Provable Data Plane Connectivity with Local Fast Failover

Introducing OpenFlow Graph Algorithms

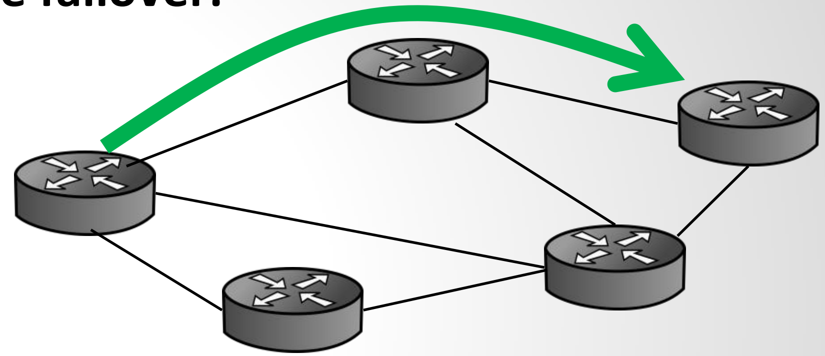
Michael Borokhovich (Ben Gurion Uni, Israel)
Liron Schiff (Tel Aviv Uni, Israel)
Stefan Schmid (TU Berlin & T-Labs, Germany)

Robust Routing Mechanisms

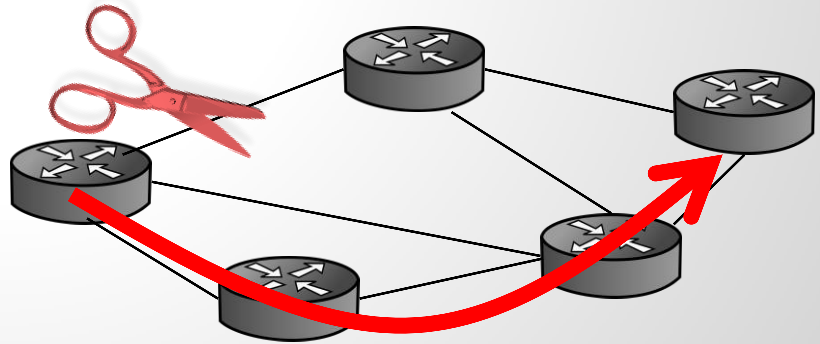
- **Link failures** today are not uncommon [1]

- Modern networks provide **robust routing mechanisms**
  - i.e., routing which reacts to failures
  - example: MPLS local and global path protection

Before failover:

