Computer Networks

EDA387/DIT663

Fault-tolerant Algorithms for Computer Networks

Self-stabilizing Software Defined Networks

(Based on slides prepared by Iosif Salem)

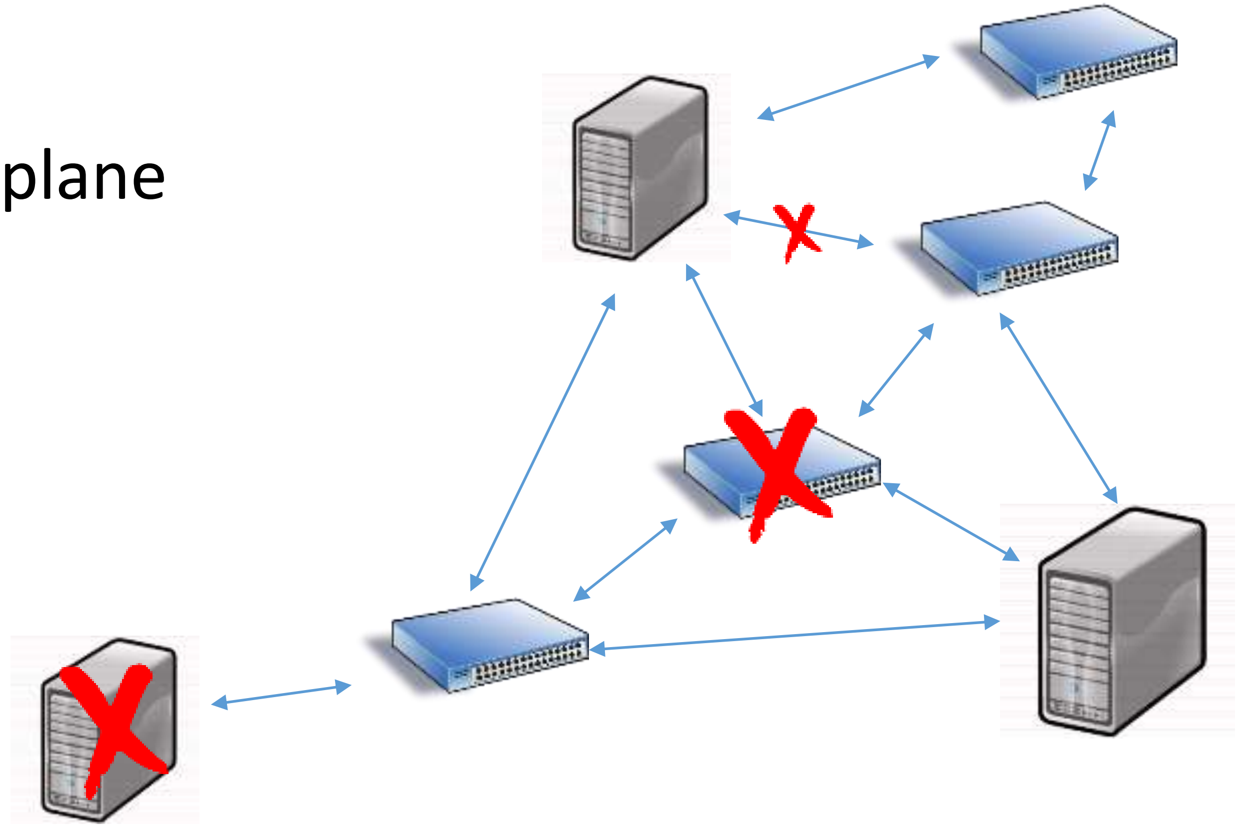
Goals

- The review and understand software defined networks (SDNs).
- To understand *Renaissance* [1].
- To prepare for a lab on Renaissance.

[1] Marco Canini, Iosif Salem, Liron Schiff, Elad Michael Schiller, Stefan Schmid, ``Renaissance: A Self-Stabilizing Distributed SDN Control Plane'' *38th IEEE International Conference on Distributed Computing Systems* (ICDCS) 2018: 233-243

In a nutshell

- Software-Defined Network control plane
- Distributed and in-band
- Tolerating:
  - Node/link failures
  - *Arbitrary* failures
- For detailed review, cf. lesson 11 at Computer Networking (Georgia Tech)



Why Software-Defined Networks (SDN)?

Application demands from the network

Decision making: Computing/Installing packet forwarding rules, local computations

