# Renaissance: A Self-Stabilizing Distributed SDN Control Plane

M. Canini[1], Iosif Salem[2] L. Schiff[3], E. M. Schiller[2], S. Schmid[4]

UCL Université catholique de Louvain

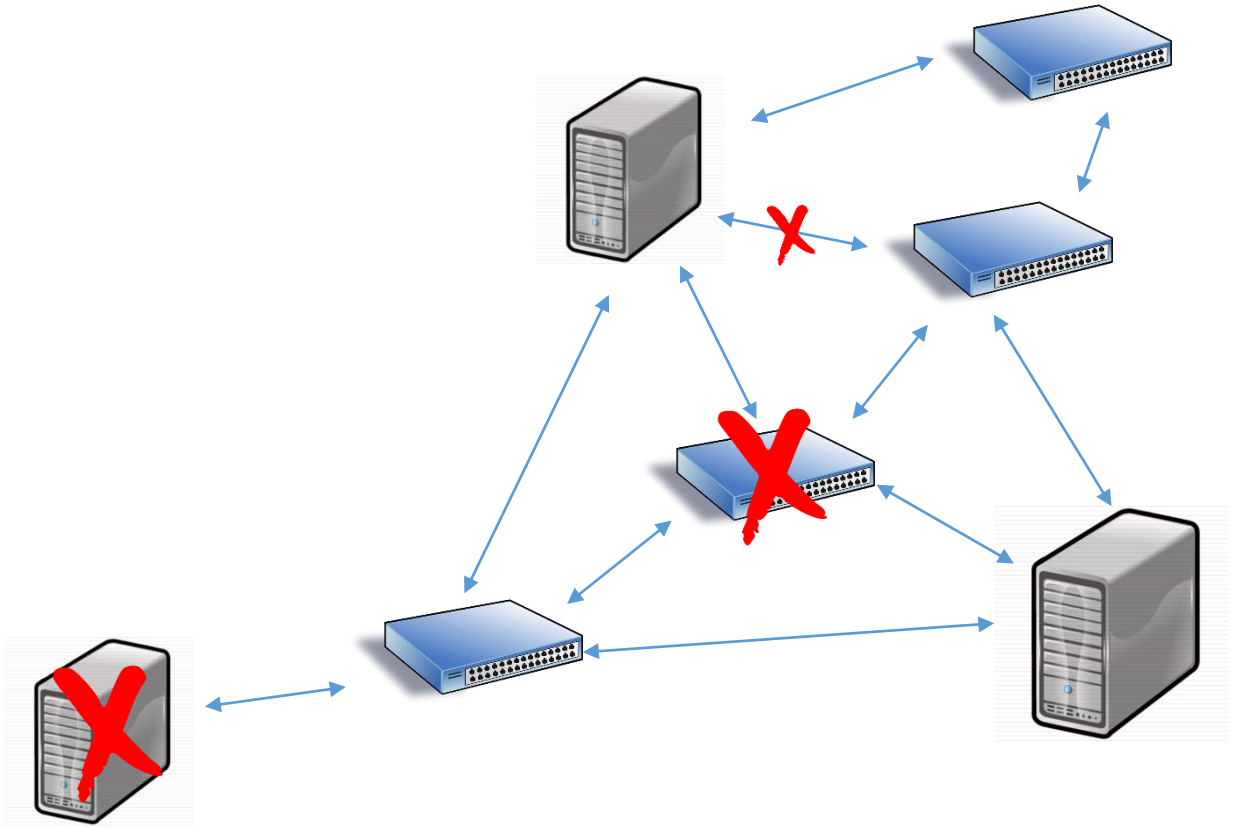
CHALMERS UNIVERSITY OF TECHNOLOGY

GuardiCore

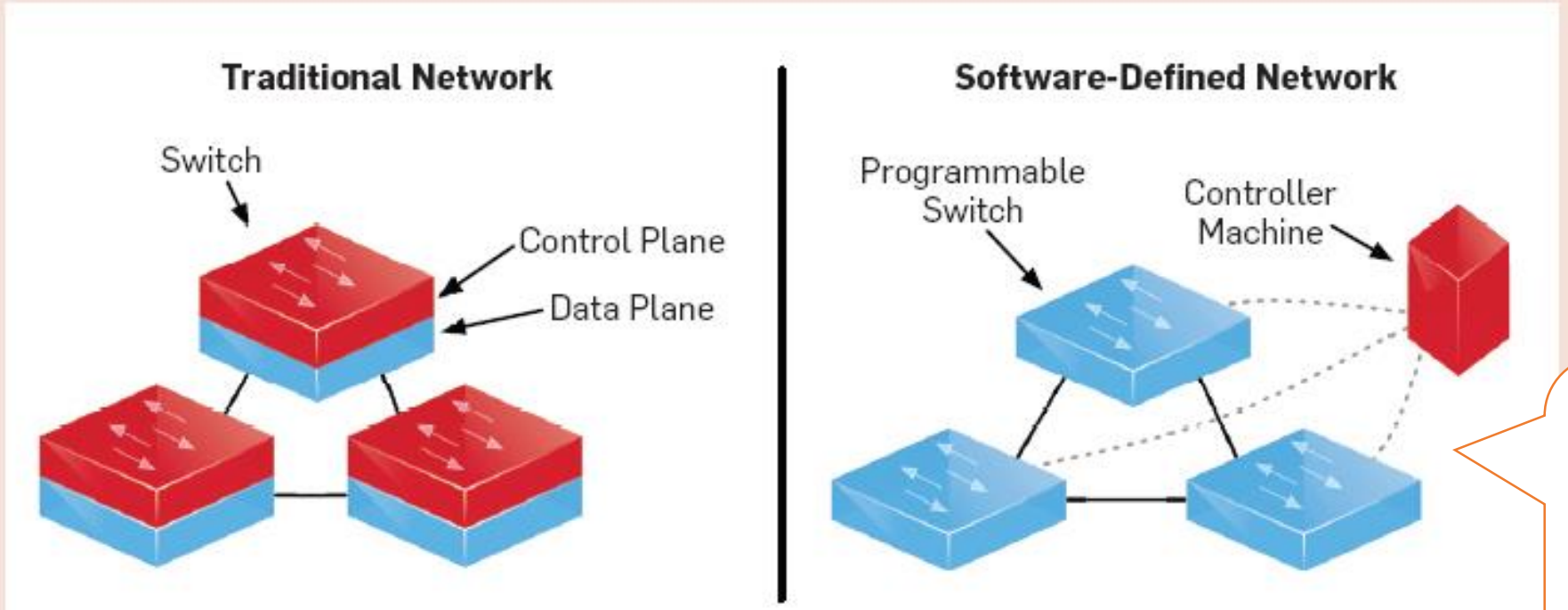
UNIVERSITY OF VIENNA

# In a nutshell

- Software-Defined Network control plane
- Distributed and in-band
- Tolerating:
  - Node/link failures
  - *Arbitrary* failures



# Software-defined networks




**Separation** of control and data plane

Image: Abstractions for Software-Defined Networks. M. Casado, N. Foster, A. Guha. Comm. of the ACM

# Software-defined network control plane

- Logically centralized, physically **distributed**:
  - Reliability
  - Availability
  - Scalability
  - Low latency

- Out-of-band SDN control:
  Physically/logically separate network acts as the controller entity

