On the Resiliency of Randomized Routing Against Multiple Edge Failures

Slobodan Mitrović (EPFL)

Joint work with M. Chiesa, A. Gurtov, A. Mądry, I. Nikolaevskiy, M. Schapira, and S. Shenker
Network routing

Given:
• a network $G = (V, E)$
• a target vertex $d$ of $V$

Goal: deliver a packet from $v$ to $d$
Network routing – Many different settings

1. per-destination
2. per-incoming-port
3. per-source-destination
4. packet-header rewriting
5. packet duplication
6. dynamic
7. ...

Two important properties:
- being resilient, and
- fast computation (static and local routing scheme)
Network routing – The problem we study

**Given:**
- a network \( G = (V, E) \), a target vertex \( d \)
- a parameter \( c \)

**Goal:** Find a per-destination static routing scheme that delivers a packet from any source \( s \) to \( d \)

subject to: at most \( c \) links of \( G \) are failed

with property: routing is local (no packet-header rewriting; no broadcasting)

We say such scheme is \( c \)-resilient.
Local routing decisions

• Each vertex $v$ has a list for each incoming link
• For incoming link from $w$, table provides a permutation of outgoing links
• The routing is continued through first non-failed link in the list

<table>
<thead>
<tr>
<th>v:</th>
<th>x:</th>
<th>y:</th>
<th>z:</th>
<th>t:</th>
</tr>
</thead>
<tbody>
<tr>
<td>x:</td>
<td>y:</td>
<td>z:</td>
<td>t:</td>
<td></td>
</tr>
<tr>
<td>y:</td>
<td>t:</td>
<td>x:</td>
<td>w:</td>
<td>z:</td>
</tr>
<tr>
<td>w:</td>
<td>x:</td>
<td>y:</td>
<td>z:</td>
<td>t:</td>
</tr>
<tr>
<td>z:</td>
<td>t:</td>
<td>z:</td>
<td>x:</td>
<td></td>
</tr>
<tr>
<td>t:</td>
<td>w:</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

s:

start: u

u:

s: y

y: z

y:

u: d u

z:

u: d

G
We need $k$-connectivity

$G_1 \cdots G_2$

$Necessary$ $condition: \ c\text{-resilience requires } k > c.$
Big challenge: Is $k > c$ sufficient too?

(from now, assume $c < k$)
Attempt 1 – Route along edge-disjoint paths

• [Menger's theorem] Between any u and v of G, there are at least k-edge disjoint paths.
Attempt 1 – Route along edge-disjoint paths

• *[Menger's theorem]* Between any u and v of G, there are at least k-edge disjoint paths.

1. Find path packing
2. Route along paths
3. On failed edge, retreat back to s, and choose the next path

• But, this is not satisfactory, because the table is source-d based.
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (*circular routing*)

[EGR-INFocom14]
[CNMPGSS-INFocom16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (circular routing)

[EGR-INFOCOM14]
[CNMPGSS-INFOCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (*circular routing*)

[EGR-INFCOM14]
[CNMPGSS-INFCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (circular routing)

[EGR-INFOCOM14]
[CNMPGSS-INFOCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence ($circular$ $routing$)

[EGR-INFOCOM14]
[CNMPGSS-INFOCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (circular routing)

[EGR-INFOCOM14]
[CNMPGSS-INFOCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (*circular routing*)

[EGR-INFOCOM14]
[CNMPGSS-INFOCOM16]
Attempt 2 – Packing arborescences


1. Find arborescences packing
2. Route along arborescences
3. On failed edge, choose the next available arborescence (circular routing)

[EGR-INFocom14]
[CNM-PGSS-INFocom16]
Arborescences give $k/2$ resilience

- Find an arborescence packing
- Order the arborescences
- Route along arborescences; on failed edge route along the next arborescence in the ordering

Each failure can affect two arborescences. The example is tight (think of a cycle).

Can we do better?
Our result: We settle this challenge and, given a $k$-connected graph, provide a randomized algorithm that is $(k-1)$-resilient.