Packet forwarding, statistics

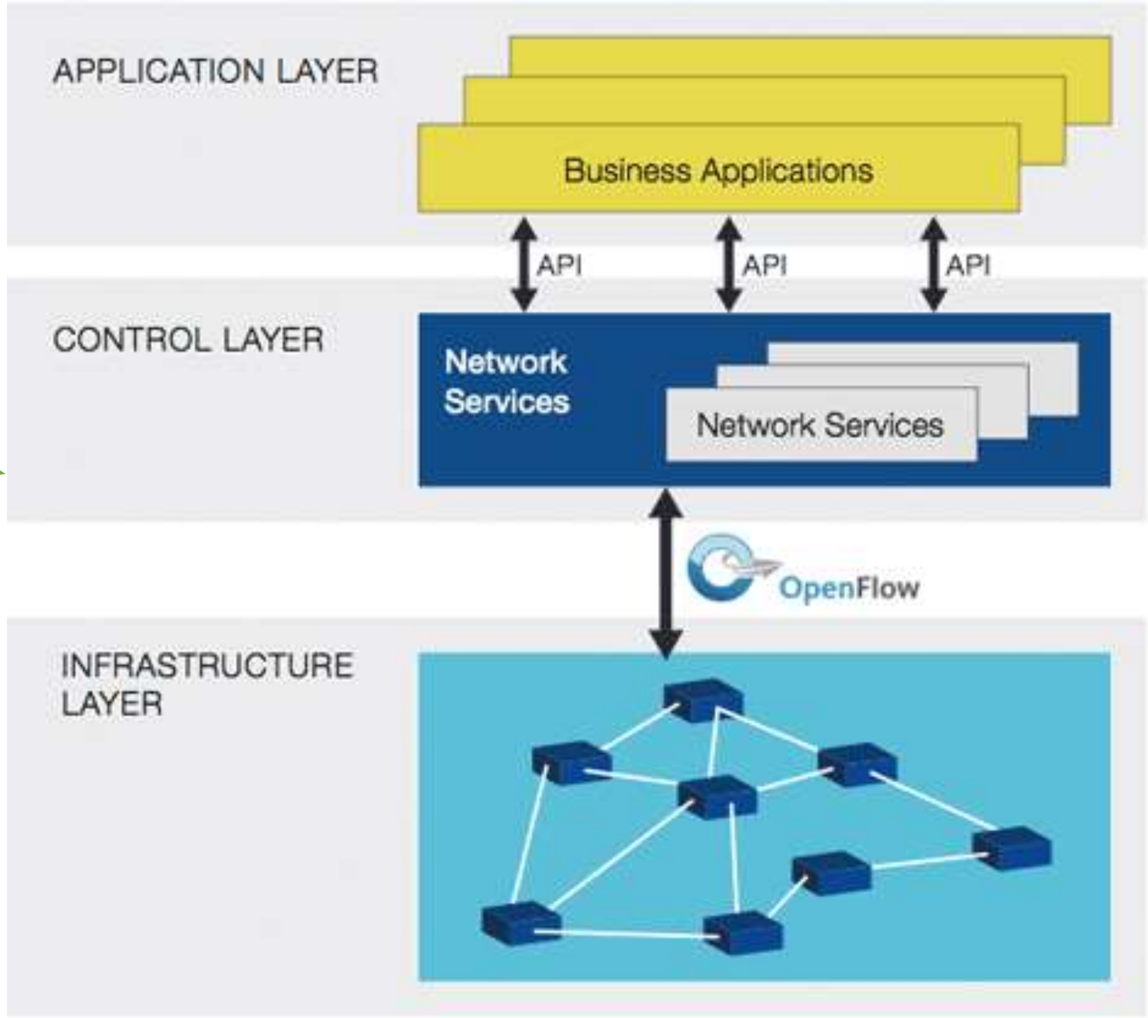


Image: Open Networking Foundation

Separation of Control and Data Plane

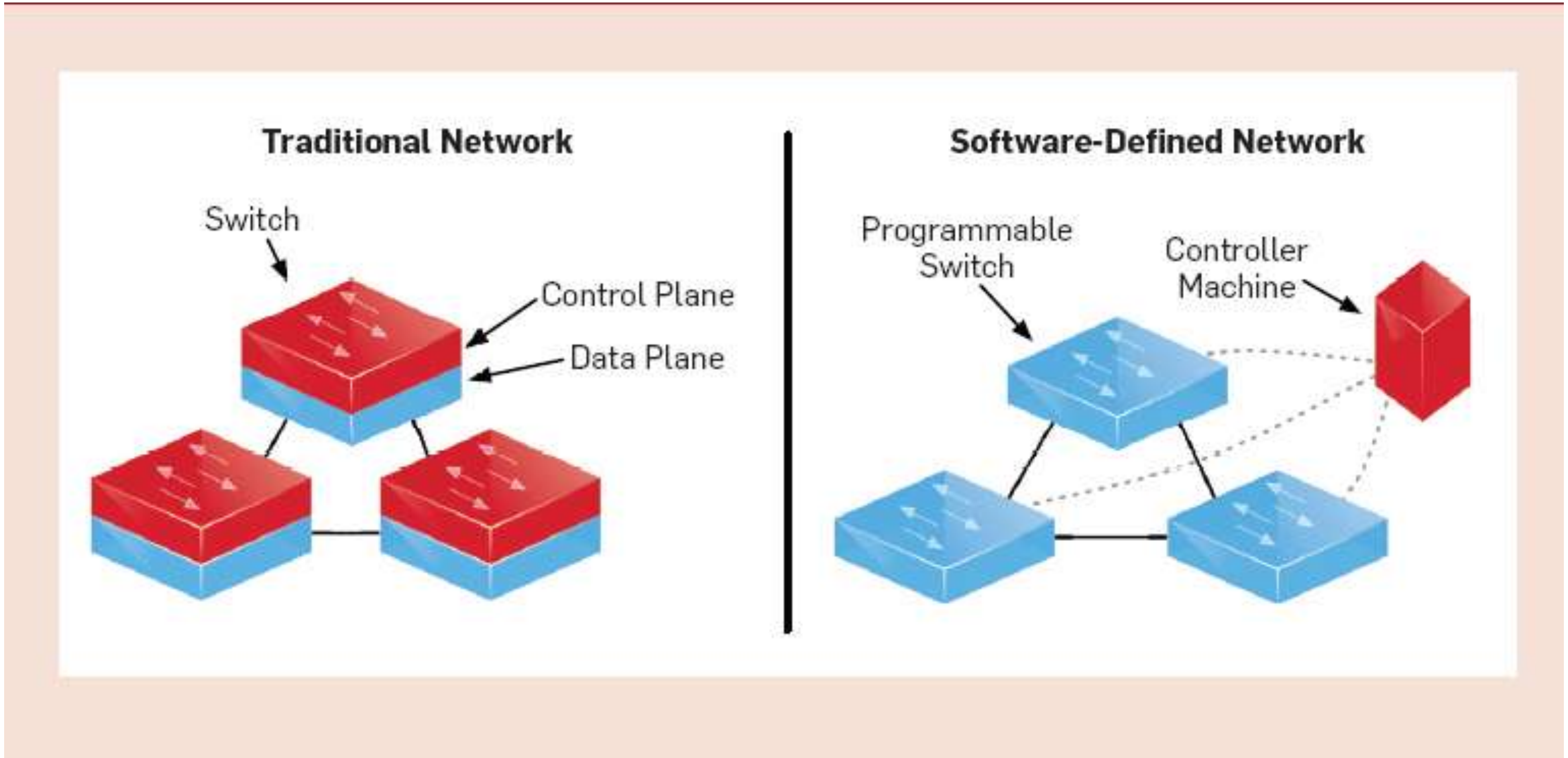


Image: Abstractions for Software-Defined Networks.
M. Casado, N. Foster, A. Guha. Comm. of the ACM

SDN switch

- A switch can
  - Receive control-traffic by a managing controller, e.g., including packet-forwarding rules.
  - Forward packets based on matching rules in its flow tables
    - A rule includes a source, destination, and action field: when a packet matches the source and destination of a rule, the switch takes the rule's action (forward to a neighbor)
    - A rule may include also priority and metadata fields

  - Keep network statistics (for the controller)
  - Respond to controller queries

SDN controller

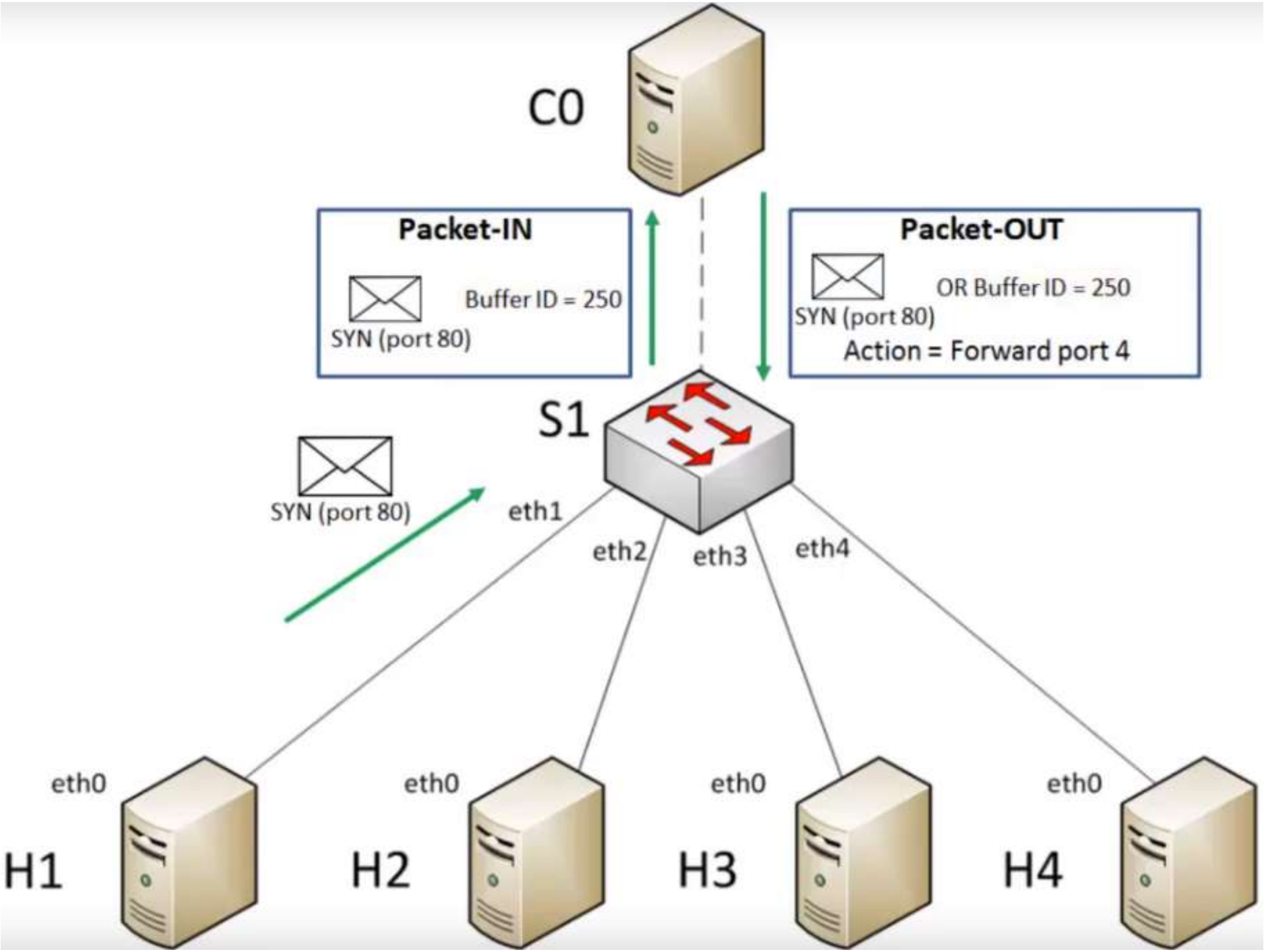
- A controller (network-attached server) can
  - manage switches
    - Ask for network statistics
    - Access switch local storage
    - Packet-out (response to packet-in), flow-MOD (install/modify rules, modify switch buffer)
  - do local computations
    - Re-compute network flows based on statistics (counters on switches)
    - Compute packet-forwarding rules for each switch they manage
    - Install flows, by updating packet forwarding rules
    - Rule updates are not trivial (loop-free updates NP-complete*)