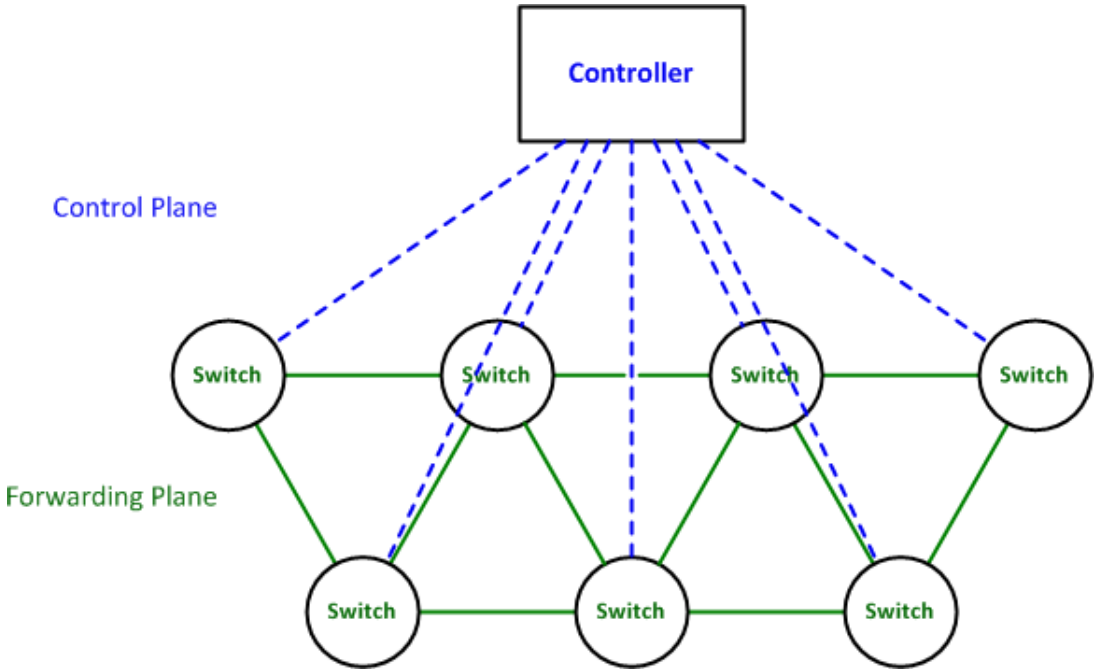
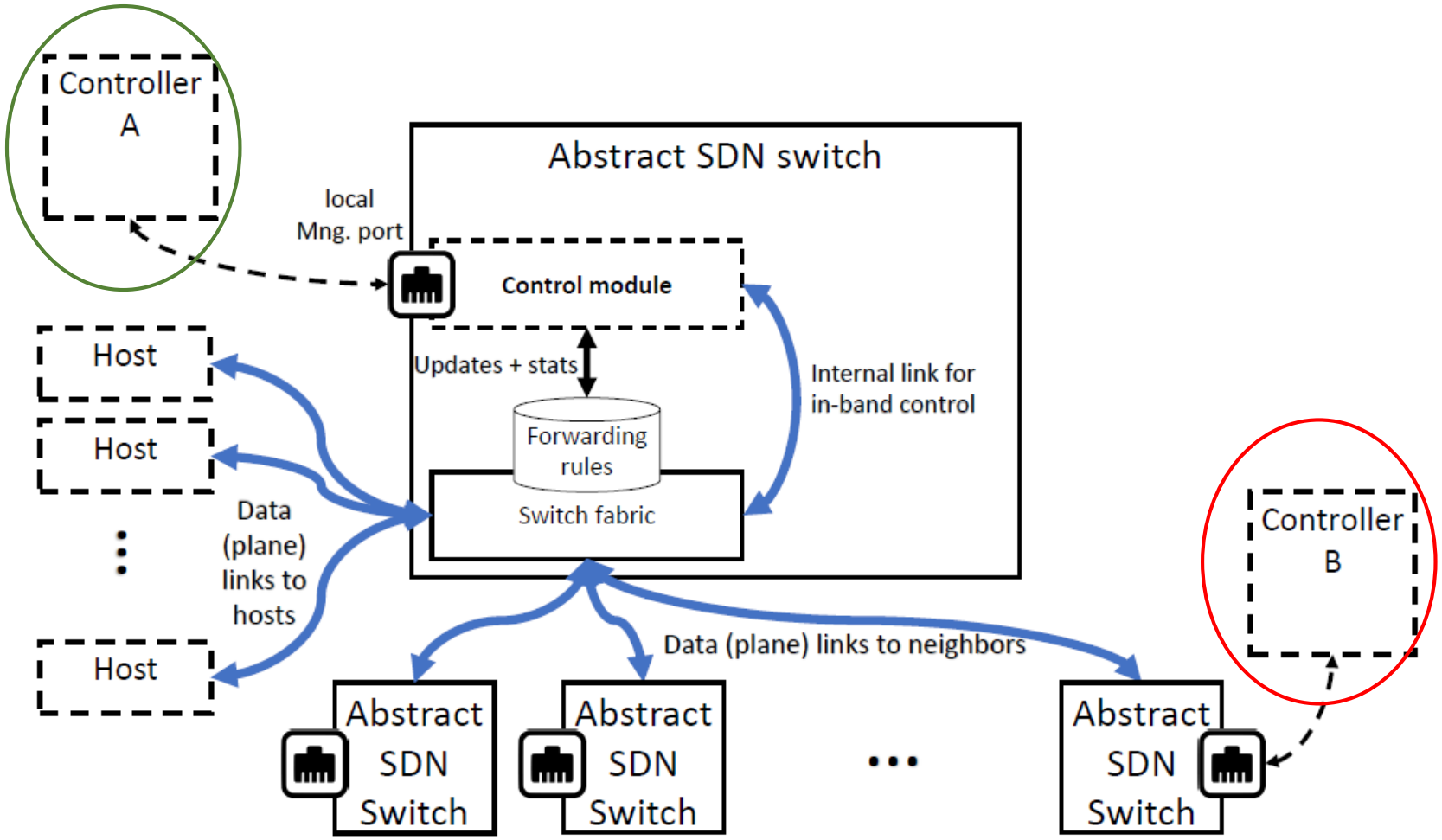



Image: packetlife.net

# In-band SDN control

- Control traffic
  - through dedicated management port (Controller A)
  - **multiplexed** with data-plane traffic (Controller B)
- **Benefits**: less cost, higher redundancy, increased partition tolerance

