# How (Not) to Shoot in Your Foot with SDN Local Fast Failover

A Load-Connectivity Tradeoff

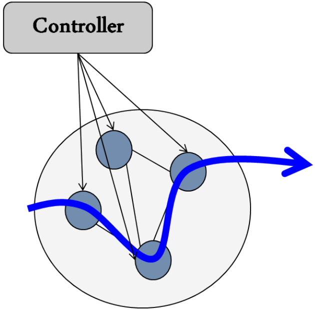
Michael Borokhovich, Stefan Schmid

Communication Systems Engineering, Ben-Gurion University, Israel

Internet Network Architectures, TU Berlin & T-Labs, Germany

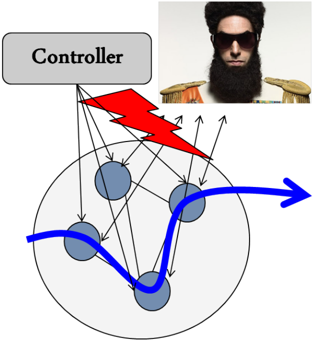
OPODIS 2013
Nice, France

# Motivation: SDN Local Fast Failover



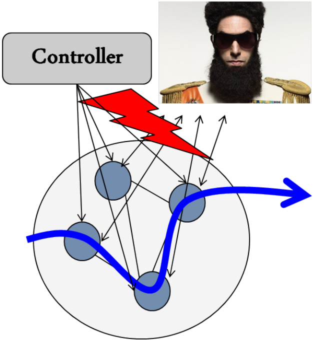
# Motivation: SDN Local Fast Failover

- Failures: a disadvantage of OpenFlow?
  - Indirection via controller (reactive control) an overhead?
  - Or even full disconnect from controller?



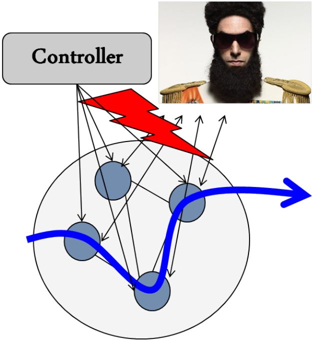
# Motivation: SDN Local Fast Failover

- Failures: a disadvantage of OpenFlow?
  - Indirection via controller (reactive control) an overhead?
  - Or even full disconnect from controller?
- Local fast failover
  - E.g., since OpenFlow 1.1 (but already MPLS,...)
  - Failover in data plane: given failed incident links, decide what to do with flow
  - **(header, failed links)** $\longrightarrow$ **(backup port)**
  - React quickly, controller can improve later!



# Motivation: SDN Local Fast Failover

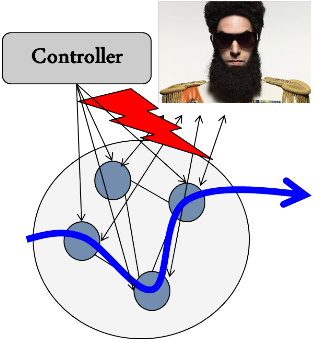
- Failures: a disadvantage of OpenFlow?
  - Indirection via controller (reactive control) an overhead?
  - Or even full disconnect from controller?
- Local fast failover
  - E.g., since OpenFlow 1.1 (but already MPLS,...)
  - Failover in data plane: given failed incident links, decide what to do with flow
  - **(header, failed links)** $\longrightarrow$ **(backup port)**
  - React quickly, controller can improve later!
- Threat: local failover may introduce loop or be inefficient in other ways (high load)



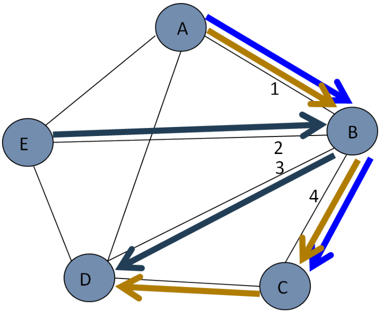
# Motivation: SDN Local Fast Failover

- Failures: a disadvantage of OpenFlow?
  - Indirection via controller (reactive control) an overhead?
  - Or even full disconnect from controller?
- Local fast failover
  - E.g., since OpenFlow 1.1 (but already MPLS,...)
  - Failover in data plane: given failed incident links, decide what to do with flow
  - **(header, failed links)** $\longrightarrow$ **(backup port)**
  - React quickly, controller can improve later!
- Threat: local failover may introduce loop or be inefficient in other ways (high load)