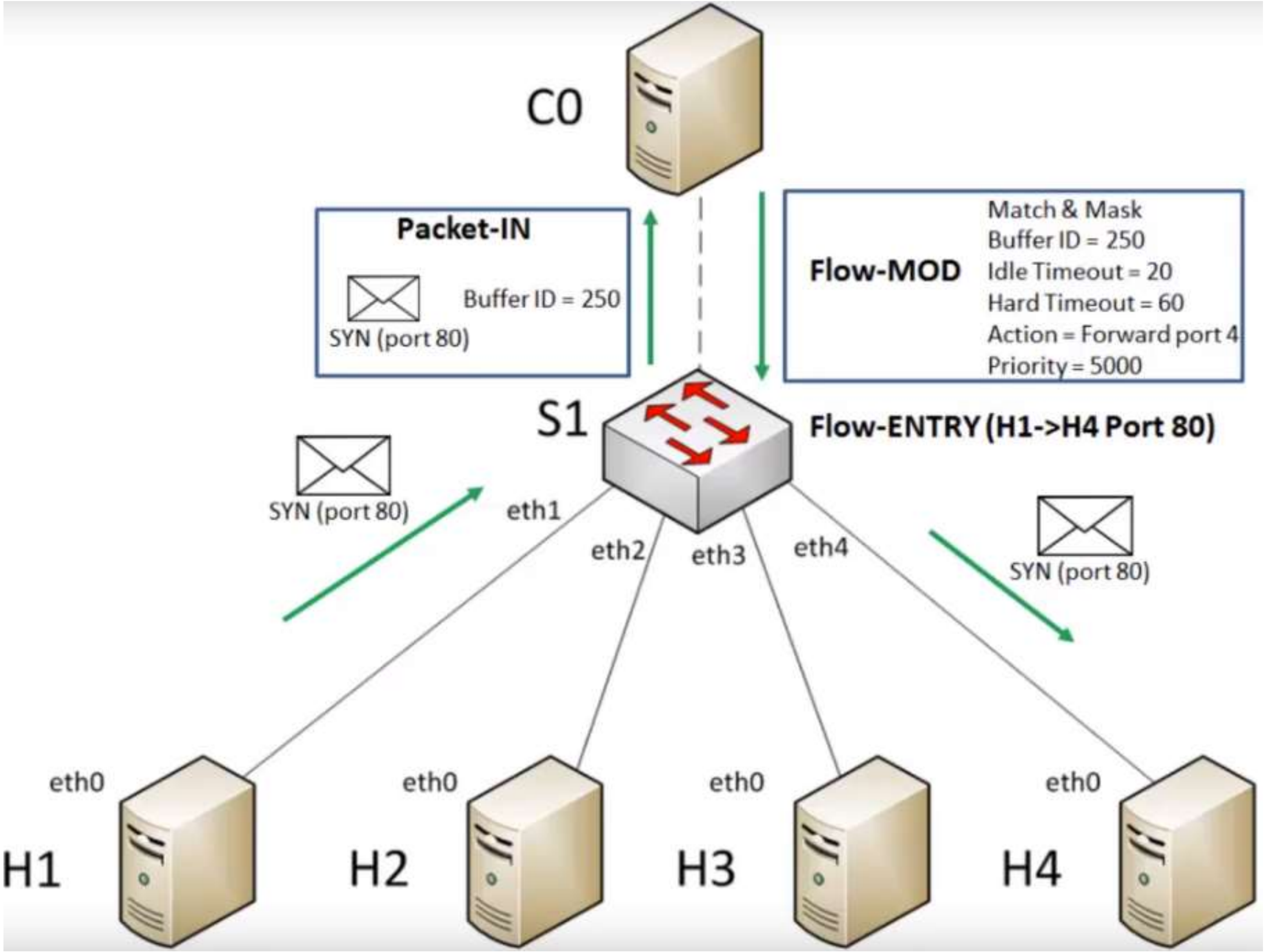
*Scheduling Loop-free Network Updates: It's Good to Relax! Arne Ludwig, Jan Marcinkowski, Stefan Schmid. PODC '15

