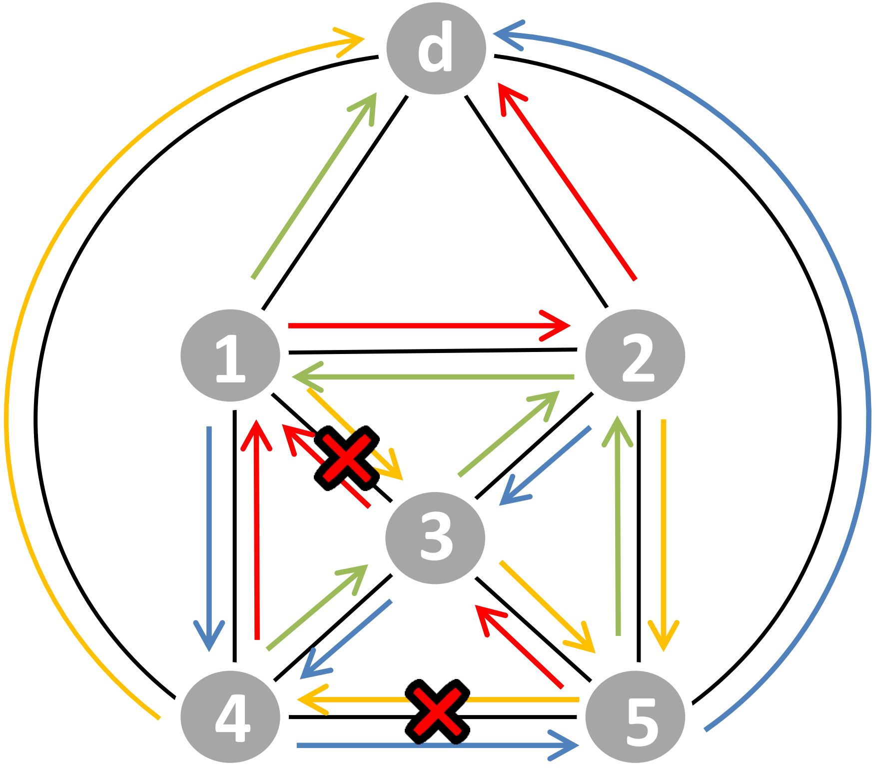


Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures
  - (3,1) is not well-bouncing
  - (1,3) is well-bouncing

- Define **good arborescence**:
  - every failed arc is well-bouncing

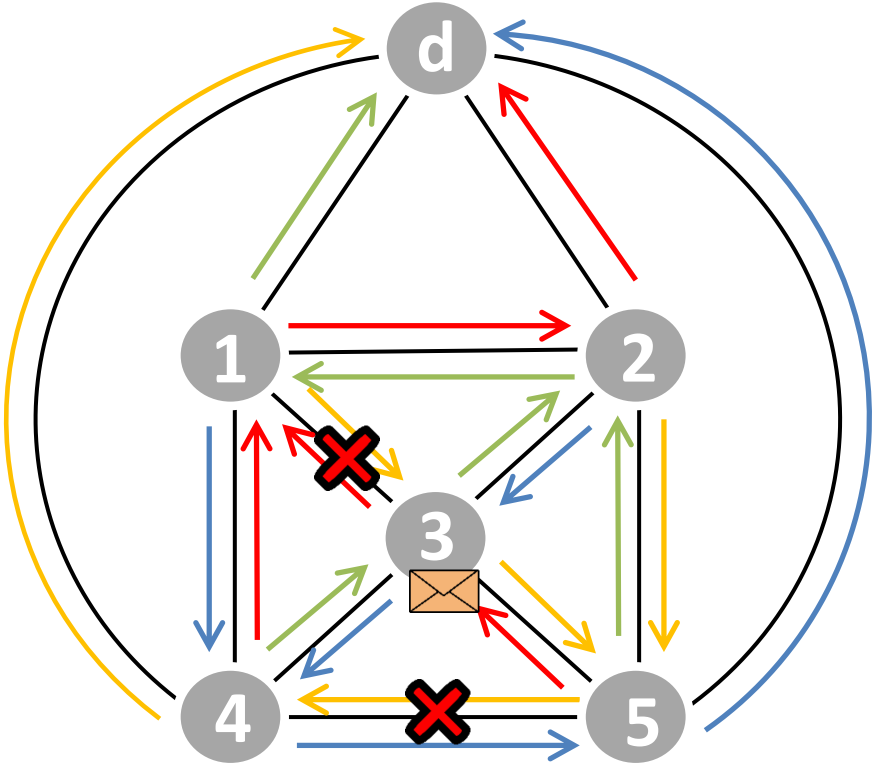


Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures
  - (3,1) is not well-bouncing
  - (1,3) is well-bouncing