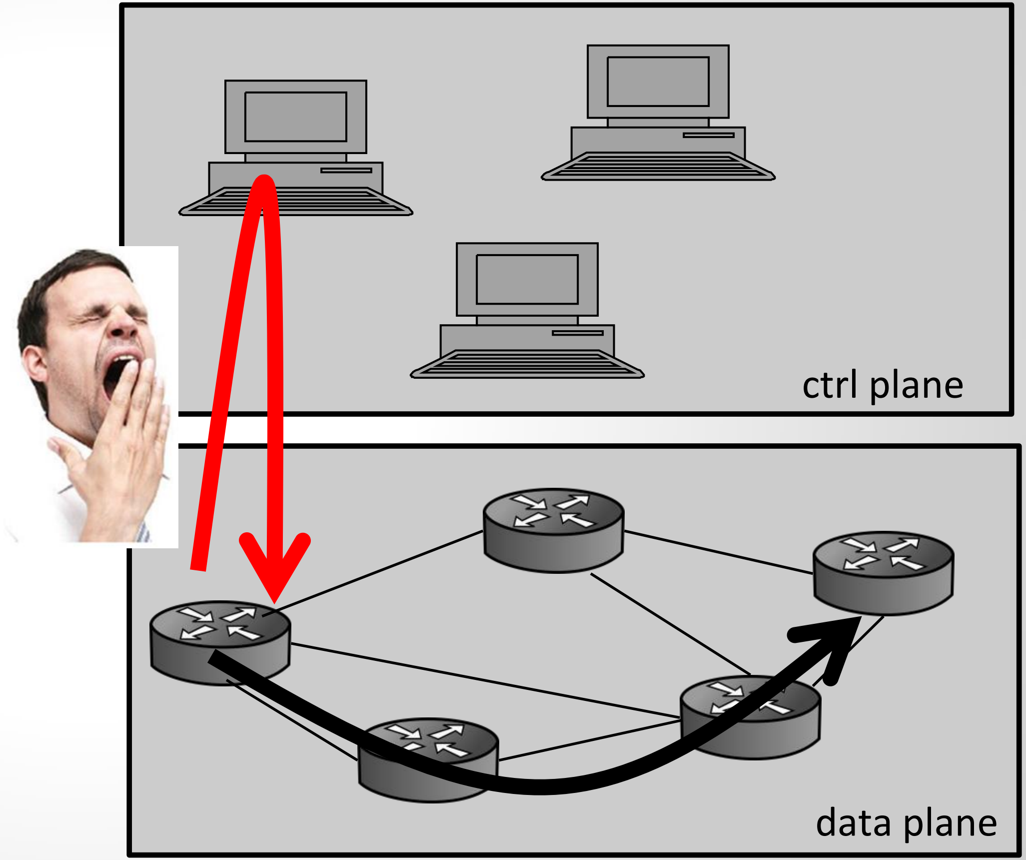


After failover:



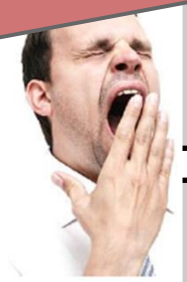
Fast In-band Failover

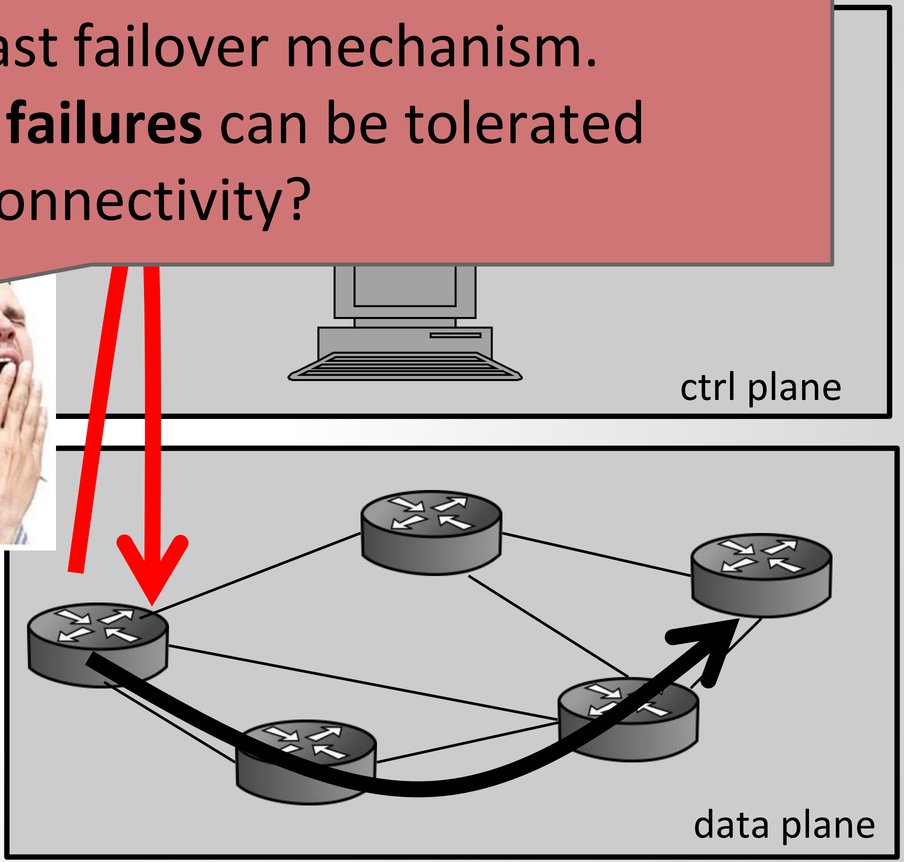
- Important that failover happens **fast = in-band**
  - Reaction time in control plane can be orders of magnitude slower [1]
- For this reason: **OpenFlow Local Fast Failover Mechanism**
  - Supports conditional forwarding rules (depend on the local state of the link: live or not?)
- Gives fast but local and perhaps “**suboptimal**” forwarding sets
  - Controller improves globally later…