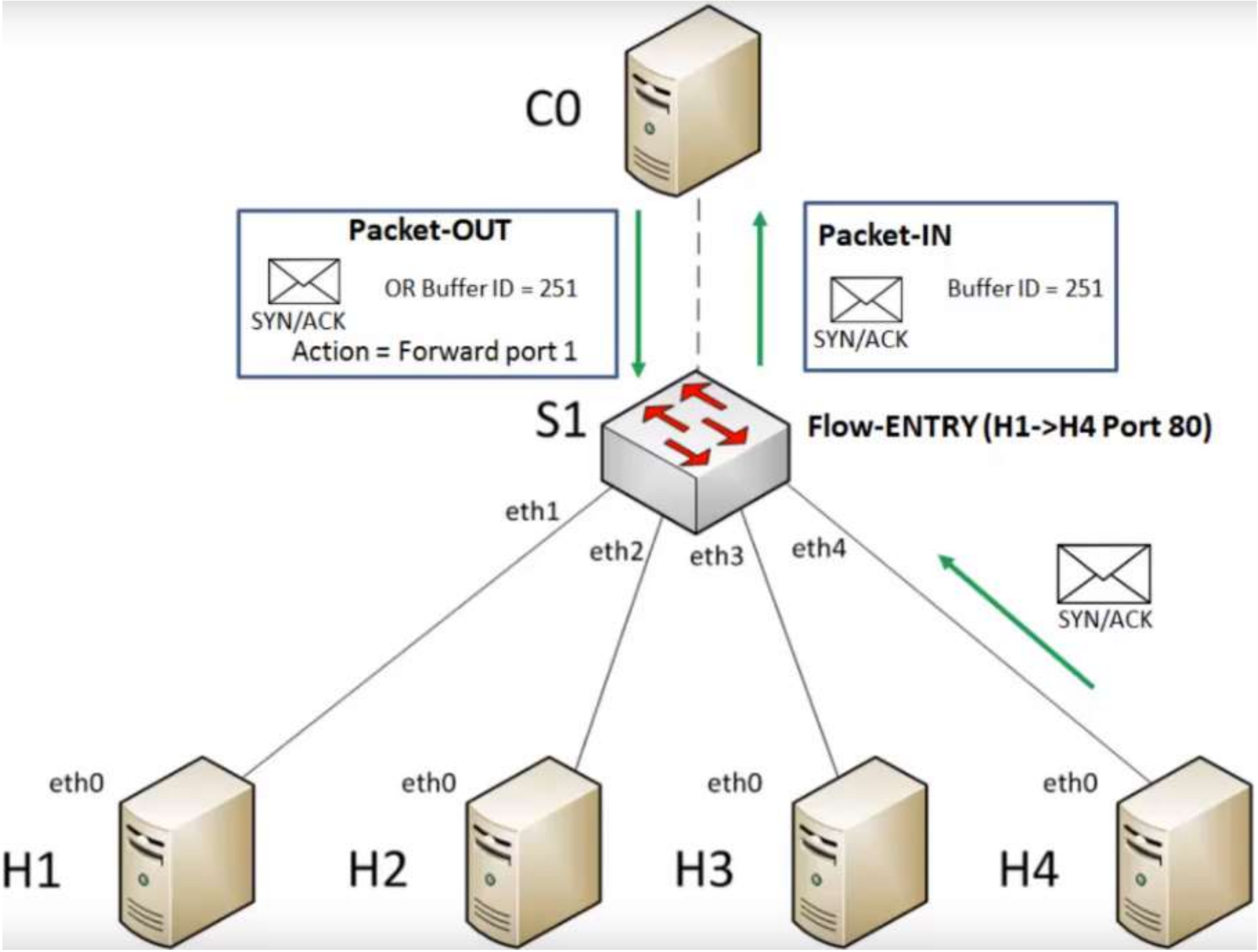
OpenFlow

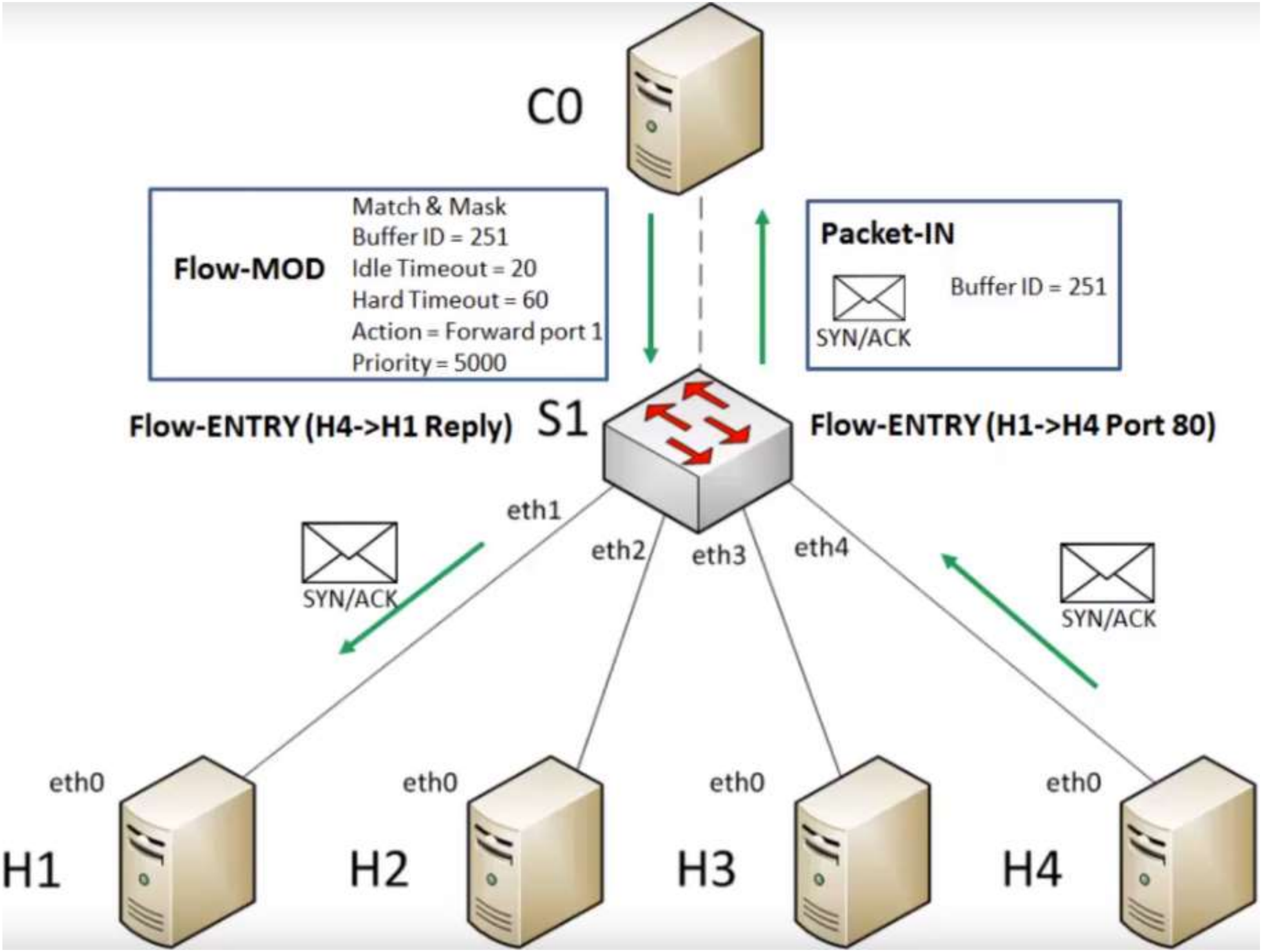
OpenFlow

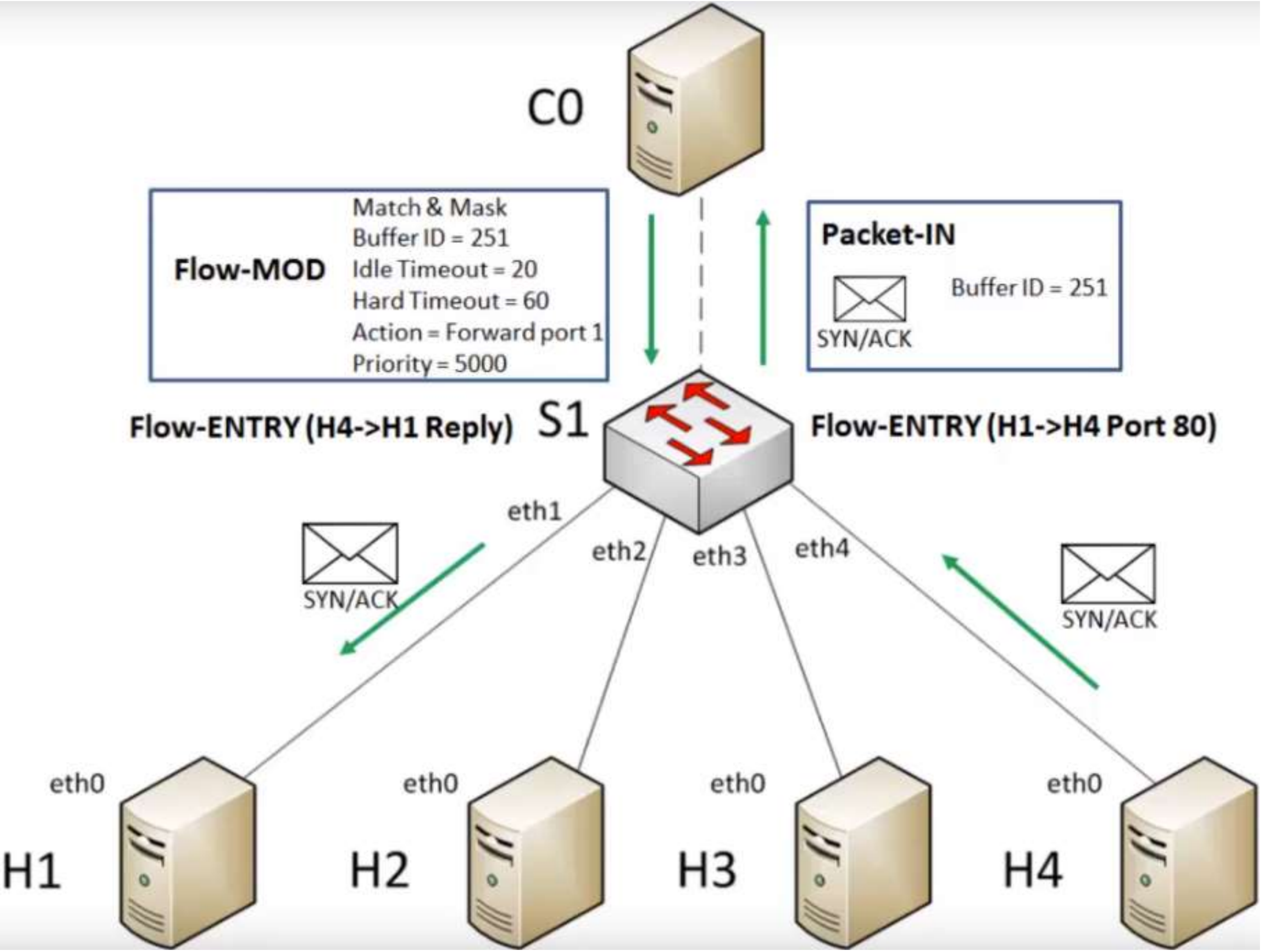
- This protocol enables access to the switch forwarding plane.
  - The controller uses OpenFlow for determining the flow of packets across the network of switches.
  - By separating the control from the packet switching a more elegant network management facilitated than using access control lists and routing protocols.
- OpenFlow allows remote administration of a layer 3, by adding, modifying and removing packet matching rules and actions.
  - Packets which are unmatched by the currently installed switch rules are forwarded to the controller; this is the PacketIN event.
  - The controller can then decide to modify existing rules (Flow-MOD), and/or
    - modify the arriving packet before releasing it (Packet-OUT).