- Define **good arborescence**:
  - every failed arc is well-bouncing
  - Red is not a good arborescence

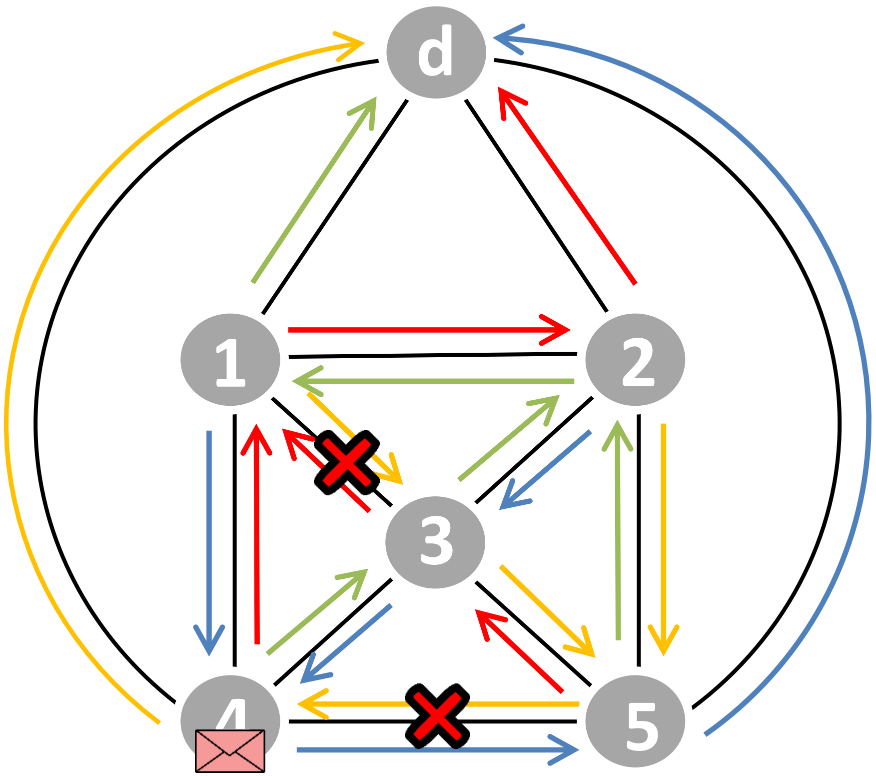


Credits: Marco Chiesa

Idea: Bounce on „Good Arborescences“

- Define **well-bouncing arc**:
  - When bounce get to the destination
  - Without hitting any other failures
  - (3,1) is not well-bouncing
  - (1,3) is well-bouncing

- Define **good arborescence**:
  - every failed arc is well-bouncing
  - Red is not a good arborescence
  - Blue is a good arboresence



Credits: Marco Chiesa

Ideas

- One can show that there is always a good arborescence

- An tempting idea:
  – route on an arborescence X until a failed link is hit:
    - if X is a good arborescence, bounce!
    - otherwise, route circular

- Too good to be true:
  – The "goodness" of an arborescence depends on the actual set of failed links!
  – How do we know a arborescence is good?

Credits: Marco Chiesa

Resilience Criteria

Ideal resilience

Given a *k*-connected graphs, we can tolerate *any k-1 link failures.*

Perfect resilience

Any source *s* can always reach any destination *t* as long as the unterlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.

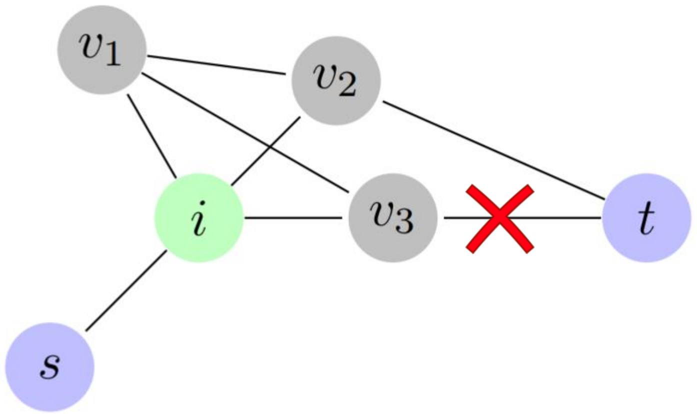
Resilience Criteria

Perfect resilience is impossible to achieve in general.