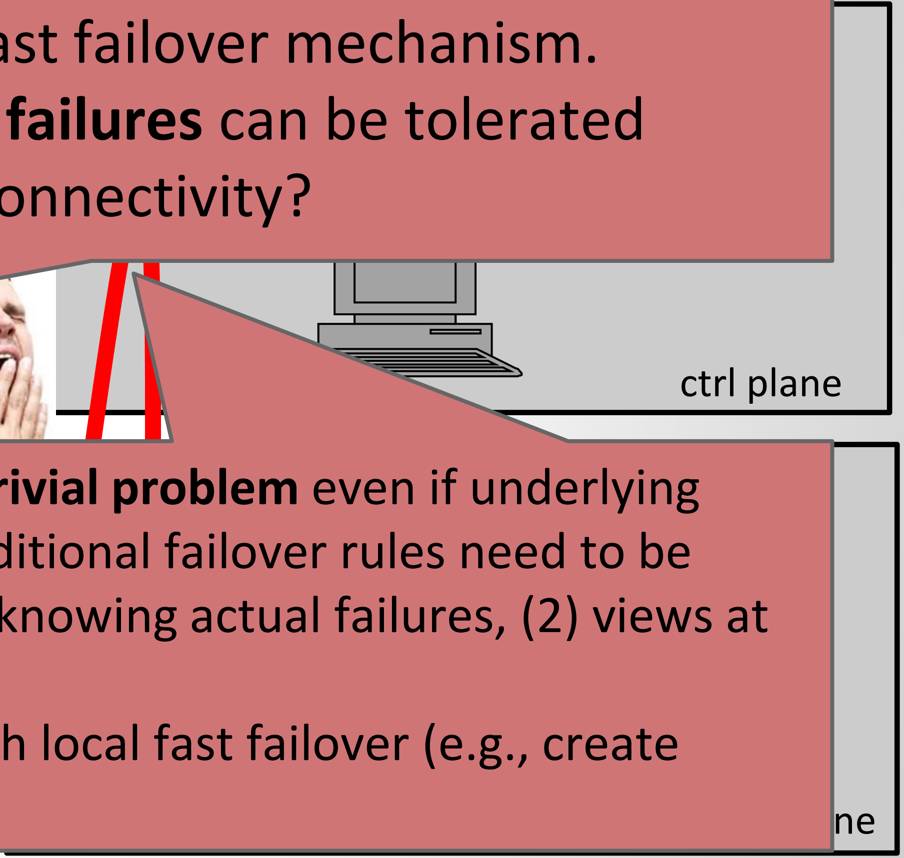
However, not much is known about how to ***use*** the OpenFlow fast failover mechanism.
E.g.: **How many failures** can be tolerated without losing connectivity?

- Important that failove
  fast = in-band
  - Reaction time in contr
    orders of magnitude s
- For this reason: **OpenFl**
  Fast Failover Mechanism
  - Supports conditional forwarding rules (depend on the local state of the link: live or not?)
- Gives fast but local and perhaps "**suboptimal**" forwarding sets
  - Controller improves globally later…





- Important that failove
  fast = in-band
  - Reaction time in contr
    orders of magnitude s
- For this reason: **OpenFlo** **Local**
  Fast Failover Mechanism
  - Sup
    (dep
    live
- Gives f
  “**subop**
  - Con



However, not much is known about how to ***use*** the OpenFlow fast failover mechanism.
E.g.: **How many failures** can be tolerated without losing connectivity?

How to use mechanism is a **non-trivial problem** even if underlying network stays connected: (1) conditional failover rules need to be allocated **ahead of time**, without knowing actual failures, (2) views at runtime are **inherently local**.
How not to **shoot in your foot** with local fast failover (e.g., create forwarding loops)?