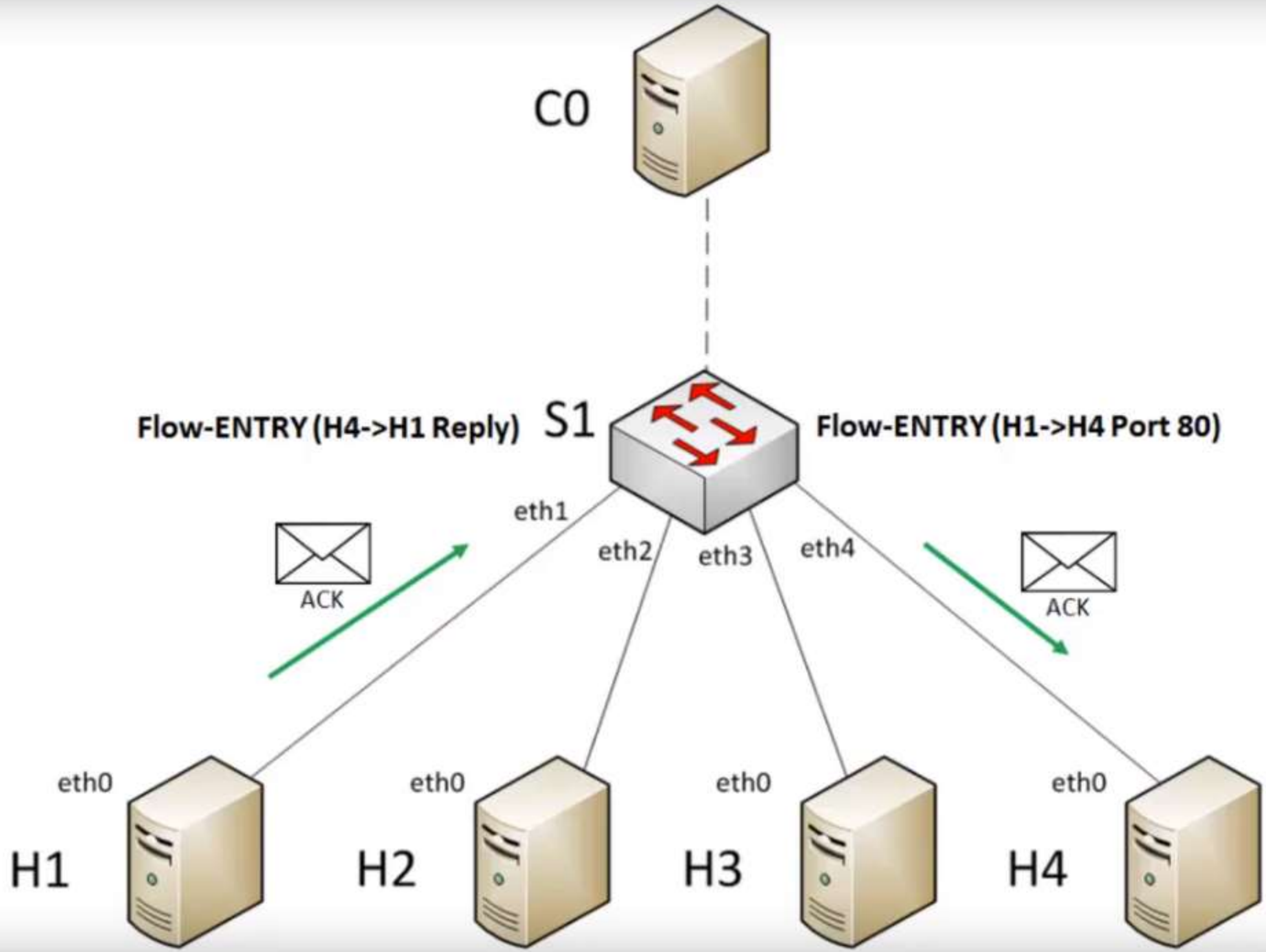

C0
Packet-IN
Buffer ID = 250
SYN (port 80)
Packet-OUT
OR Buffer ID = 250
SYN (port 80)
Action = Forward port 4
S1
SYN (port 80)
eth1
eth2
eth3
eth4
eth0
eth0
eth0
eth0
H1
H2
H3
H4

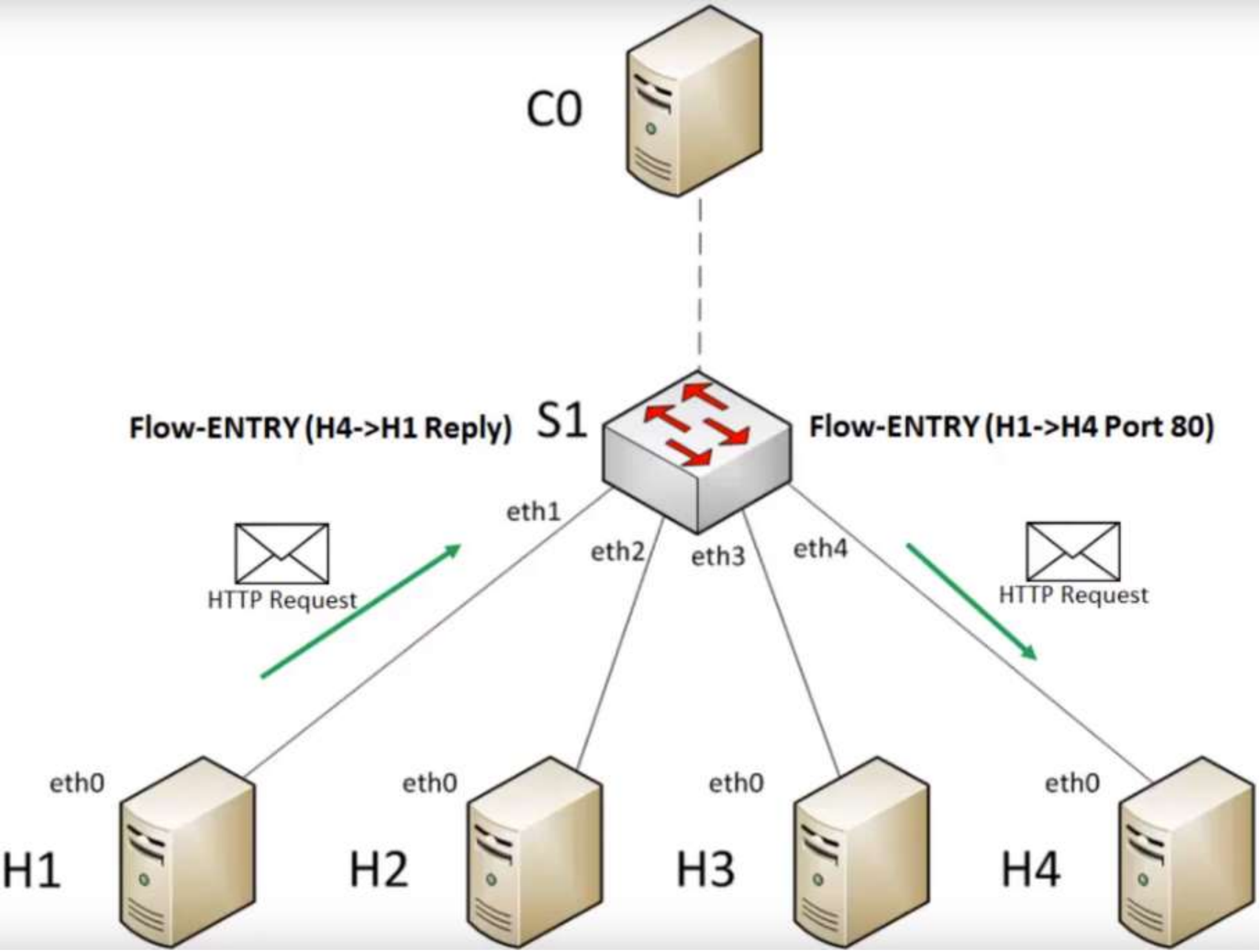

C0
Packet-IN
Buffer ID = 250
SYN (port 80)
Flow-MOD
Match & Mask
Buffer ID = 250
Idle Timeout = 20
Hard Timeout = 60
Action = Forward port 4
Priority = 5000
S1
Flow-ENTRY (H1->H4 Port 80)
SYN (port 80)
eth1
eth2
eth3
eth4
SYN (port 80)
eth0
eth0
eth0
eth0
H1
H2
H3
H4

