The algorithmic challenges of local fast re-routing

Stefan Schmid (TU Berlin)
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

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)

Remote Controller

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

Minister of Education  VS  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.)

**In Data Plane**
- Static forwarding table
- Rules pre-installed *before* failures are known
Approaches for Failover

**In Control Plane**
- Slow but "globally informed".
- Distributed recomputation of shortest paths ("re-convergence")
  - Centralized recomputation of paths (SDN)
  - **Link-reversal** algorithms (e.g., Gafni et al.)

**In Data Plane**
- Fast but "local knowledge".
  - Static forwarding table
  - Rules pre-installed **before** failures are known
The FRR Problem

Phase 1: Rule installation
The FRR Problem

Phase 1: Rule installation

if x fwd to y
The FRR Problem

Phase 1: Rule installation

Phase 2: Failures and routing

if x fwd to y
The FRR Problem

Phase 1: Rule installation

Phase 2: Failures and routing

if x fwd to y

Without coordination!
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.
```

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.

Does not see 2nd failure...

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.

Credits: Klaus-Tycho Förster
What information is *locally* available in a switch for handling a packet?
Locally Available Information:
The Forwarding Table: Match -> Action

Credits: Marco Chiesa
Locally Available Information: The Packet Header
Locally Available Information: The Import of the Received Packet

Credits: Marco Chiesa
Locally Available Information: The Outgoing Port Depends on Failed Links
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?
If we partition the network, there is not much to do
The connectivity $k$ of a network $N$: the minimum number of link deletions that partitions $N$

The connectivity of this network is \textit{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 underlying 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 underlying network is *physically connected*.

Can this be achieved? Assume undirected link failures.
Spectrum of Models

Recall our switch model:

Achievable resilience depends on **what can be matched**:

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
</tr>
</thead>
</table>

Credits: Marco Chiesa
Spectrum of Models

Recall our switch model:

Achievable resilience depends on

Can carry global information, but often undesirable.

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
</tr>
</thead>
</table>

Credits: Marco Chiesa
Per-destination routing cannot cope with even one link failure

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
<th>Resiliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0</td>
</tr>
</tbody>
</table>

Without matching inport: sends back – *loop*!
Can we achieve $k-1$ resiliency in $k$-connected graph here?

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
<th>Resiliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>
Can we achieve $k - 1$ resiliency in $k$-connected graph here?

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
<th>Resiliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td>$X$</td>
<td></td>
<td>Yes</td>
</tr>
</tbody>
</table>

$k$ 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?

<table>
<thead>
<tr>
<th>Per-destination</th>
<th>Per source</th>
<th>Incoming port</th>
<th>Probabilistic forwarding</th>
<th>Packet header rewriting</th>
<th>Resiliency</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td>?</td>
</tr>
</tbody>
</table>

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

25
Ideal Resilience: Example 2-dim Torus?
Ideal Resilience: Example 2-dim Torus?

k=4 connected: tolerate 3 failures?
Idea: Decomposition into Hamilton Cycles

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

- Can route in both directions:
  - 4-arc-disjoint HC sets
  - 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!

Idea: Decomposition into Hamilton Cycles

1st Hamilton cycle
• 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!

Idea: Decomposition into Hamilton Cycles
Idea: Decomposition into Hamilton Cycles

- Decompose torus into 2-edge-disjoint Hamilton Cycles (HC)
- Can route in both directions: \textit{4-arc-disjoint} HCs
Idea: Decomposition into Hamilton Cycles

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

3-resilient routing to destination \(d\):
- go along \textit{1st directed HC}, if hit failure, reverse direction
- if again failure switch to \textit{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?

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

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

A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 1972]
A $k$-connected network contains $k$ arc-disjoint spanning arborescences [Edmonds, 1972]
A k-connected network contains k arc-disjoint spanning arborescences [Edmonds, 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]
A $k$-connected network contains $k$ arc-disjoint spanning arborescences [Edmonds, 1972]
General technique: routing along the same tree
When a failed link is hit...
... how do we choose the next arborescence?
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

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

\[1 \quad 2 \quad 3 \quad 4\]

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
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...
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
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
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
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 arborescence
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?
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 underlying 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
• 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).

Considered node 1 will not see any local failures.

So we must fix a permutation for node 1.
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.
Impotence: On Planar Graphs

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

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

For node 1:
5\rightarrow2 \text{ implies } (5,2,3,4) \ (b) \\
5\rightarrow3 \text{ implies } (5,3,4,2) \ (a) \\
5\rightarrow4 \text{ implies } (5,4,2,3) \ (c) \\
(5,2,4,3) \ (a)

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.

Arriving on inport 5, forwarded to 4.
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 $G \rightarrow$ 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 $G \rightarrow$ 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

A Recent Survey

**A Survey of Fast-Recovery Mechanisms in Packet-Switched Networks**
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.

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 Kamiński, Yvonne-Anne Pignolet, Stefan Schmid, and Gilles Tredan.

Resilient Capacity-Aware Routing
Stefan Schmid, Nicolas Schnepf and Jiri Srba.

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
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.
Randomized Local Fast Rerouting for Datacenter Networks with Almost Optimal Congestion
Gregor Bankhammer, 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.

Load-Optimal Local Fast Rerouting for Dense Networks
Michael Borokhovich, Yvonne-Anne Pignolet, Gilles Tredan, and Stefan Schmid.

PURR: A Primitive for Reconfigurable Fast Reroute
Marco Chiesa, Roshan Sedar, Gianni Antichi, Michael Borokhovich, Andrzej Kamisinski, Georgios Nikolaidis, and Stefan Schmid.
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?