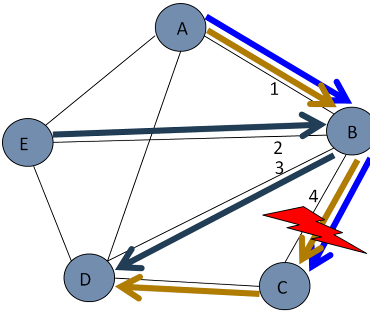
**How not to shoot in your foot?!**



# Model: General Failover Rules



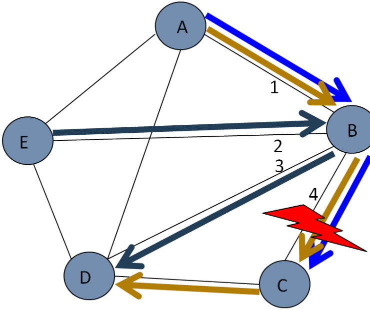
# Model: General Failover Rules




- if (B,C) fails:
  - fwd (A,C) to port 3
  - fwd (A,D) to port 2

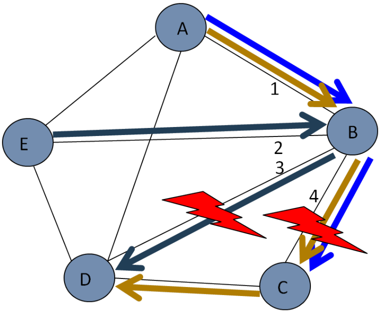
# Model: General Failover Rules

Flows can be treated individually!




- if (B,C) fails:
  - **fwd (A,C) to port 3**
  - **fwd (A,D) to port 2**

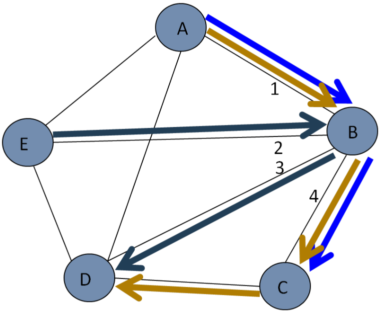
# Model: General Failover Rules




- if (B,C) fails:
  - fwd (A,C) to port 3
  - fwd (A,D) to port 2
- if (B,C) and (B,D) fail:
  - fwd (A,C) to port 2
  - fwd (A,D) to port 2
  - fwd (E,D) to port 1

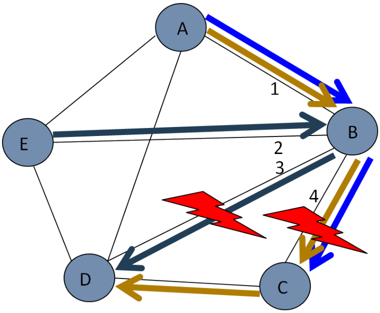
# Model: General Failover Rules

Depending on failure set,
(A,C) is forwarded differently!




- if (B,C) fails:
  - **fwd (A,C) to port 3**
  - fwd (A,D) to port 2
- if (B,C) and (B,D) fail:
  - **fwd (A,C) to port 2**
  - fwd (A,D) to port 2
  - fwd (E,D) to port 1

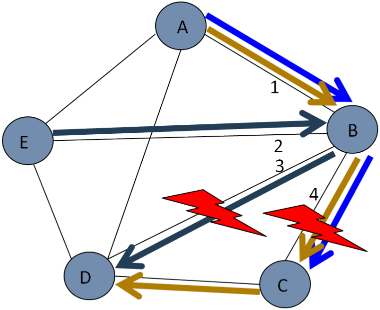
# Model: Destination-Based Failover Rules




- if (B,C) fails:
  - fwd (A,C) to port 3
  - fwd (A,D) to port 2
- if (B,C) and (B,D) fail:
  - fwd (A,C) to port 2
  - fwd (A,D) to port x
  - fwd (E,D) to port x

# Model: Destination-Based Failover Rules

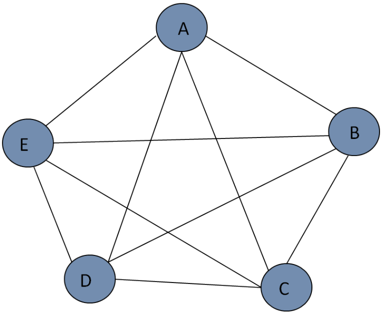
Same destination requires same forwarding port.