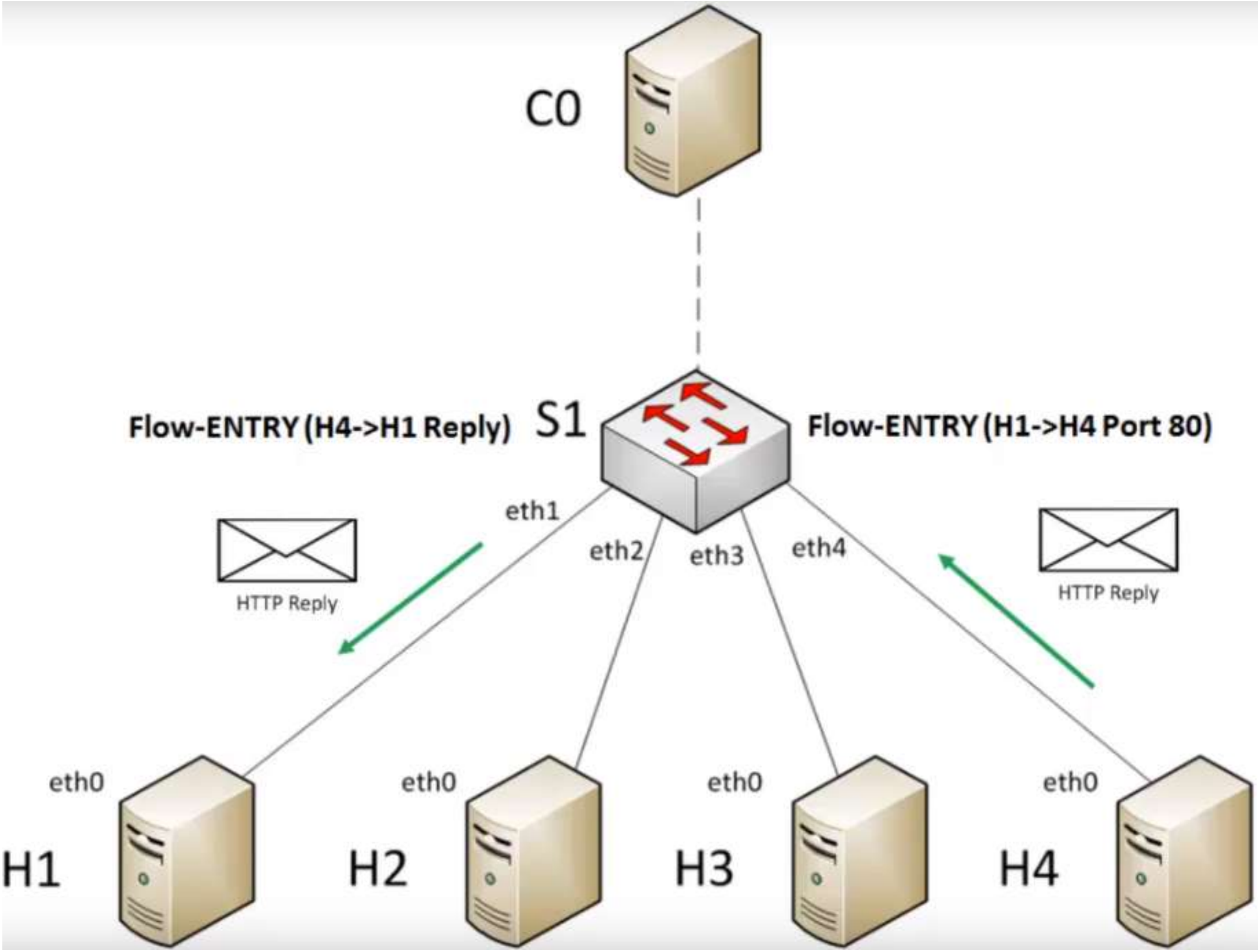

C0
Packet-OUT
SYN/ACK
OR Buffer ID = 251
Action = Forward port 1
Packet-IN
SYN/ACK
Buffer ID = 251
S1
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
SYN/ACK
eth0
eth0
eth0
eth0
H1
H2
H3
H4


C0
Flow-MOD
Match & Mask
Buffer ID = 251
Idle Timeout = 20
Hard Timeout = 60
Action = Forward port 1
Priority = 5000
Packet-IN
Buffer ID = 251
SYN/ACK
S1
Flow-ENTRY (H4->H1 Reply)
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
SYN/ACK
SYN/ACK
eth0
eth0
eth0
eth0
H1
H2
H3
H4


C0
Flow-MOD
Match & Mask
Buffer ID = 251
Idle Timeout = 20
Hard Timeout = 60
Action = Forward port 1
Priority = 5000
Packet-IN
SYN/ACK
Buffer ID = 251
Flow-ENTRY (H4->H1 Reply)
S1
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
SYN/ACK
SYN/ACK
eth0
eth0
eth0
eth0
H1
H2
H3
H4

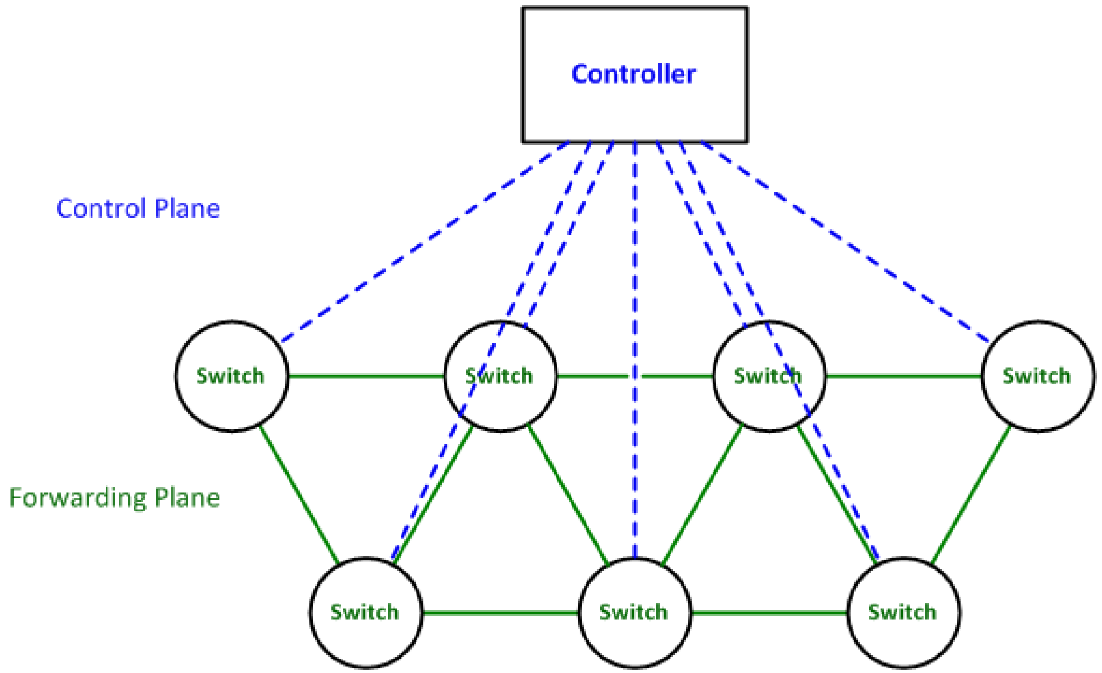

C0
S1
Flow-ENTRY (H4->H1 Reply)
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
ACK
ACK
eth0
eth0
eth0
eth0
H1
H2
H3
H4


C0
S1
Flow-ENTRY (H4->H1 Reply)
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
HTTP Request
HTTP Request
eth0
eth0
eth0
eth0
H1
H2
H3
H4


C0
S1
Flow-ENTRY (H4->H1 Reply)
Flow-ENTRY (H1->H4 Port 80)
eth1
eth2
eth3
eth4
HTTP Reply
HTTP Reply
eth0
eth0
eth0
eth0
H1
H2
H3
H4