Relevant Neighbors

- Routing table of node i: matches in-ports of i to out-ports of i
  - ... depending on the incident failures

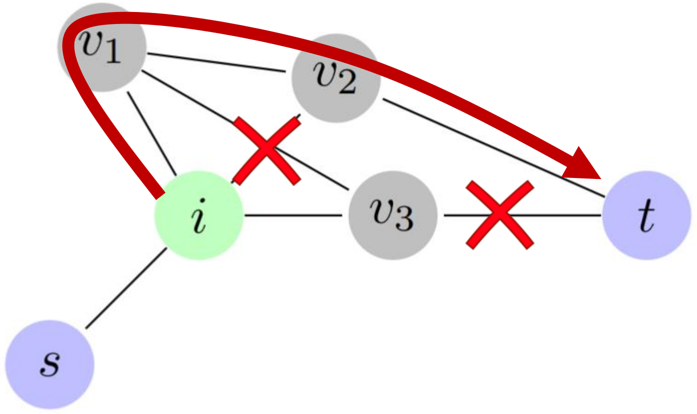
- But not all neighbors are **relevant**: only if potentially required to reach destination!
  - *Without local failures*: just v_2, v_3 for i, since v_1 does not give extra connectivity



Relevant Neighbors

- Routing table of node i: matches in-ports of i to out-ports of i
  - ... depending on the incident failures

- But not all neighbors are **relevant**: only if potentially required to reach destination!
  - *Without local failures*: just v_2, v_3 for i, since v_1 does not give extra connectivity
  - *With additional failures* v_1 becomes relevant, since v_1 might be only choice to reach destination t
    - Note: v_1 is unaware of these non-incident failures!

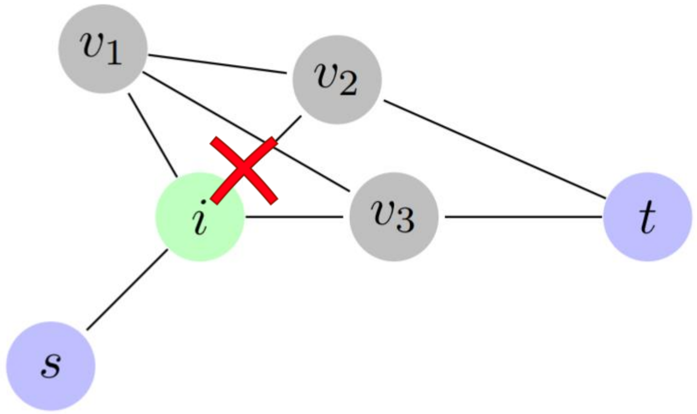


High-level definition of ***relevant***: From the local view-point of the node i, a relevant neighbor might be only neighbor to reach destination (without taking a detour over a current neighbor).

How to Achieve Perfect Resilience?

- Necessary: need to *try all relevant* neighbors
  - Here, if local link to v_2 broken: v_1 and v_3

- That is, if packet
  - comes from v_3: eventually try v_1
  - comes from v_1: eventually try v_3



Impossibility: On Planar Graphs

Some observations:

- Additional failures only ***add relevant neighbors*** to nodes
- Any node of ***degree 2*** of G after failures must forward packets with incoming port p to port p'
- If all neighbors are relevant, the forwarding function of a node must be a ***cyclic permutation***

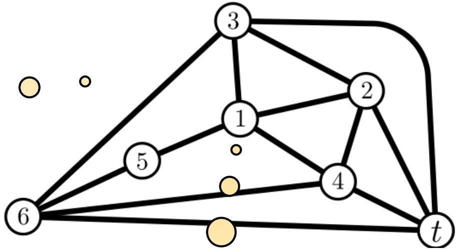
Impossibility: On Planar Graphs

Some observations:

- Additional failures only ***add relevant neighbors*** to nodes
- Any node of ***degree 2*** of G after failures must forward packets with incoming port p to port p'
- If all neighbors are relevant, the forwarding function of a node must be a ***cyclic permutation***

Idea of the counter example:

All neighbors of all nodes are relevant (even without failures).