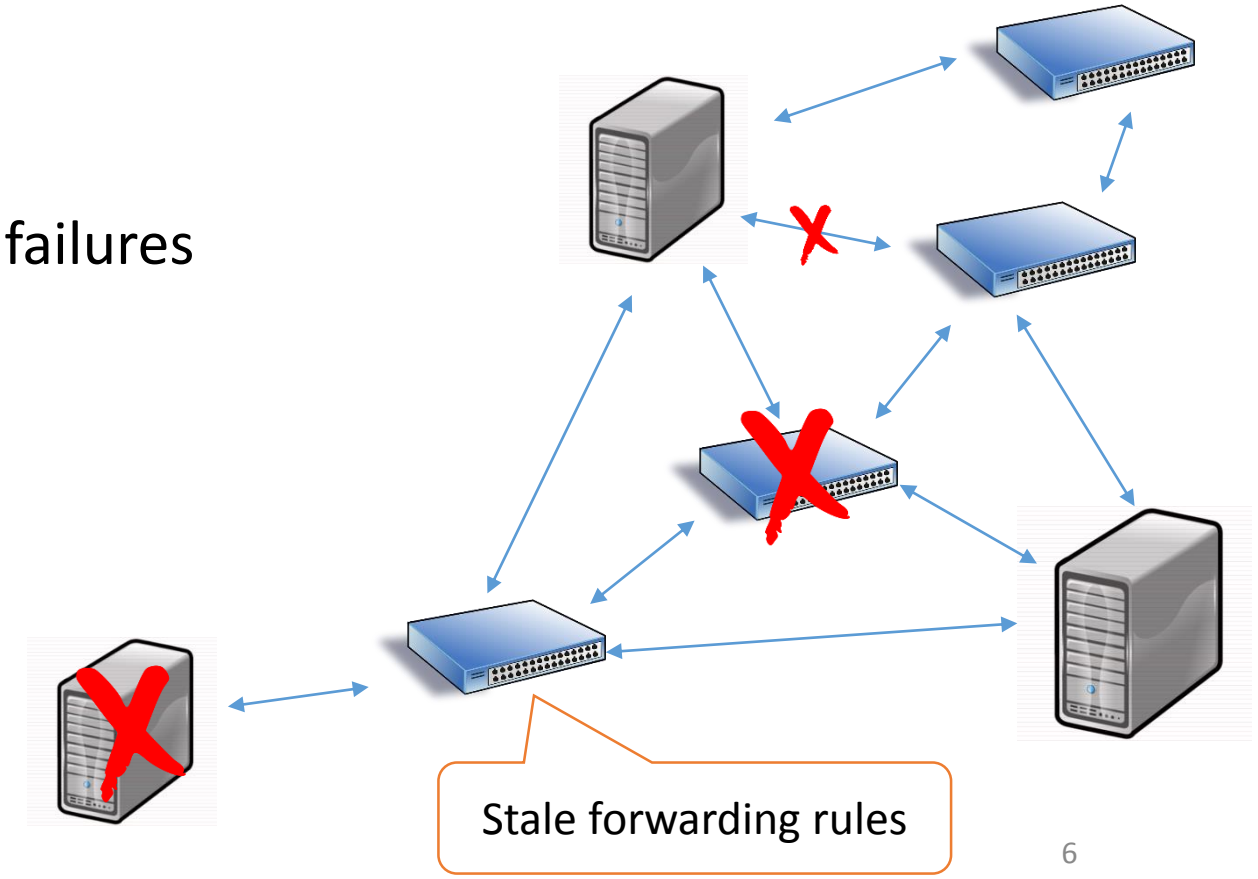


# Problem: *Distributed & In-band Software-defined network control in the presence of failures*

- Establish **bounded communication delays** from every controller to every other node, assuming
  - no out-of-band control
  - fail-stop node/link failures
  - at most K concurrent temporary link failures
  - transient faults