- if (B,C) fails:
  - fwd (A,C) to port 3
  - fwd (A,D) to port 2
- if (B,C) and (B,D) fail:
  - fwd (A,C) to port 2
  - fwd (A,D) to **port x**
  - fwd (E,D) to **port x**

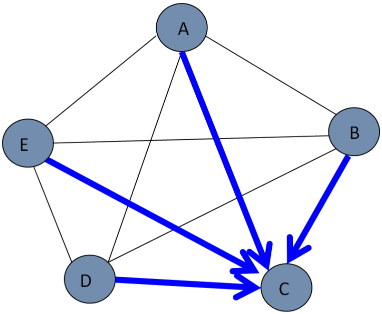
# Negative Result: You must shoot in your foot!

A simple example: full mesh (clique) & all-to-one communication



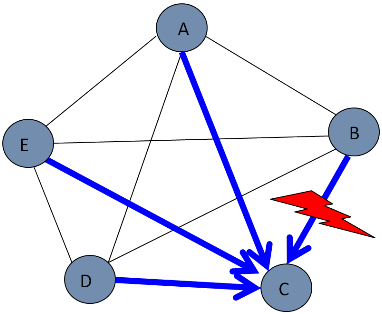
# Negative Result: You must shoot in your foot!

A simple example: full mesh (clique) & all-to-one communication



# Negative Result: You must shoot in your foot!

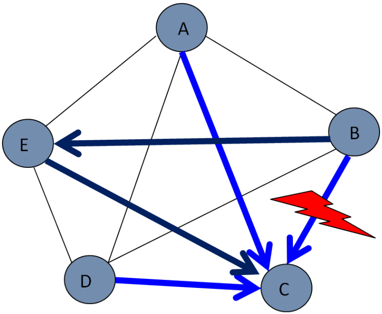
A simple example: full mesh (clique) & all-to-one communication




if *Event* then try other port (set of failures, per flow, ...)

# Negative Result: You must shoot in your foot!

A simple example: full mesh (clique) & all-to-one communication

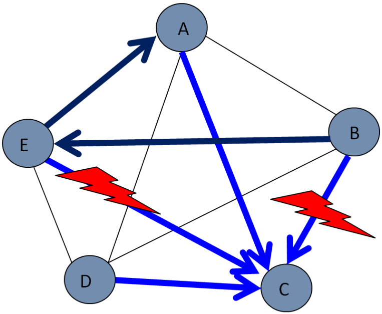



if *Event* then try other port (set of failures, per flow, ...)

# Negative Result: You must shoot in your foot!

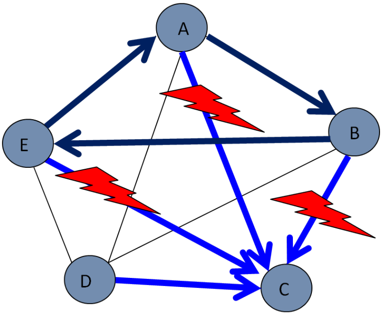
A simple example: full mesh (clique) & all-to-one communication

if *Event* then try other port (set of failures, per flow, ...)



# Negative Result: You must shoot in your foot!

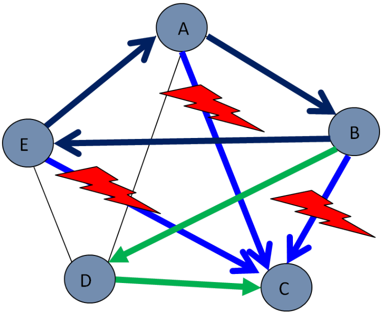
A simple example: full mesh (clique) & all-to-one communication




**Loop!**

# Negative Result: You must shoot in your foot!

A simple example: full mesh (clique) & all-to-one communication




**Loop!**

Unnecessary: Many paths left!
But do not know remote state...

# Negative Result: You must shoot in your foot!

How bad can it get?

How bad can it get?