So we must fix a permutation for node 1.

Considered node 1 will not see any local failures.

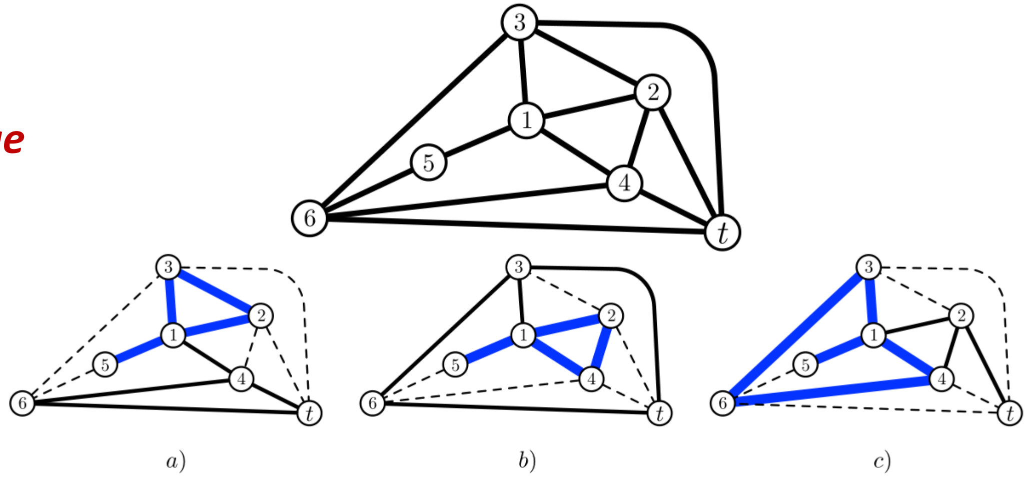
Impossibility: On Planar Graphs

Some observations:

- Additional failures only ***add relevant neighbors*** to nodes
- Any node of ***degree 2*** of G after failures must forward packets with incoming port p to port p'
- If all neighbors are relevant, the forwarding function of a node must be a ***cyclic permutation***

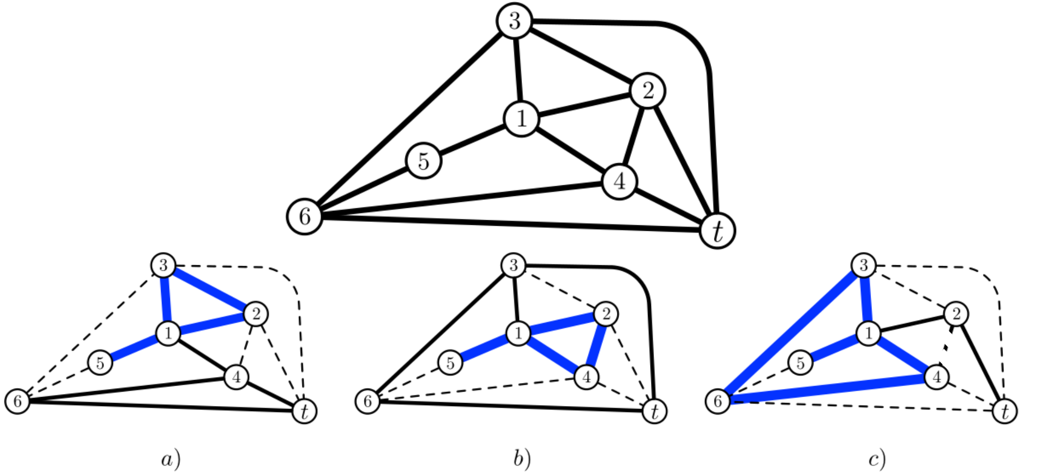
Proof idea, with three cases:

- If the ***dashed*** links fail (***non-local*** to node 1), in any forwarding pattern, packets will be stuck in one of the ***blue loops***…
- … even though there is at least one ***remaining path*** to the target



Go through all possible permutations @1 and give counter example.