failures

- Only controllers can compute!
  - Switches can only store rules



# Roadmap

- Algorithm
- Proof highlights
- Evaluation

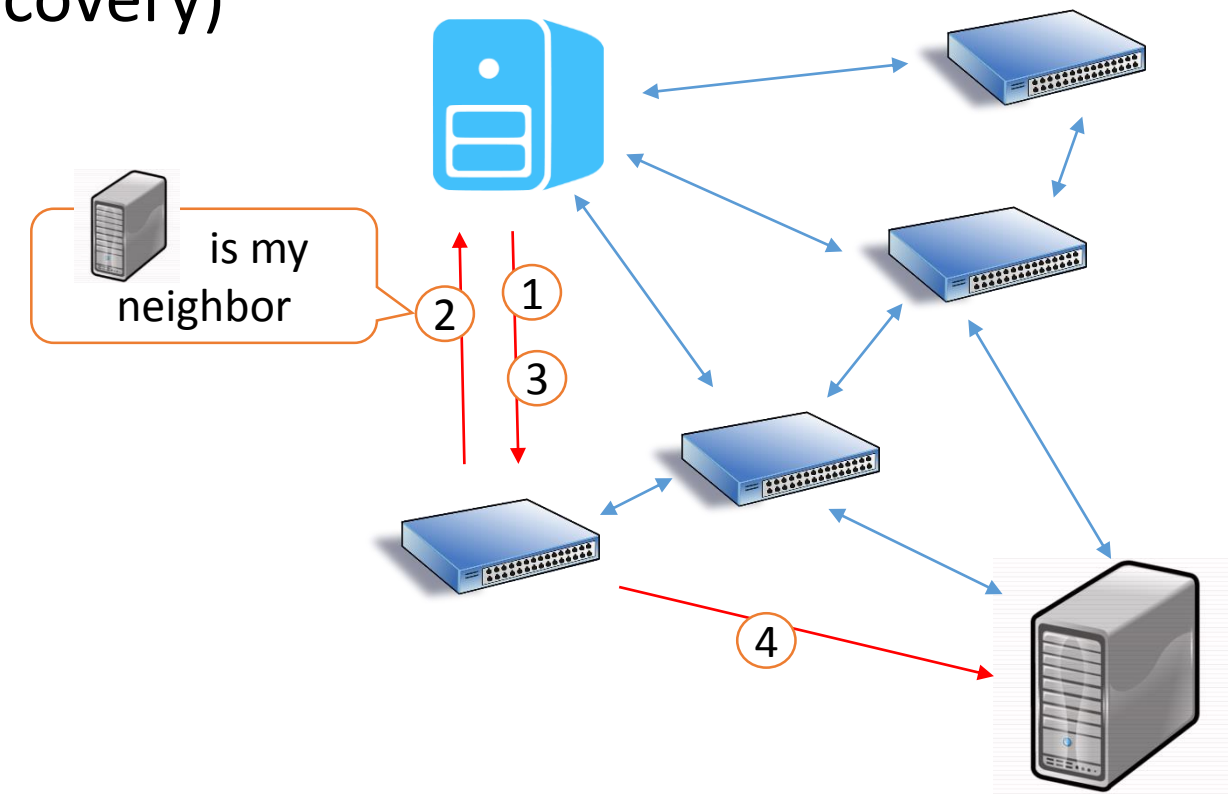
# Roadmap

- Algorithm
- Proof highlights
- Evaluation

# Renaissance: Self-Stabilizing, distributed, in-band control plane

Challenge: discover the network topology

✓Solution: **repeatedly query** discovered nodes about their local topology (BFS discovery)

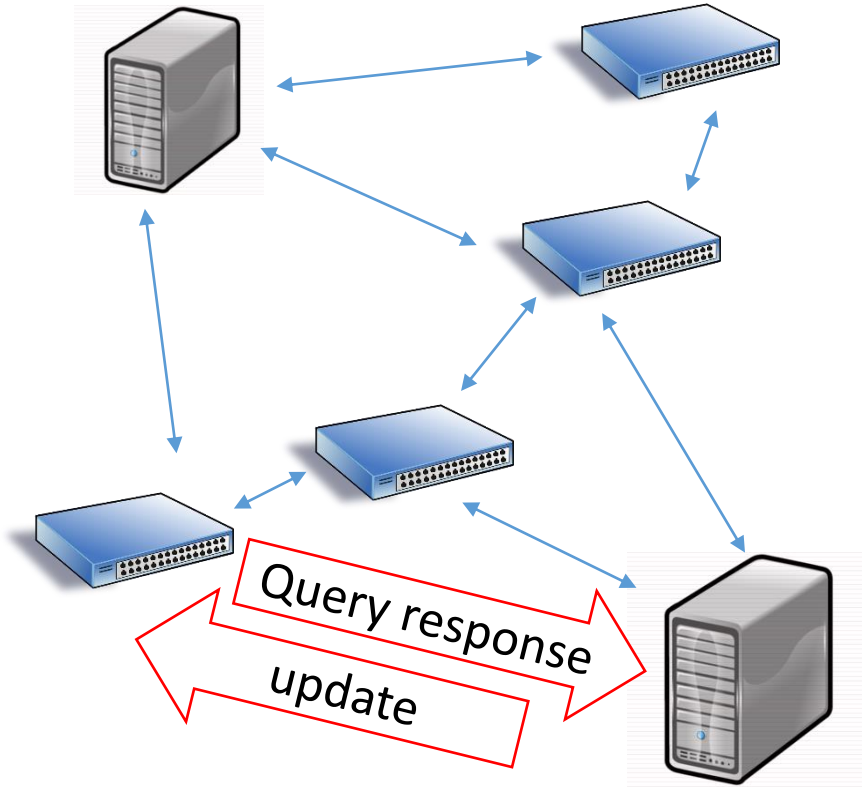


# Renaissance: Self-Stabilizing, distributed, in-band control plane

Challenge: clean up switch memory from stale information

✓Solution: **repeatedly** use query responses, compute updates locally, push to switches