**Theorem**

*No local failover scheme can tolerate $n-1$ or more link failures, even though the graph is still $n/2$-connected.*

# Negative Result: You must shoot in your foot!

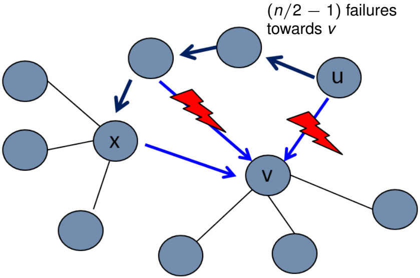
How bad can it get?

**Theorem**

*No local failover scheme can tolerate $n-1$ or more link failures, even though the graph is still $n/2$-connected.*

Proof idea:

- Fail any link which would directly lead to destination node $v$ until $(n/2-1)$ links failed



# Negative Result: You must shoot in your foot!

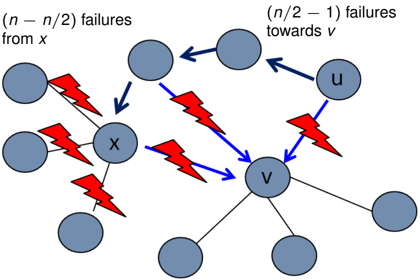
How bad can it get?

**Theorem**

*No local failover scheme can tolerate $n - 1$ or more link failures, even though the graph is still $n/2$-connected.*

Proof idea:

- Fail any link which would directly lead to destination node $v$ until $(n/2 - 1)$ links failed
- Fail links from $x$ to $(n - n/2)$ other nodes



# Negative Result: You must shoot in your foot!

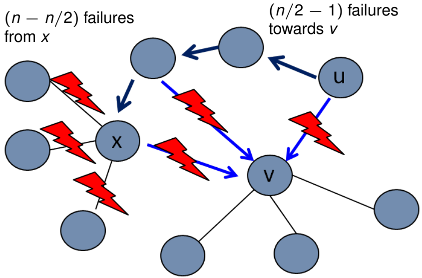
How bad can it get?

**Theorem**

*No local failover scheme can tolerate $n - 1$ or more link failures, even though the graph is still $n/2$-connected.*

Proof idea:

- Fail any link which would directly lead to destination node $v$ until $(n/2 - 1)$ links failed
- Fail links from $x$ to $(n - n/2)$ other nodes
- $x$ only has links to already visited nodes: **loop unavoidable!**
- But all nodes still have degree at least $n/2 - 1$ ($x$ and $v$ have the lowest)



# Negative Result: You must shoot in your foot!

Another consequence: high load!

# Negative Result: You must shoot in your foot!

Another consequence: high load!

**Theorem**

*For any local failover scheme, there exists a failure scenario which uses $\varphi$ failures and yields max link load of at least $\sqrt{\varphi}$, while the mincut is at least $n - \varphi - 1$.*

# Negative Result: You must shoot in your foot!

Another consequence: high load!

**Theorem**

*For any local failover scheme, there exists a failure scenario which uses $\varphi$ failures and yields max link load of at least $\sqrt{\varphi}$, while the mincut is at least $n - \varphi - 1$.*

If rules failover destination-based only, the load is much higher.

# Negative Result: You must shoot in your foot!

Another consequence: high load!

**Theorem**

*For any local failover scheme, there exists a failure scenario which uses $\varphi$ failures and yields max link load of at least $\sqrt{\varphi}$, while the mincut is at least $n - \varphi - 1$.*

If rules failover destination-based only, the load is much higher.

**Theorem**

*For any local **destination-based** failover scheme, there exists a failure scenario which uses $\varphi$ failures and yields max link load of at least $\varphi$, while the mincut is still at least $n - \varphi - 1$.*

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least* $\sqrt{\varphi}$.

# Negative Result: You must shoot in your foot!

**Theorem**

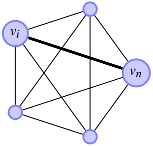
*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*

- Consider the following failover paths:

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*



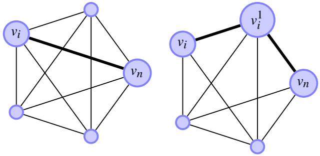

- Consider the following failover paths:

$(v_i \rightarrow v_n)$, now let's fail $(v_i, v_n)$

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*




- Consider the following failover paths:

$(v_i \to v_n)$, now let's fail $(v_i, v_n)$

$(v_i \to \cdots \to v_i^1 \to v_n)$, now let's fail $(v_i^1, v_n)$

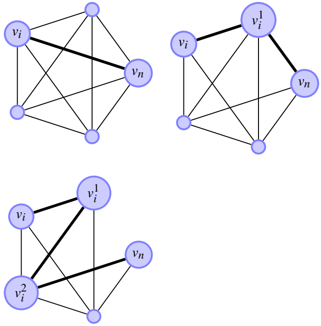
# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least* $\sqrt{\varphi}$.