Impossibility: On Planar Graphs

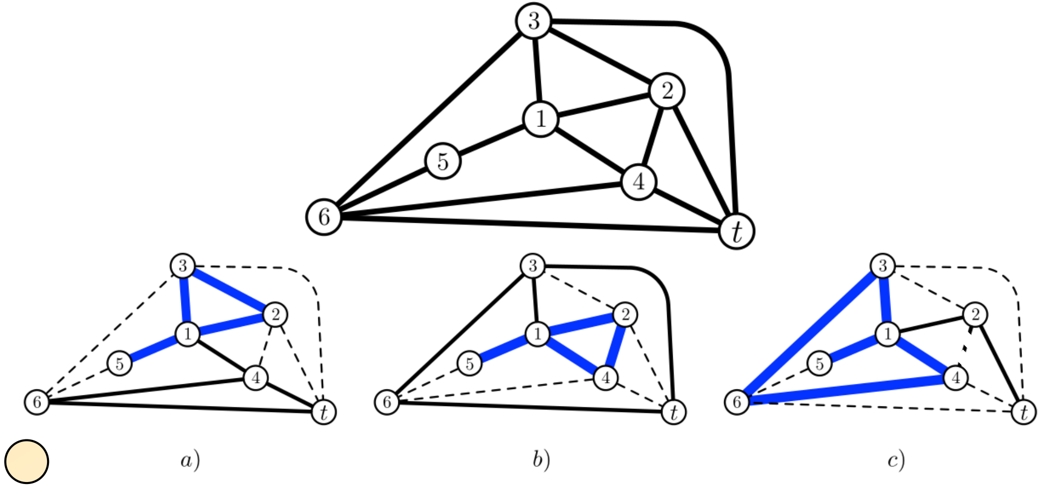


Arriving on inport 5, forwarded to 2.

For node 1:
5->2 implies
(5,2,3,4) (b)
(5,2,4,3) (a)

Possible cyclic permutations: when a packet arrives from 2, due to cyclic permutation, it can only be forwarded to either 3 or 4. Leads to *loops* in scenarios (b) (4 goes to 5, 2 can only go to 4) and (a) (3 goes to 5, 2 can only go to 3), respectively.

Impossibility: On Planar Graphs



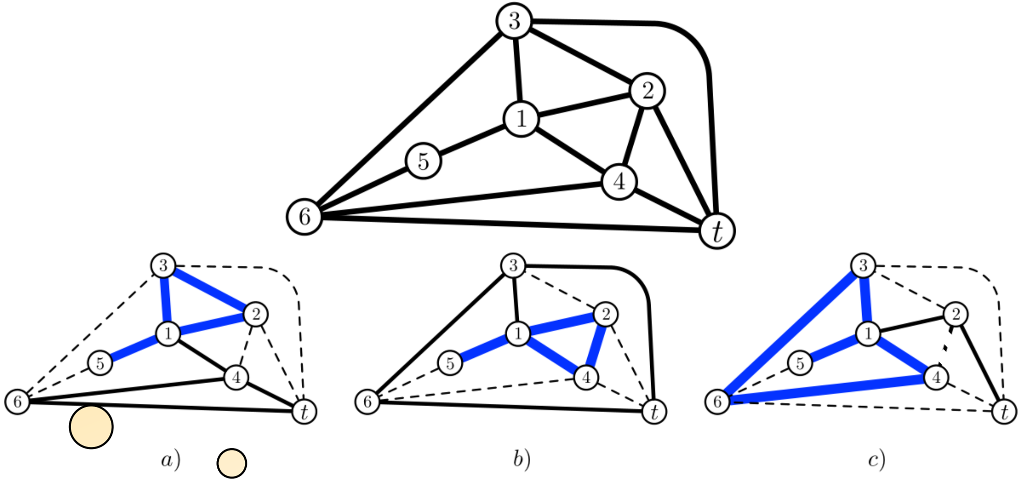
Arriving on inport 5, forwarded to 3.

For node 1:
5->2 implies
(5,2,3,4) (b)
(5,2,4,3) (a)

For node 1:
5->3 implies
(5,3,4,2) (a)
(5,3,2,4) (c)

Possible cyclic permutations: when a packet then arrives on port 4, it can only be forwarded to either 2 or 5. Leads to *loops* in scenarios (a) (2 will go to 5, 5 can only go to 1 and 3 only to 2) and (c) (5 goes to 3, 4 goes to 5, rest degree-2), respectively.

Impossibility: On Planar Graphs



Arriving on inport 5, forwarded to 4.