✓Updates include alternative paths, tolerating up to K concurrent link failures

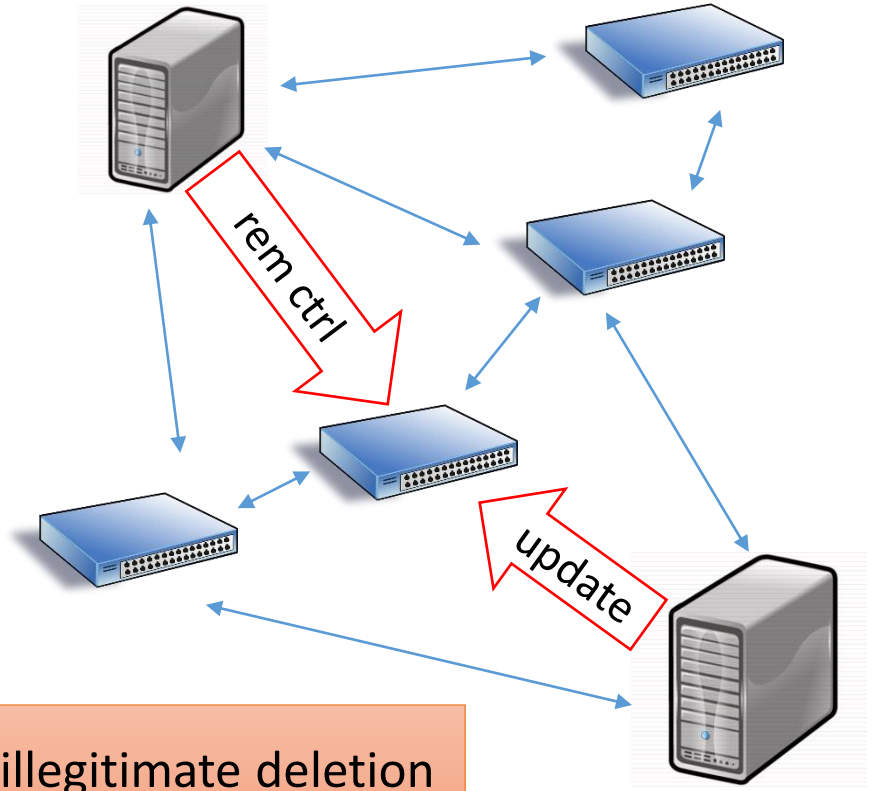


# Renaissance: Self-Stabilizing, distributed, in-band control plane

Challenge: avoid two controllers removing each other's updates

✓Solution:

- use synchronization rounds
- round ends when topology is re-discovered
- *when round ends*,
  remove failing controller info from switches



# Roadmap

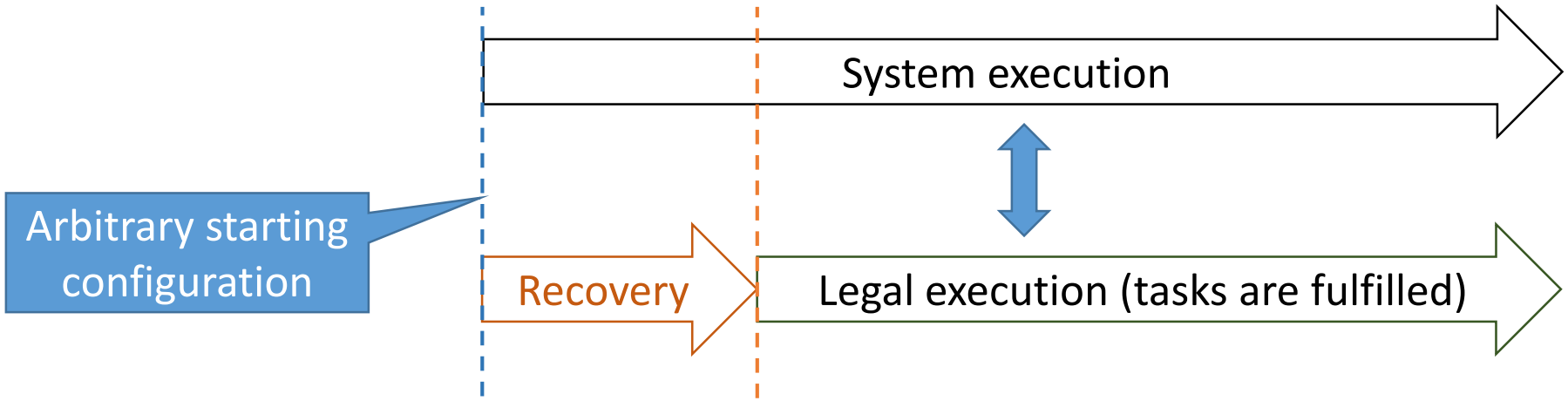
- Algorithm
- Proof highlights
- Evaluation