(Naive use of randomization would be to just take a random walk. But this would be very inefficient both in amount of randomness used and the expected length of the routing paths.)
Two types of failed edges

- \{t, v\} edge is shared
- edges \{z, u\} and \{u, d\} are non-shared
Non-shared failed edges are not a problem

One failed edge destroys at most one arborescence

As there are at most k-1 failed edges, at least one arborescence has no failed arc.
How to "recycle" failed edges? Bounce!

- Route from $u$ along blue.
- If $(u, v)$ failed, no blue $u$-$d$ path, but...
- $(u, v)$ is shared and there is a green $u$-$d$ path!
- So, bounce at $u$. 
How to "recycle" failed edges? Bounce!

- Route from $u$ along blue.
- If $(u, v)$ failed, no blue $u$-$d$ path, but ...
- $(u, v)$ is shared and there is a green $u$-$d$ path!
- So, bounce at $u$. 

**Diagram:**

- Graph $G$ with nodes $u$, $v$, $d$, and $z$.
- Failed edge marked with a red cross.
- Blue, green, and orange arrows showing different paths.
- Order of colors: blue, orange, green.
How to "recycle" failed edges? Bounce!

- Route from $u$ along blue.
- If $(u, v)$ failed, no blue $u$-$d$ path, but ...
- $(u, v)$ is shared and there is a green $u$-$d$ path!
- So, bounce at $u$. 
Bouncing every time does not work

1. The packet is on blue at u.
Bouncing every time does not work

1. The packet is on blue at $u$.
2. The packet is on blue at $v$. 
Bouncing every time does not work

1. The packet is on blue at u.
2. The packet is on blue at v.
Bouncing every time does not work

G

1. The packet is on blue at u.
2. The packet is on blue at v.
4. The packet is on green at v.
Bouncing every time does not work

1. The packet is on **blue** at \( u \).
2. The packet is on **blue** at \( v \).
4. The packet is on **green** at \( v \).
5. The packet is on **green** at \( u \).
Bouncing every time does not work

1. The packet is on blue at u.
2. The packet is on blue at v.
4. The packet is on green at v.
5. The packet is on green at u.
Bouncing every time does not work

1. The packet is on blue at u.
2. The packet is on blue at v.
4. The packet is on green at v.
5. The packet is on green at u.
7. The packet is on blue at u.

LOOP!
Well-bouncing arcs and good arborescences

- An arc is **well-bouncing arcs** if bouncing on it takes a packet to $d$ without any interruption. (no more loops)
- An arborescence is **good** if its every failed arc is well-bouncing.

$(u, z)$ and $(v, t)$ are not well-bouncing

$(z, u)$ and $(t, v)$ are well bouncing
Well-bouncing arcs and good arborescences

• An arc is *well-bouncing arcs* if bouncing on it takes a packet to \( d \) without any interruption. (*no more loops*)

• An arborescence is *good* if its every failed arc is well-bouncing.

We can show that such good arborescence **always exists**!
Goal now: identify good arborescence

When to bounce?
If we know the given arborescence is good, we can bounce.

• Route along arborescence B
• If a packet hits a failed link:
  • If B is not good, route circularly
  • Otherwise, bounce

Too good to be true ... how do we know if an arborescence is good? We don’t. We guess!
Goal: routing on a good arborescence

- Route along arborescence B
- If a packet hits a failed link make a guess

With prob $p$

$(p$ is our likelihood that B is good$)$

With prob $1-p$
Our main result

**Theorem:**
Let $G$ be a $k$-connected graph, and $A$ be a decomposition of $G$ into $k$ arc-disjoint arborescences rooted at $d$. Assume that there are at most $f$ many failed edges. Then, $A$ contains at least $k-f$ good arborescences.

(for details see our paper)
Efficiency of our algorithm

- $(k-1)$-resilient routing scheme using randomness.