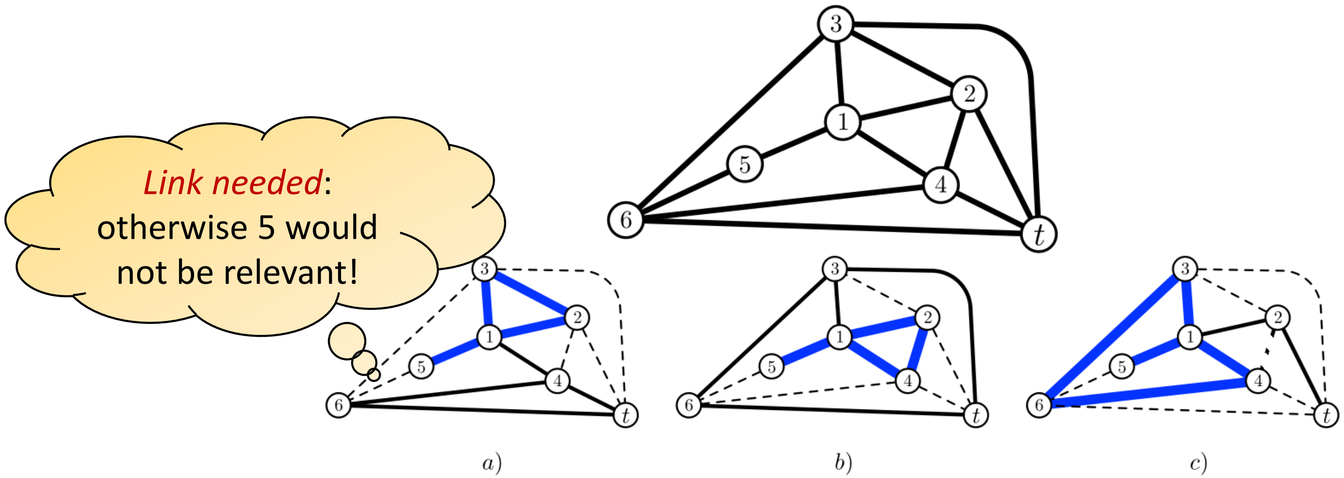
For node 1: 5->2 implies (5,2,3,4) (b) (5,2,4,3) (a)

For node 1: 5->3 implies (5,3,4,2) (a) (5,3,2,4) (c)

For node 1: 5->4 implies (5,4,2,3) (c) (5,4,3,2) (b)

Possible cyclic permutations: packet arriving on port 3 can only be forwarded to either 5 or 2. Leads to *loops* in scenarios (c) and (b), respectively.

Impossibility: On Planar Graphs



For node 1:
5->2 implies
(5,2,3,4) (b)
(5,2,4,3) (a)

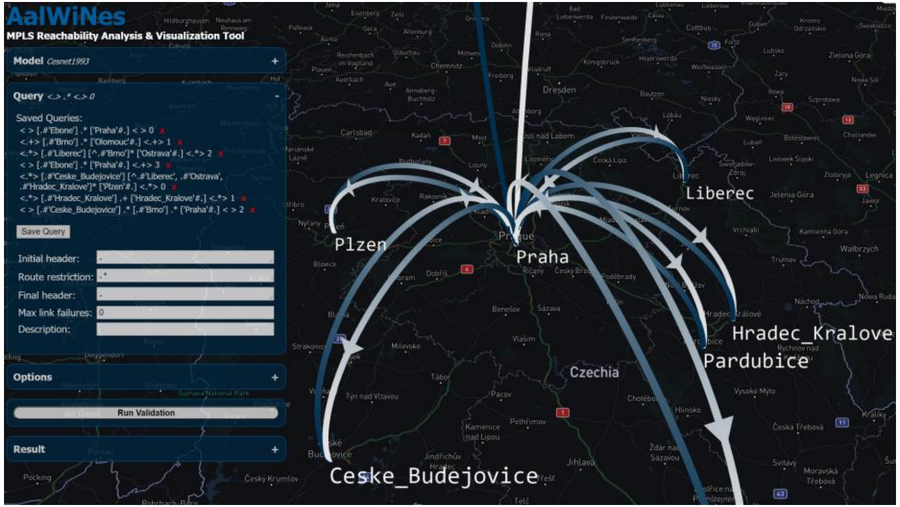
For node 1:
5->3 implies
(5,3,4,2) (a)
(5,3,2,4) (c)

For node 1:
5->4 implies
(5,4,2,3) (c)
(5,4,3,2) (b)

Possible cyclic permutations: packet arriving on port 3 can only be forwarded to either 5 or 2. Leads to *loops* in scenarios (c) and (b), respectively.