- Consider the following failover paths:

$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$



# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*

- Consider the following failover paths:

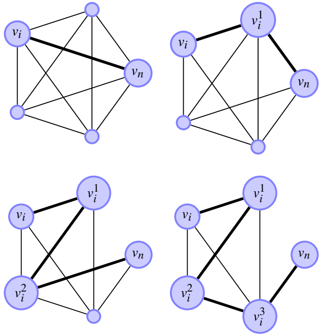
$(v_i \rightarrow v_n)$, now let's fail $(v_i, v_n)$

$(v_i \rightarrow \cdots \rightarrow v_i^1 \rightarrow v_n)$, now let's fail $(v_i^1, v_n)$

$(v_i \rightarrow \cdots \rightarrow v_i^1 \rightarrow \cdots \rightarrow v_i^2 \rightarrow v_n)$,

$\ldots$

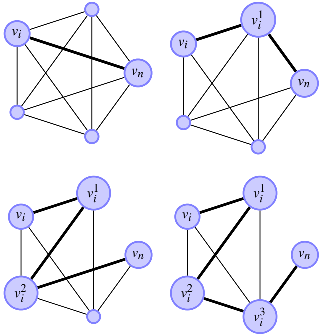
$(v_i \rightarrow \cdots \rightarrow v_i^1 \rightarrow \cdots \rightarrow v_i^2 \rightarrow \cdots \rightarrow v_i^{\varphi} \rightarrow v_n)$.



# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*




- Consider the following failover paths:

$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$
$$\ldots$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n).$$

- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.

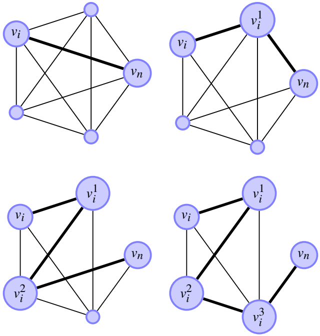
# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*

- Consider the following failover paths:

$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$
$$\ldots$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n).$$

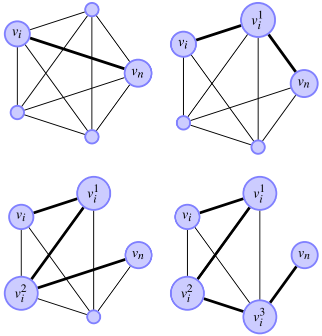



- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.
- Consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{\varphi}}\}$ (set of possible last hops on the path to $v_n$).

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*




- Consider the following failover paths:

$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$
$$\ldots$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n).$$

- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.
- Consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{\varphi}}\}$ (set of possible last hops on the path to $v_n$).
- Consider a multiset of $\left|\bigcup_i A_i\right| = (n-1)(\sqrt{\varphi}+1)$ nodes.

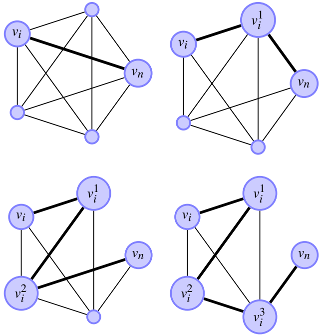
# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*

- Consider the following failover paths:

$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$
$$\ldots$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n).$$

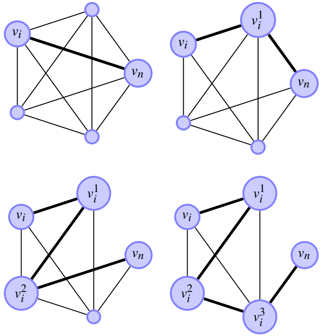



- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.
- Consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{\varphi}}\}$ (set of possible last hops on the path to $v_n$).
- Consider a multiset of $\left|\bigcup_i A_i\right| = (n-1)(\sqrt{\varphi}+1)$ nodes.
- By a counting argument, there exists a node $x \in \bigcup_i A_i$ which appears in at least $\sqrt{\varphi}$ sets $A_i$.