# Self-stabilizing systems

Bounded recovery after the occurrence of an arbitrary combination of failures

- benign failures (crash failures/recoveries, communication failures, etc.)
- transient faults (arbitrary violation of the system's assumptions)

as long as the algorithm's code stays intact



# Proving bounded recovery period

We show:

- Bounded memory requirements
  - Switch: *O*(#controllers(#controllers + #switches))
  - Controller: *O*(#controllers + #switches)
- Bounded number of illegitimate deletions: (c'•maxDiameter + 1)
- If no illegitimate deletions,
  transient fault recovery within (c"+2)•maxDiameter comm rounds

Can also tolerate topological changes after recovery in *O*(maxDiameter)

Recovery within:
((c"+2)• maxDiameter + 1) • [#illegitimateDeletions • #switches + #controllers + 1] =
***O*(maxDiameter² • #nodes)** rounds

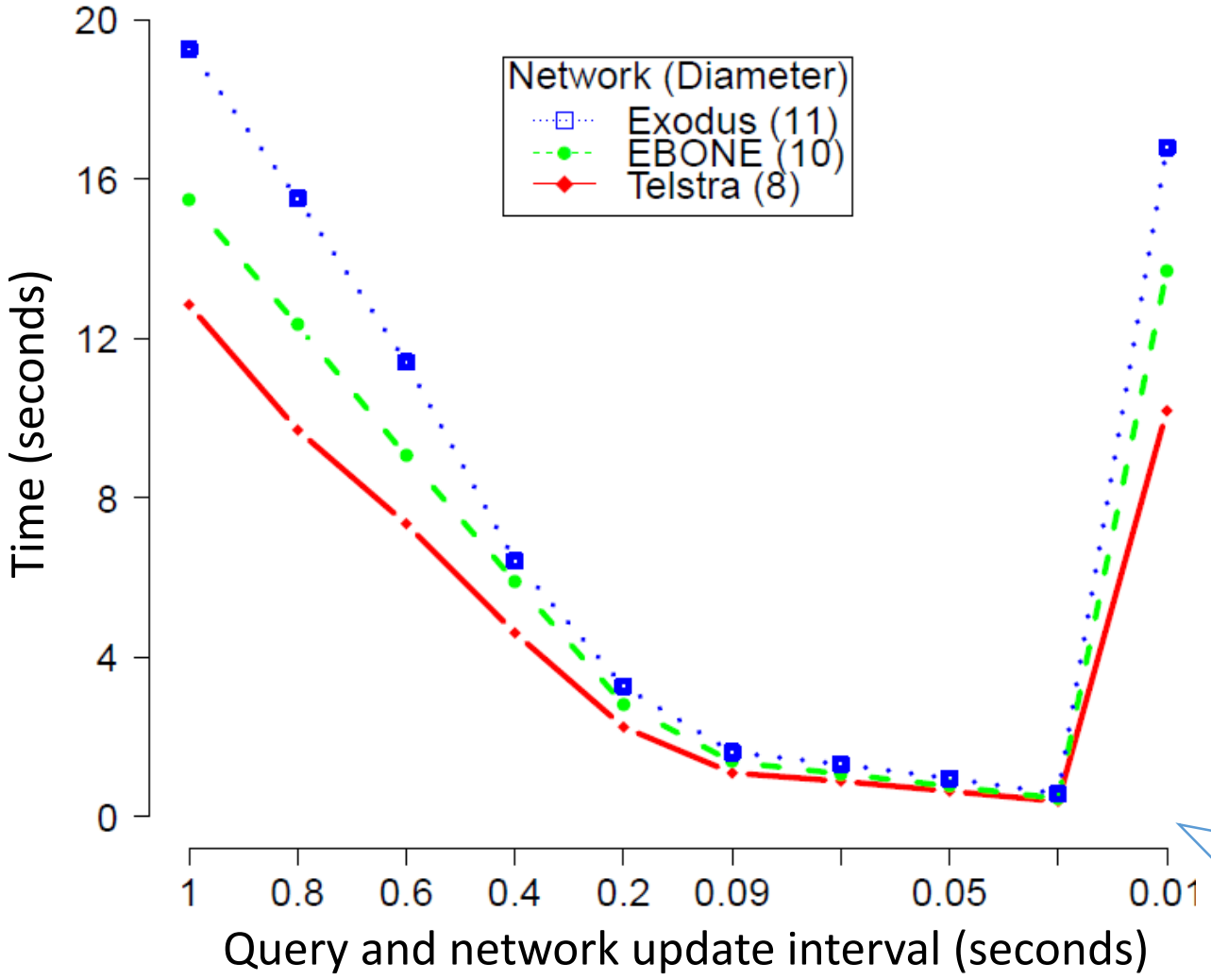
#nodes = #controllers + #switches

# Roadmap

- Algorithm
- Proof highlights
- Evaluation
  - Done on a PC, using Mininet, and testing standard SDN topologies such as Clos, B4, and Rocketfuel networks (Exodus, Telstra, Ebone)

    source code: [renaissance-sdn.net](renaissance-sdn.net)