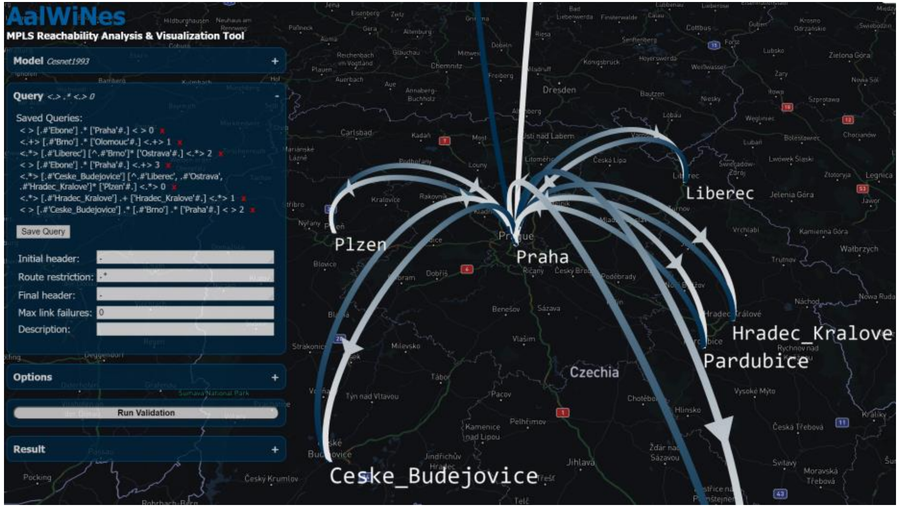
A Pity: Planar Graphs Are Important

- Internet Topology Zoo and Rocketfuel topologies
  - 88% of the graphs are ***planar***



A Pity: Planar Graphs Are Important

- Internet Topology Zoo and Rocketfuel topologies
  - 88% of the graphs are ***planar***
  - However:
    - Almost a third (32%) belong to the family of ***cactus*** graphs
    - Roughly half of the graphs (49%) are ***outerplanar***
    - … and they work ☺



Where Can Perfect Resilience Be Achieved?

For example on **outerplanar graphs**:

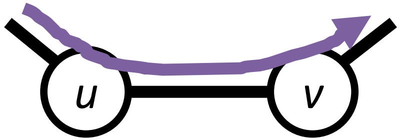
- Via *geometric routing*, well studied in sensor networks etc.
- Embed graph in the plane s.t. all nodes are on the outer face
  - Note: If a link l belongs to the outer face of a planar graph G, it also belongs to the outer face for all subgraphs of G
- Apply *right-hand rule* to forwarding (skipping failures)
  - Ensures packets use only the links of the outer face and do not change the direction despite failures
- Strategy traverses all nodes on the outer face

- Also works for any graph which is *outerplanar without the source* (e.g., K4)

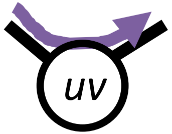
Some Observations

- K_5, $K\_3,3$: *no perfect resilience*

- Perfect resiliency on graph G -> any *subgraph* G‘ of G also allows for perfect resiliency
  - Idea: Take routing on G, fail edges to create G‘, routing must still work



- Contraction works as well, by a simulation argument
  - A bit technical

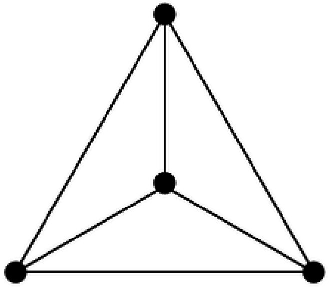


- Combined: Perfect resilience on graph G -> any minor G‘ of G as well
  - But since K_5, $K\_3,3$ not: *non-planar graphs not perfectly resilient*

What we know about perfect resilience

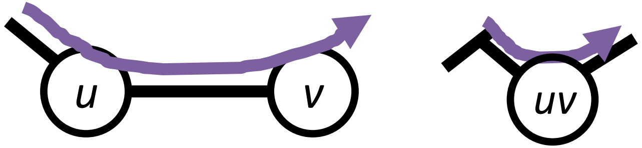
Possible:

- On all outerplanar graphs [right-hand rule]
- On every graph that is outerplanar without the destination (e.g. non-outerplanar planar K_4)



Impossible:

- On some planar graphs
- Every non-planar graph
- Perfect resilience must hold on minors



Foerster et al. **On the Feasibility of Perfect Resilience with Local Fast Failover.** SIAM Symposium on Algorithmic Principles of Computer Systems (APOCS), 2021.