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*




- Consider the following failover paths:

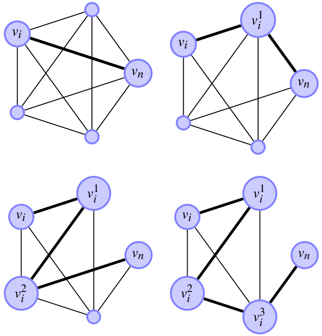
$$(v_i \to v_n), \quad \text{now let's fail } (v_i, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to v_n), \quad \text{now let's fail } (v_i^1, v_n)$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n),$$
$$\ldots$$
$$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n).$$

- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.
- Consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{\varphi}}\}$ (set of possible last hops on the path to $v_n$).
- Consider a multiset of $\left|\bigcup_i A_i\right| = (n-1)(\sqrt{\varphi}+1)$ nodes.
- By a counting argument, there exists a node $x \in \bigcup_i A_i$ which appears in at least $\sqrt{\varphi}$ sets $A_i$.
- For each such $A_i$, the adversary can route $v_i \to v_n$ via $x$ by failing links to $v_n$.

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local failover scheme, there exists a failure scenario which yields max link load of at least $\sqrt{\varphi}$.*




- Consider the following failover paths:

$(v_i \to v_n)$, now let's fail $(v_i, v_n)$

$(v_i \to \cdots \to v_i^1 \to v_n)$, now let's fail $(v_i^1, v_n)$

$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to v_n)$,

$\ldots$

$(v_i \to \cdots \to v_i^1 \to \cdots \to v_i^2 \to \cdots \to v_i^{\varphi} \to v_n)$.

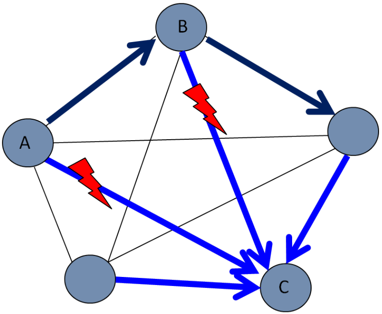
- The last hop $v_i^j$ ($j \in [1, \ldots, \varphi]$) is unique for every path.
- Consider the set $A_i = \{v_i, v_i^1, \ldots, v_i^{\sqrt{\varphi}}\}$ (set of possible last hops on the path to $v_n$).
- Consider a multiset of $\left|\bigcup_i A_i\right| = (n-1)(\sqrt{\varphi}+1)$ nodes.
- By a counting argument, there exists a node $x \in \bigcup_i A_i$ which appears in at least $\sqrt{\varphi}$ sets $A_i$.
- For each such $A_i$, the adversary can route $v_i \to v_n$ via $x$ by failing links to $v_n$.
- The adversary will fail $\sqrt{\varphi} \times \sqrt{\varphi} = \varphi$ links incident to $v_n$; load of link $(x, v_n)$ becomes $\sqrt{\varphi}$.

# Negative Result: You must shoot in your foot!

**Theorem**

*For any local **destination-based** failover scheme, there exists a failure scenario which yields max link load of at least $\varphi$.*

Why worse for destination-based? Intuition:




At B, flow (A,C) gets combined with flow (B,C) and never splits again. Etc.!

# Positive Result: Make the best out of the situation!

A general failover scheme:

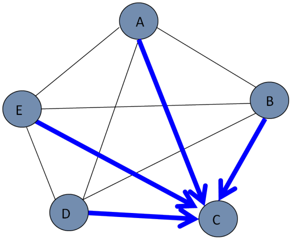
$\delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2}$

$\ldots$

$\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2}$

$\ldots$

$\delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2}$



# Positive Result: Make the best out of the situation!

A general failover scheme:

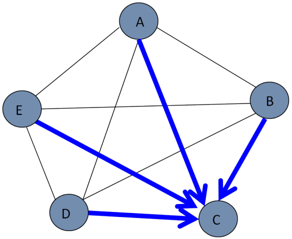
$\delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2}$

$\ldots$

$\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2}$

$\ldots$

$\delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2}$




- Matrix $\delta_{i,j}$: if node $v_i$ cannot reach destination directly, try node $\delta_{i,1}$; if not reachable either, try node $\delta_{i,2}$, ...