Contribution: Very Robust Routing Possible with OpenFlow

Theorem: «Ideal» Forwarding Connectivity Possible

There exist algorithms which guarantee that packets always reach their destination, **independently of the number and locations** of failures, as long as the remaining network is connected.

Contribution: Very Robust Routing Possible with OpenFlow

Theorem: «Ideal» Forwarding Connectivity Possible

There exist algorithms which guarantee that packets always reach their destination, **independently of the number and locations** of failures, as long as the remaining network is connected.

Three algorithms:

- **Modulo**
- **Depth-First**
- **Breadth-First**

Essentially classic **graph algorithms** (routing, graph search) implemented **in OpenFlow**. Make use of **tagging** to equip packets with meta-information to avoid forwarding loops.

Contribution: Very Robust Routing Possible with OpenFlow

Theorem: «Ideal» Forwarding Connectivity Possible

The … always reach
the … **locations** of
fail… ed.

Analysis of their **complexity**: maximum stretch (route length compared to ideal route), number of tags, number of OpenFlow rules.

Three algorithms:

- **Modulo**
- **Depth-First**
- **Breadth-First**

Essentially classic **graph algorithms** (routing, graph search) implemented **in OpenFlow**. Make use of **tagging** to equip packets with meta-information to avoid forwarding loops.

Overview of Contributions

High-Level Algorithms

Algorithm 2 Algorithm DFS

Input: current node: v_i, input port: in, packet dest: d, packet failover global params: $pkt.start$, packet tag array: $\{pkt.v_j\}_{j\in[n]}$
Output: output

```
if pkt.start =
   out ← defa
   if out failed
      pkt.start
      pkt.v_i.pa
      out ← 1
else
   if pkt.v_i.cu
      pkt.v_i.pa
   out ← pkt.
   if out = Δ_
      out ← pk
      goto 19
   while out fai
      out ← out ·
      if out = Δ_i + 1 then
         out ← pkt.v_i.par
         goto 19

pkt.v_i.cur ← out
return out
```

Algorithm 1 Algorithm Mod

Input: current node: v_i, packet dest: d, packet tag array: $\{pkt.v_j\}_{j\in[n]}$
Output: output port: out

```
if no tag then {same as pkt.v_i = 0}
   out ← default_route(i, d)
else
   out ← (pkt.v_i mod Δ_i) + 1
   pkt.v_i ← out
   while out failed do
      out ← (pkt.v_i mod Δ_i) + 1
      pkt.v_i ← out
return out
```

Algorithm 3 Algorithm BFS

Input: current node: v_i, input port: in, packet dest: d, packet failover global params: $pkt.start$, packet tag array: $\{pkt.v_j\}_{j\in[n]}$
Output: output port: out

```
if pkt.start = 0 then
   i, d)
pkt.v_i.par = 0 then
d pkt.v_i.par ≠ in then
= pkt.v_i.par do
if out = 0 then
   out ← 1
   while out failed do
      out ← out + 1
      if out = Δ_i + 1 then
         Drop
   pkt.v_i.cur ← out
return out
```

Flow-Table Implementations

Flow Table $\Delta_i - 1$

Match: sb	Instructions
0	Gr $\Delta_i - 1$, Table Δ_i
1	Drop

Flow Table Δ_i

Match: sb	Instructions
0	Gr Δ_i,Table 0.2 (Send-Parent)
1	Drop

Table 5: DFS: Flow Tables of switch i.

Flow Table A (Start)

Match		Instructions
$pkt.start$	dst	
0	1	Gr 0.1, Table 1
0	2	Gr 0.2, Table 1
...	...	...
0	n	Gr 0.n, Table 1
*	*	Table B