# How efficiently can Renaissance bootstrap an SDN?




Bootstrap time: Empty switch configuration to legitimate state

- bootstrap time reduces when reducing query and network update interval, until saturation
- bootstrap time is proportional to network diameter
- 4-5 seconds for all tested topologies

Bootstrap time for Rocketfuel networks using **7 controllers**, as a function of query intervals

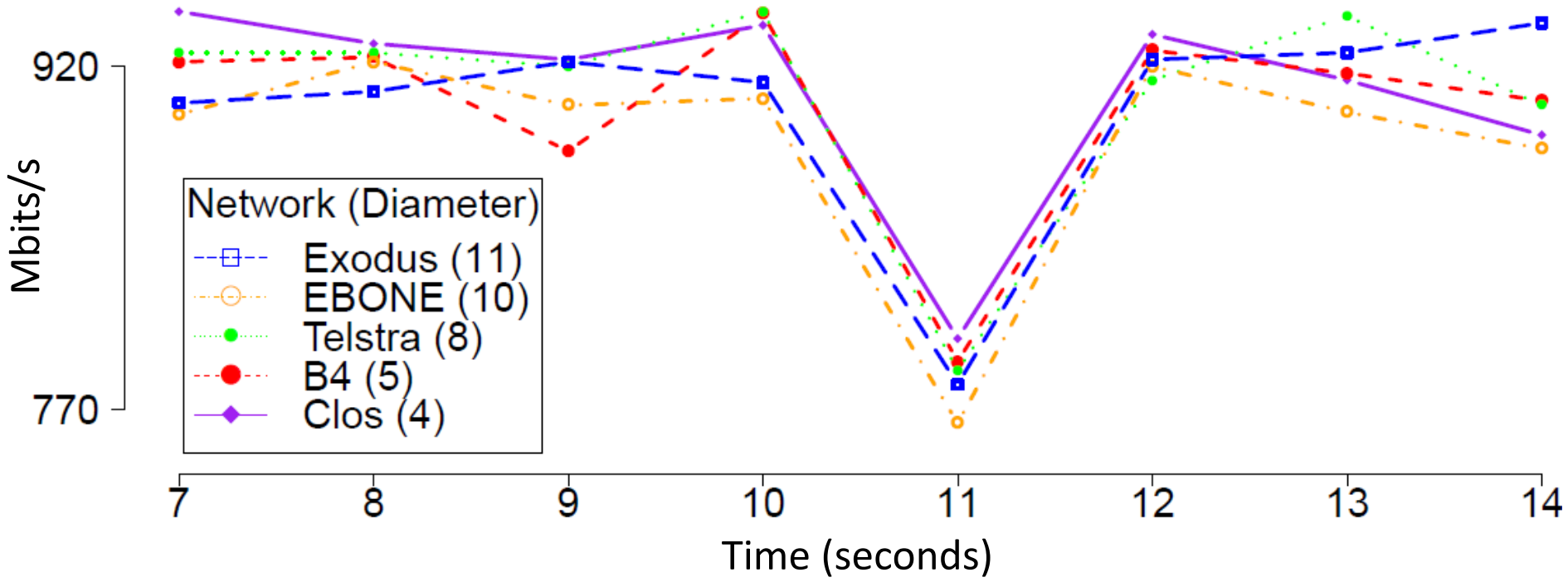
# How efficiently does Renaissance recover in the presence of link and node failures?

Legitimate state → link/node failure → Legitimate state

| | Recovery after failure (seconds) | | |
|---|---|---|---|
| **#controllers (topology)** | **1 controller failure** | **1-6 controller failures** | **2-6 permanent link failures** |
| 3 controllers (B4, Clos) | ~ 3.5 to 5 seconds | - | ~ 3.5 to 5 seconds |
| 7 controllers (Rocketfuel networks) | | ~ 4 to 5 seconds | |

- Recovery time roughly **linear** in the number of nodes
- Diameter affects time to recover to a **small** extent

# Throughput and message loss upon link failure




Link failure in primary path:

- Throughput  drop roughly from 900 Mbits/s to 750 Mbits/s for 2 seconds
- Avoid further drop by packet tagging and forcing traffic through alternative paths

# Wrap-up

Thank you for your attention!

Self-stabilizing, distributed, in-band, control of software-defined networks in the presence of failures

- Deal with **concurrent** updates of switches
- Bounded recovery from topological/comm failures, **transient faults**

Future directions:

- Combination of in-band and out-of-band control
- Consider data traffic dynamics when constructing backup paths