A Recent Survey

A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks
Marco Chiesa, Andrzej Kamisinski, Jacek Rak, Gabor Retvari, and Stefan Schmid.
IEEE Communications Surveys and Tutorials (**COMST**), 2021.

References

On the Price of Locality in Static Fast Rerouting
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
52nd IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Baltimore, Maryland, USA, June 2022.

The Hazard Value: A Quantitative Network Connectivity Measure Accounting for Failures
Pieter Cuijpers, Stefan Schmid, Nicolas Schnepf, and Jiri Srba.
52nd IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Baltimore, Maryland, USA, June 2022.

On the Feasibility of Perfect Resilience with Local Fast Failover
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
SIAM Symposium on Algorithmic Principles of Computer Systems (**APOCS**), Alexandria, Virginia, USA, January 2021.

Brief Announcement: What Can(not) Be Perfectly Rerouted Locally
Klaus-Tycho Foerster, Juho Hirvonen, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
International Symposium on Distributed Computing (**DISC**), Freiburg, Germany, October 2020.

Improved Fast Rerouting Using Postprocessing
Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
IEEE Transactions on Dependable and Secure Computing (**TDSC**), 2020.

Resilient Capacity-Aware Routing
Stefan Schmid, Nicolas Schnepf and Jiri Srba.
27th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (**TACAS**), Virtual Conference, March 2021.

AalWiNes: A Fast and Quantitative What-If Analysis Tool for MPLS Networks
Peter Gjøl Jensen, Morten Konggaard, Dan Kristiansen, Stefan Schmid, Bernhard Clemens Schrenk, and Jiri Srba.
16th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Barcelona, Spain, December 2020.

P-Rex: Fast Verification of MPLS Networks with Multiple Link Failures
Jesper Stenbjerg Jensen, Troels Beck Krogh, Jonas Sand Madsen, Stefan Schmid, Jiri Srba, and Marc Tom Thorgersen.
14th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Heraklion/Crete, Greece, December 2018.

Polynomial-Time What-If Analysis for Prefix-Manipulating MPLS Networks
Stefan Schmid and Jiri Srba.
37th IEEE Conference on Computer Communications (**INFOCOM**), Honolulu, Hawaii, USA, April 2018.

More References

[Randomized Local Fast Rerouting for Datacenter Networks with Almost Optimal Congestion]
Gregor Bankhamer, Robert Elsässer, and Stefan Schmid..
International Symposium on Distributed Computing (**DISC**), Freiburg, Germany, October 2021.

[Bonsai: Efficient Fast Failover Routing Using Small Arborescences]
Klaus-Tycho Foerster, Andrzej Kamisinski, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.
49th IEEE/IFIP International Conference on Dependable Systems and Networks (**DSN**), Portland, Oregon, USA, June 2019.

[CASA: Congestion and Stretch Aware Static Fast Rerouting]
Klaus-Tycho Foerster, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan
38th IEEE Conference on Computer Communications (**INFOCOM**), Paris, France, April 2019.

[Load-Optimal Local Fast Rerouting for Dense Networks]
Michael Borokhovich, Yvonne-Anne Pignolet, Gilles Tredan, and Stefan Schmid.
IEEE/ACM Transactions on Networking (**TON**), 2018.

[PURR: A Primitive for Reconfigurable Fast Reroute]
Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisinski, Georgios Nikolaidis, and Stefan Schmid.
15th ACM International Conference on emerging Networking EXperiments and Technologies (**CoNEXT**), Orlando, Florida, USA, December 2019.
Artefact Evaluation: Available, Functional, Reusable.

[On the Resiliency of Static Forwarding Tables]
In IEEE/ACM Transactions on Networking (**ToN**), 2017
M. Chiesa, I. Nikolaevskiy, S. Mitrovic, A. Gurtov, A. Madry, M. Schapira, S. Shenker

Questions?