# Positive Result: Make the best out of the situation!

A general failover scheme:

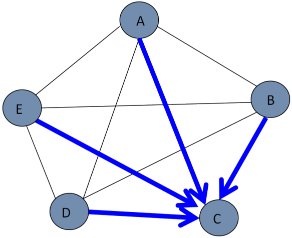
$\delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2}$

$\ldots$

$\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2}$

$\ldots$

$\delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2}$




- Matrix $\delta_{i,j}$: if node $v_i$ cannot reach destination directly, try node $\delta_{i,1}$; if not reachable either, try node $\delta_{i,2}$, ...
- Choosing random permutations:

**Theorem**

***Random*** *Failover Scheme (RFS) can tolerate $\varphi$ failures ($0 < \varphi < n$) with load no more than $\sqrt{\varphi} \log n$.*

# Positive Result: Make the best out of the situation!

Can also be achieved **deterministically**, as long as number of failures bounded by log $n$:

$$\delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2}$$

$$\ldots$$

$$\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2}$$

$$\ldots$$

$$\delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2}$$

$$\xrightarrow[\delta_{i,j} = (i-1) + 2^{j-1}]{}$$

$$1, 2, 4, 8, \ldots, \left(0 + 2^{\lfloor \log n \rfloor}\right) \mod n$$

$$2, 3, 5, 9, \ldots, \left(1 + 2^{\lfloor \log n \rfloor}\right) \mod n$$

$$3, 4, 6, 10 \ldots, \left(2 + 2^{\lfloor \log n \rfloor}\right) \mod n$$

$$\ldots$$

# Positive Result: Make the best out of the situation!

Can also be achieved **deterministically**, as long as number of failures bounded by log $n$:

$$\delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2}$$
$$\ldots$$
$$\delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2}$$
$$\ldots$$
$$\delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2}$$

$$\xrightarrow[\delta_{i,j} = (i-1) + 2^{j-1}]{}$$

$$1, 2, 4, 8, \ldots, \left(0 + 2^{\lfloor \log n \rfloor}\right) \mod n$$
$$2, 3, 5, 9, \ldots, \left(1 + 2^{\lfloor \log n \rfloor}\right) \mod n$$
$$3, 4, 6, 10 \ldots, \left(2 + 2^{\lfloor \log n \rfloor}\right) \mod n$$
$$\ldots$$

**Theorem**

***Deterministic*** *Failover Scheme (DFS) can tolerate $\varphi$ failures ($0 < \varphi < \log n$) with load no more than $\sqrt{\varphi}$.*

# Positive Result: Make the best out of the situation!

Can also be achieved **deterministically**, as long as number of failures bounded by $\log n$:

$$\begin{array}{l} \delta_{1,1}, \delta_{1,2}, \ldots, \delta_{1,n-2} \\ \ldots \\ \delta_{i,1}, \delta_{i,2}, \ldots, \delta_{i,n-2} \\ \ldots \\ \delta_{n-1,1}, \delta_{n-1,2}, \ldots, \delta_{n-1,n-2} \end{array} \quad \xrightarrow[\delta_{i,j} = (i-1) + 2^{j-1}]{} \quad \begin{array}{l} 1, 2, 4, 8, \ldots, \left(0 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ 2, 3, 5, 9, \ldots, \left(1 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ 3, 4, 6, 10 \ldots, \left(2 + 2^{\lfloor \log n \rfloor}\right) \mod n \\ \ldots \end{array}$$

**Theorem**

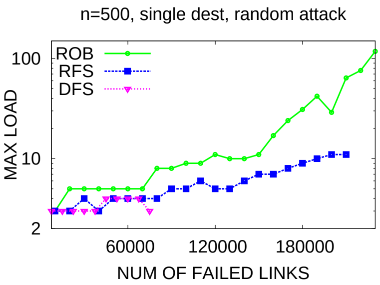
***Deterministic** Failover Scheme (DFS) can tolerate $\varphi$ failures ($0 < \varphi < \log n$) with load no more than $\sqrt{\varphi}$.*

Proof idea:

- There are no repetitions in the matrix columns. Thus any node appears exactly once at the first position, exactly once at the second, and so on...
- For any node index $\ell$, all $\ell$-prefixes (sets of indices preceding $\ell$ in the sequences) are disjoint.