Out of Band Control Plane

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

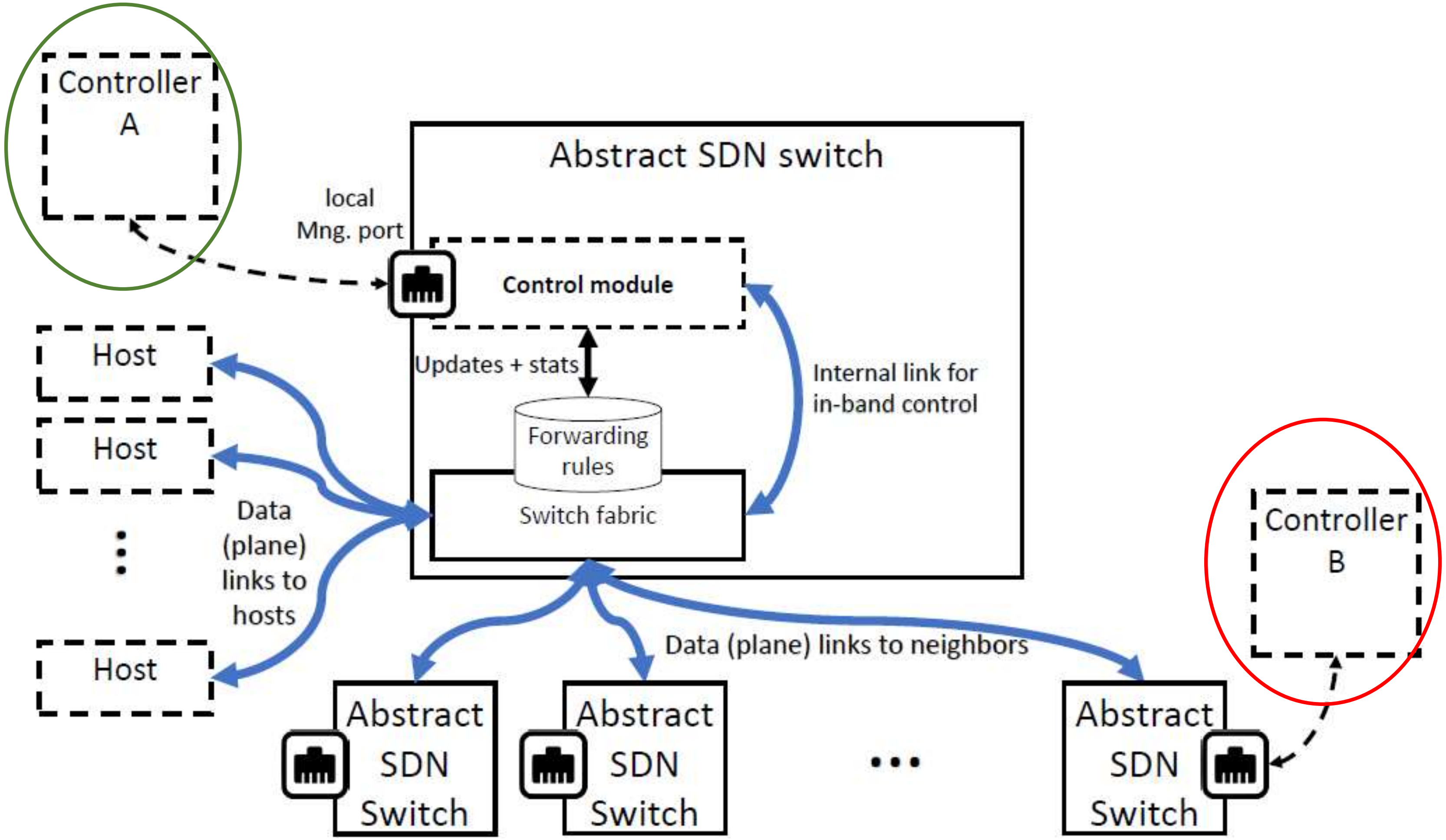


- Out-of-band SDN control:
  Physically/logically separate network acts as the controller entity

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

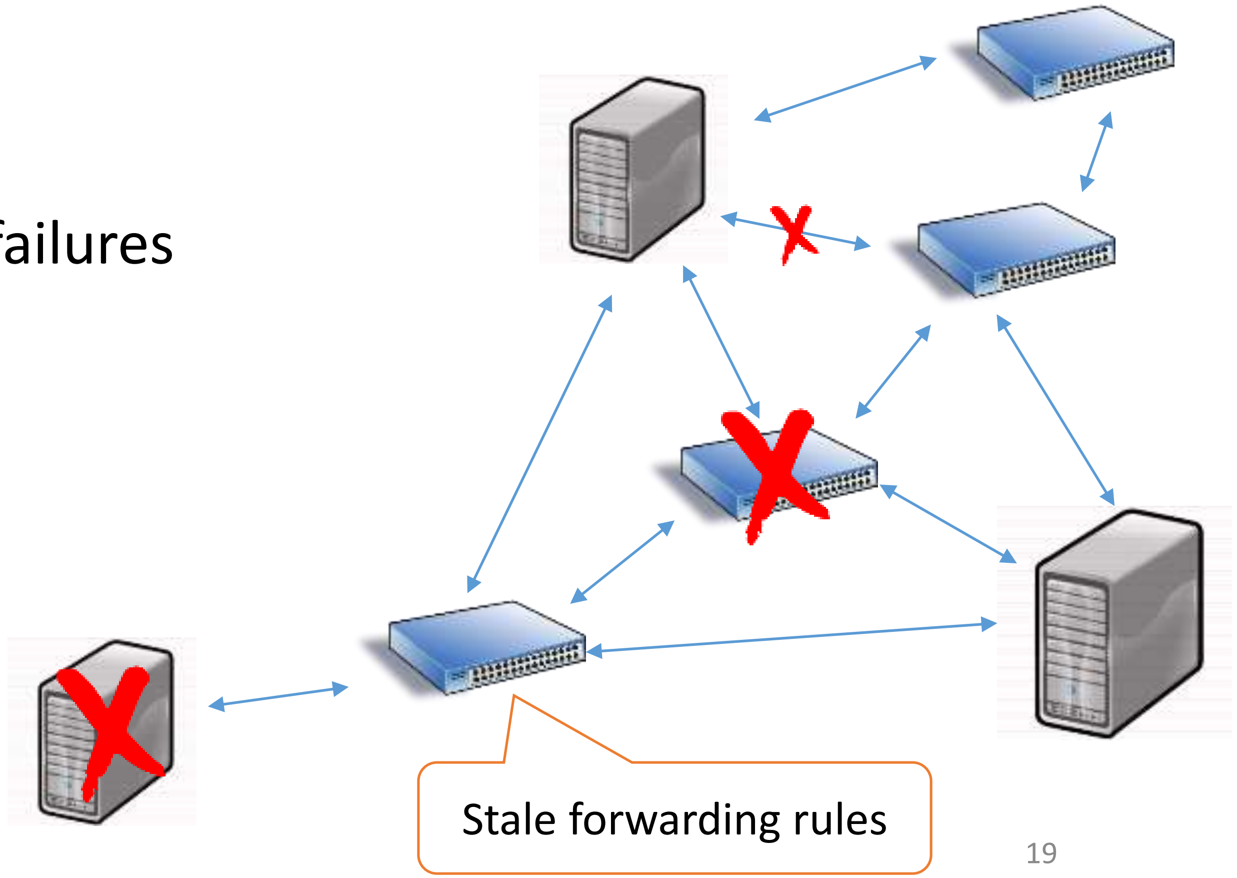


Problem: *Distributed & In-band Software-defined network control* *in the presence of failures*

- 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

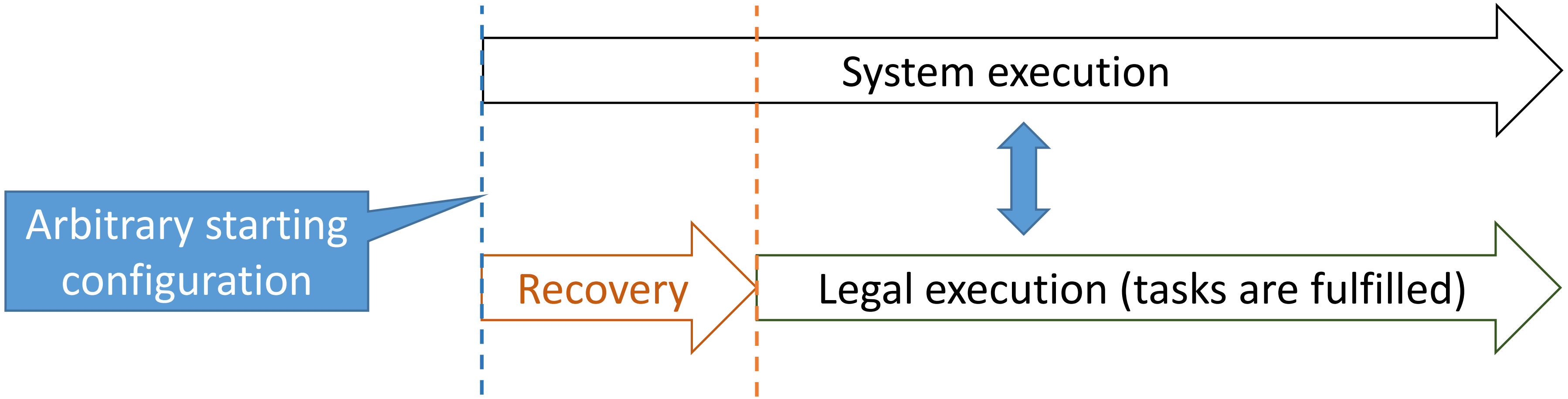


Self-stabilizing systems

Bounded recovery after the occurrence of an arbitrary combination of failures

- benign failures: transient link failures and permanent link/node failures.
- transient faults: arbitrary violation of the system's assumptions

as long as the algorithm's code stays intact



Fault Model

	Frequency	
Duration	*Rare*	*Often*
Transient	Any volition of the assumptions according to which the system is assumed to operation (as long as the code stays intact). This can result in any state corruption.	Packet failures: omissions, duplications, reordering (assuming communication fairness holds).
		Link failures (assuming at most κ links failures).
Permanent	Node and link failures.	