Flow Table B

Match			Instructions
in	$pkt.v_i.cur$	$pkt.v_i.par$	
1	0	*	$pkt.v_i.par \leftarrow in$, Table 2
1	1	2	Table 3
2	2	3	Table 4
3	3	4	Table 5
...	...	...	...
$\Delta_i - 2$	$\Delta_i - 2$	$\Delta_i - 1$	Table Δ_i
$\Delta_i - 1$	$\Delta_i - 1$	Δ_i	Table C
*	0	*	$pkt.v_i.par \leftarrow in$, Table 1
1	1	*	Table 2
2	2	*	Table 3
3	3	*	Table 4
...	...	...	...
$\Delta_i - 1$	$\Delta_i - 1$	*	Table Δ_i
Δ_i	Δ_i	*	Table C
1	*	*	Fwd 1
2	*	*	Fwd 2

Complexity Analysis

THEOREM 1. MOD *ensures data plane connectivity when-*

Algorithm	Packet Memory	Message count	Rules space
Module	nlogd	Exp(n)	O(n*d)
DFS	nlogd	2|E|	O(n*d)
BFS	nlogd	2kn	O(n*d)
BFS*	k(logd+logn)	2kn	O(n*(d+k))

Related Work

- Borokhovich, OPODIS'13
- [1] Liu et al. NSDI'13
- Graph-search literature

Overview of Contributions

We expect that our algorithms scale up to 500-node networks (ignoring link capacities)
(e.g., using our NoviKit 250 switches, with 32MB flow table space and full support for extended match fields)

v-Table Implementations

```
if out = 0 then
    out ← 1
    while out failed do
        out ← out + 1
        if out = Δ_i + 1 then
            Drop
    pkt.v_i.cur ← out
return out
```

Flow Table Δ_i

Match sb	Instructions
0	Gr Δ_i,Table 0.2 (Send-Parent)
1	Drop

Table 5: DFS: Flow Tables of switch i.

Flow Table A (Start)

Match $pkt.start$	Match dst	Instructions
0	1	Gr 0.1, Table 1
0	2	Gr 0.2, Table 1
...	...	...
0	n	Gr 0.n, Table 1
*	*	Table B

Flow Table B

Match in	Match $pkt.v_i.cur$	Match $pkt.v_i.par$	Instructions
1	0	*	$pkt.v_i.par \leftarrow in$, Table 2
1	1	2	Table 3
2	2	3	Table 4
3	3	4	Table 5
...	...	...	...
$\Delta_i - 2$	$\Delta_i - 2$	$\Delta_i - 1$	Table Δ_i
$\Delta_i - 1$	$\Delta_i - 1$	Δ_i	Table C
*	0	*	$pkt.v_i.par \leftarrow in$, Table 1
1	1	*	Table 2
2	2	*	Table 3
3	3	*	Table 4
...	...	...	...
$\Delta_i - 1$	$\Delta_i - 1$	*	Table Δ_i
Δ_i	Δ_i	*	Table C
1	*	*	Fwd 1
2	*	*	Fwd 2

Complexity Analysis

THEOREM 1. MOD *ensures data plane connectivity when-*

Algorithm	Packet Memory	Message count	Rules space
Module	nlogd	Exp(n)	O(n*d)
DFS	nlogd	2|E|	O(n*d)
BFS	nlogd	2kn	O(n*d)
BFS*	k(logd+logn)	2kn	O(n*(d+k))

Related Work

- Borokhovich, OPODIS'13
- [1] Liu et al. NSDI'13
- Graph-search literature

Overview of Contributions

We expect that our algorithms scale up to 500-node networks (ignoring link capacities) (e.g., using our NoviKit 250 switches, with 32MB flow table space and full support for extended match fields)

Flow-Table Implementations

Inherent tradeoffs between robustness and network load of failover without tagging.

Same objective: ideal connectivity. But their link-reversal algorithms not applicable to OpenFlow: require dynamic state at router.

Complexity Analysis

Lower bounds with implications on optimality of our algorithms.

Algorithm		Rules space
Module		O(n*d)
DFS		O(n*d)
BFS		
BFS*		O(n*(d+k))

Related Work

- Borokhovich, OPODIS'13
- [1] Liu et al. NSDI'13
- Graph-search literature

Conclusion

- Fast failover: example of a function that should be kept in the data plane
- Our result shows that non-trivial functions can be computed in the OpenFlow data plane!
- Our algorithms: may serve in compilers for higher-level languages, e.g., FatTire