**Better in reality (i.e., under random failures):**

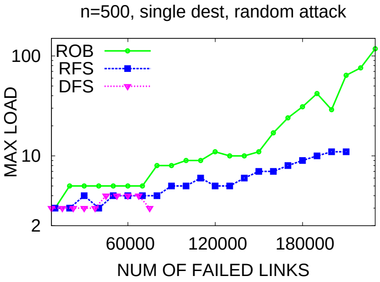
- ROB is a simple destination-based scheme.
- In ROB, when a link fails, use next available link.



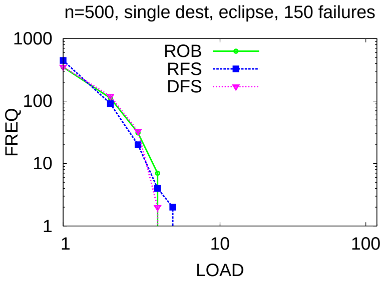
# Simulations: Beyond Worst-Case Failures

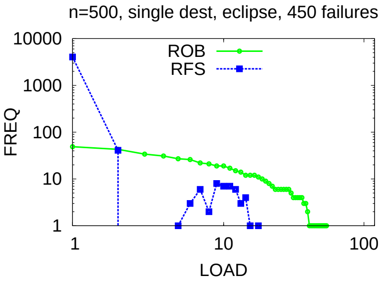
**Better in reality (i.e., under random failures):**

- ROB is a simple destination-based scheme.
- In ROB, when a link fails, use next available link.




**Only small fraction of links highly loaded:**



# Summary

- How to shoot in your foot:
  - No local failover scheme can tolerate more than $n - 1$ failures.
  - Any local failover scheme can yield a max load of $\sqrt{\varphi}$, where $\varphi < n$.
  - Any **destination-based** local failover scheme can yield a max load of $\varphi$, where $\varphi < n$.

# Summary

- How to shoot in your foot:
  - No local failover scheme can tolerate more than $n - 1$ failures.
  - Any local failover scheme can yield a max load of $\sqrt{\varphi}$, where $\varphi < n$.
  - Any **destination-based** local failover scheme can yield a max load of $\varphi$, where $\varphi < n$.
- How **not** to shoot in your foot:
  - **Random** local failover scheme (RFS) yields a max load of at most $\sqrt{\varphi} \log n$, where $\varphi \leq n$.
  - **Deterministic** local failover scheme (DFS) yields a max load of at most $\sqrt{\varphi}$, where $\varphi \leq \log n$.
  - A **simple destination-based** local failover scheme (ROB) performs well under random failures.

# Summary

- How to shoot in your foot:
  - No local failover scheme can tolerate more than $n-1$ failures.
  - Any local failover scheme can yield a max load of $\sqrt{\varphi}$, where $\varphi < n$.
  - Any **destination-based** local failover scheme can yield a max load of $\varphi$, where $\varphi < n$.
- How **not** to shoot in your foot:
  - **Random** local failover scheme (RFS) yields a max load of at most $\sqrt{\varphi} \log n$, where $\varphi \leq n$.
  - **Deterministic** local failover scheme (DFS) yields a max load of at most $\sqrt{\varphi}$, where $\varphi \leq \log n$.
  - A **simple destination-based** local failover scheme (ROB) performs well under random failures.
- Extensions and future work:
  - For **random** failures, RFS yields a max load of at most $\sqrt{\varphi}$.
  - All-to-all communication.

# Summary

- How to shoot in your foot:
  - No local failover scheme can tolerate more than $n - 1$ failures.
  - Any local failover scheme can yield a max load of $\sqrt{\varphi}$, where $\varphi < n$.
  - Any **destination-based** local failover scheme can yield a max load of $\varphi$, where $\varphi < n$.
- How **not** to shoot in your foot:
  - **Random** local failover scheme (RFS) yields a max load of at most $\sqrt{\varphi}\log n$, where $\varphi \leq n$.
  - **Deterministic** local failover scheme (DFS) yields a max load of at most $\sqrt{\varphi}$, where $\varphi \leq \log n$.
  - A **simple destination-based** local failover scheme (ROB) performs well under random failures.
- Extensions and future work:
  - For **random** failures, RFS yields a max load of at most $\sqrt{\varphi}$.
  - All-to-all communication.

THANK YOU!