- Number of failed links a packet hits is $\Theta\left(\frac{k}{k-f}\right)$ by choosing appropriate $p$. (more details in the paper)

- If $f < (1 - \varepsilon)k$, for constant $\varepsilon < 1$, the packet hits only a constant number of failed links in expectation.
Open problems

• Can we devise deterministic \((k-1)\)-resilient routing tables?

• Can we slightly alter the given network and improve the resiliency?

• Can we devise even a randomized \((k-1)\)-vertex-resilient routing scheme, if the network is \(k\)-vertex-connected?
Meta-graph

\[ G \]

- \( f = \# \) of failed links

- \( H \)
  - constructed based on failed links only.
  - has exactly \( k \) vertices and \( f \) edges.
Meta-graph – connected components

Lemma: H contains at least \( k - f \) connected components which are trees.

Proof (sketch). Start with the empty graph on \( k \) vertices, and add \( f \) edges. Each edge reduces the number of trees by one at most.
A tree-component of the meta-graph

In every edge corresponds to a shared link.

$(u, v)$ is in $A$, and $(v, u)$ is in $B$

In total, $T$ has at least $|V(T)|$ green arcs and thus less than $|V(T)|$ red arcs.

Every arborescence has a well-bouncing arc, i.e. every vertex of $H$ has a green incoming arc.

There is a vertex of $T$ having only green outgoing arcs.
Bouncing and meta-graph

• The meta-graph has at least $k-f$ tree-connected components.
• Each tree-connected component has at least one vertex corresponding to a (good) arborescence on which any bouncing takes a packet to $d$.
• There are at least $k-f$ good arborescences.
Well-bouncing arcs

- $(u, v)$ and $(w, z)$ are well-bouncing arcs, as bouncing on them takes a packet to $d$ without any interruption.

- **Observation:** Each arborescence, having a failed arc, has at least one arc $(a, b)$ such that $(b, a)$ is well-bouncing. (e.g. the one closest to $d$).
Well-bouncing arcs – an example

\[(u, v)\] is the only arc which is not well-bouncing
A good arborescence – an example

Red and blue are good arborescences.
Should we do attempt 3 then – Random walks?

• Reaches $d$ as long as there is a path, even if $k \leq c$!
• But, can take can take $\Theta(n \cdot m)$ steps – too slow!
What is the resilience of circular routing?

ordering:

1. The packet is on blue at u.
What is the resilience of circular routing?

ordering:
1. The packet is on blue at u.
2. Link failed. Switch to yellow.
What is the resilience of circular routing?

1. The packet is on blue at u.
2. Link failed. Switch to yellow.
3. Link failed. Switch to red.
What is the resilience of circular routing?

ordering:

1. The packet is on blue at u.
2. Link failed. Switch to yellow.
3. Link failed. Switch to red.
4. Move to v along red.
What is the resilience of circular routing?

ordering:
1. The packet is on blue at u.
2. Link failed. Switch to yellow.
3. Link failed. Switch to red.
4. Move to v along red.
5. Link failed. Switch to green.
What is the resilience of circular routing?

1. The packet is on blue at u.
2. Link failed. Switch to yellow.
3. Link failed. Switch to red.
4. Move to v along red.
5. Link failed. Switch to green.
6. Link has failed. Switch to blue.
What is the resilience of circular routing?

1. The packet is on blue at u.
2. Link failed. Switch to yellow.
3. Link failed. Switch to red.
4. Move to v along red.
5. Link failed. Switch to green.
6. Link has failed. Switch to blue.
7. Move to u along blue.

LOOP!
What is the resilience of circular routing?

It does not result in \((k-1)\)-resiliency. In fact, we can generalize this example to show it does not achieve better than \(k/2\)-resiliency. A simple counting argument provides only \((k/2-1)\)-resiliency. (every failed edges “kills” at most two arborescences)

Can we do better?
Well-bouncing arcs and good arborescences

- An arc is *well-bouncing arcs* if bouncing on it takes a packet to \( d \) without any interruption. (*no more loops*)

- An arborescence is *good* if its every failed arc is well-bouncing.

Both \((u, v)\) and \((v, u)\) are well-bouncing