Prior to the system start, consider all faults

Recovery Period

Legal Execution

Consider only non-transient faults

Consider only benign faults

Roadmap

- Algorithm
- Proof highlights
- Evaluation

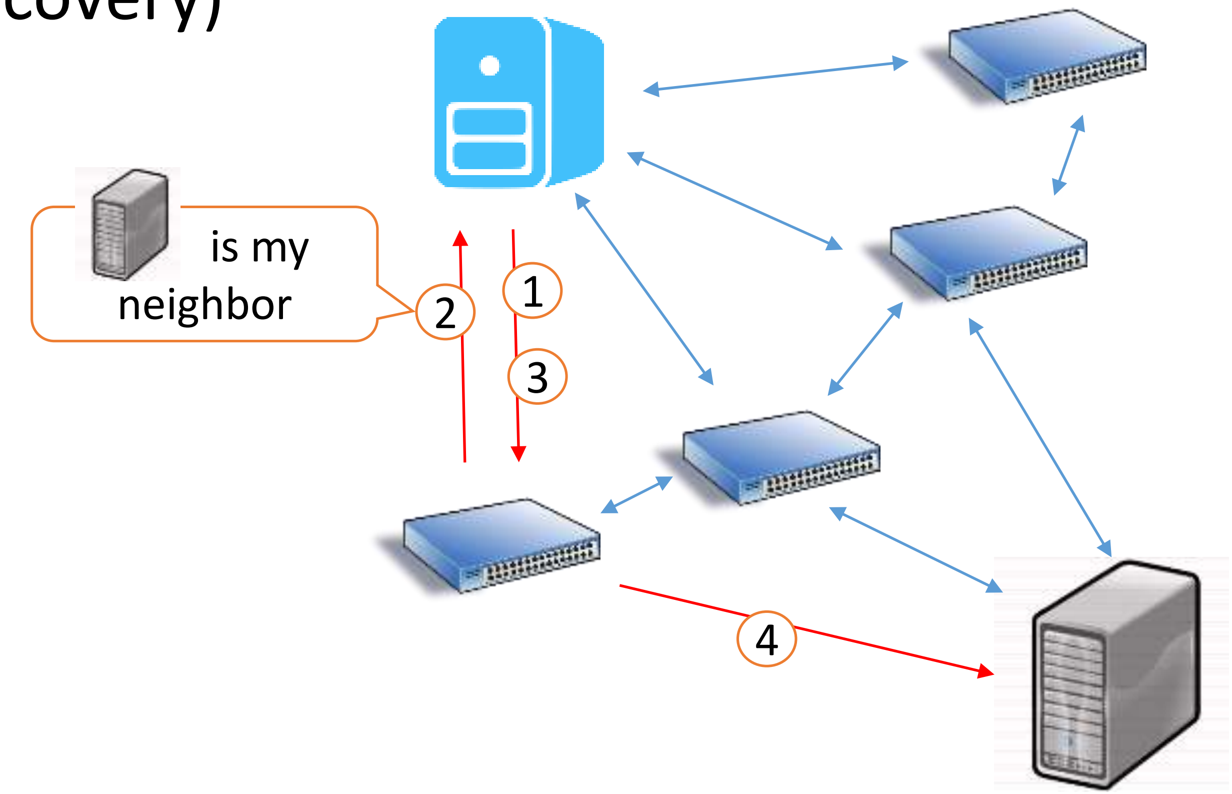
Roadmap

- 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)

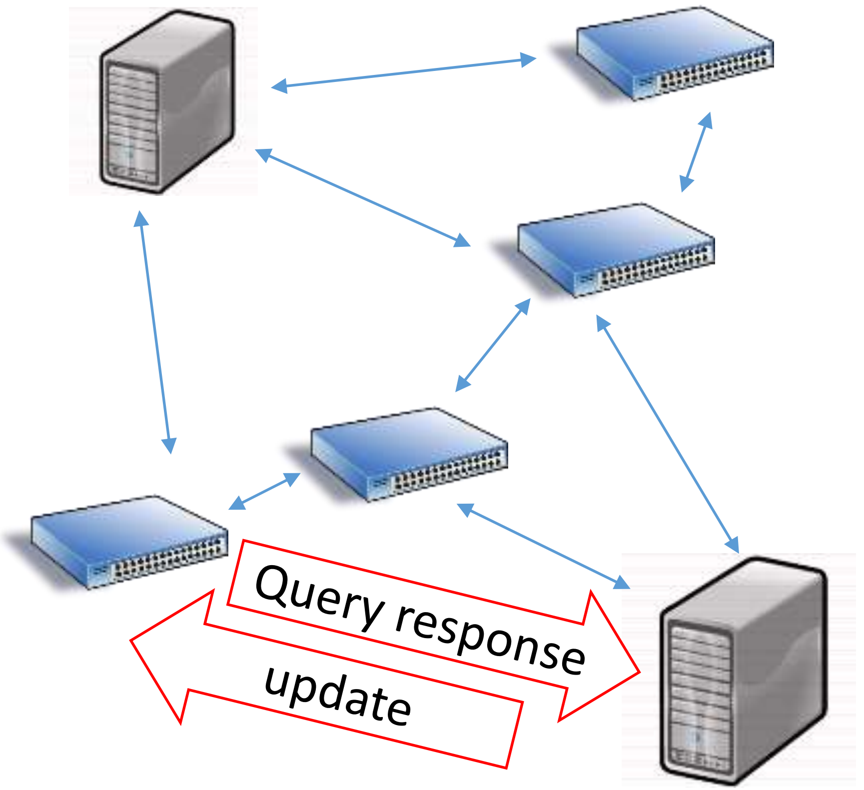


Renaissance: Self-Stabilizing, distributed, in-band control plane

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

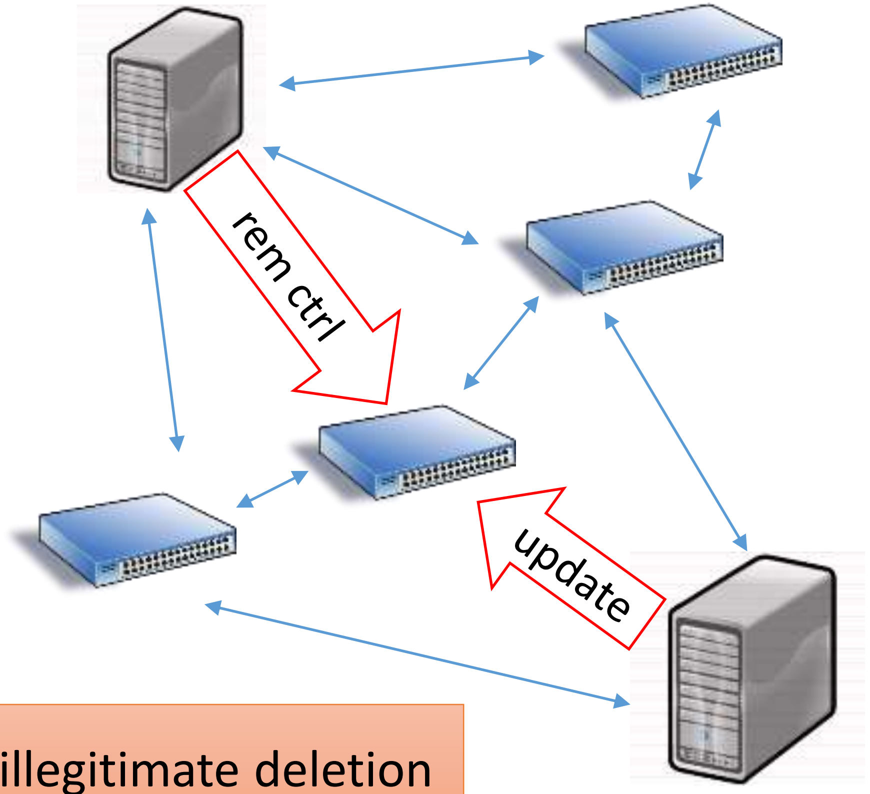


Renaissance: Self-Stabilizing, distributed, in-band control plane

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



illegitimate deletion

Roadmap

- Algorithm
- Proof highlights
- Evaluation

Self-Stabilizing Distributed SDN Control Plane

<u>Algorithm sketch for controller p_i</u>

keep a set of responses for each discovered node

curr and prev synchronization round tags

do-forever

    remove stale responses (unreachable nodes)

    if topology is discovered **then** start new synch round

    for each reachable switch p_j **do**

        if new round started **then** remove unreachable controller info

        update p_i's info (manager set an rule set) on p_j

    for every node p_k **do** query p_k and send updates if p_k is a switch

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

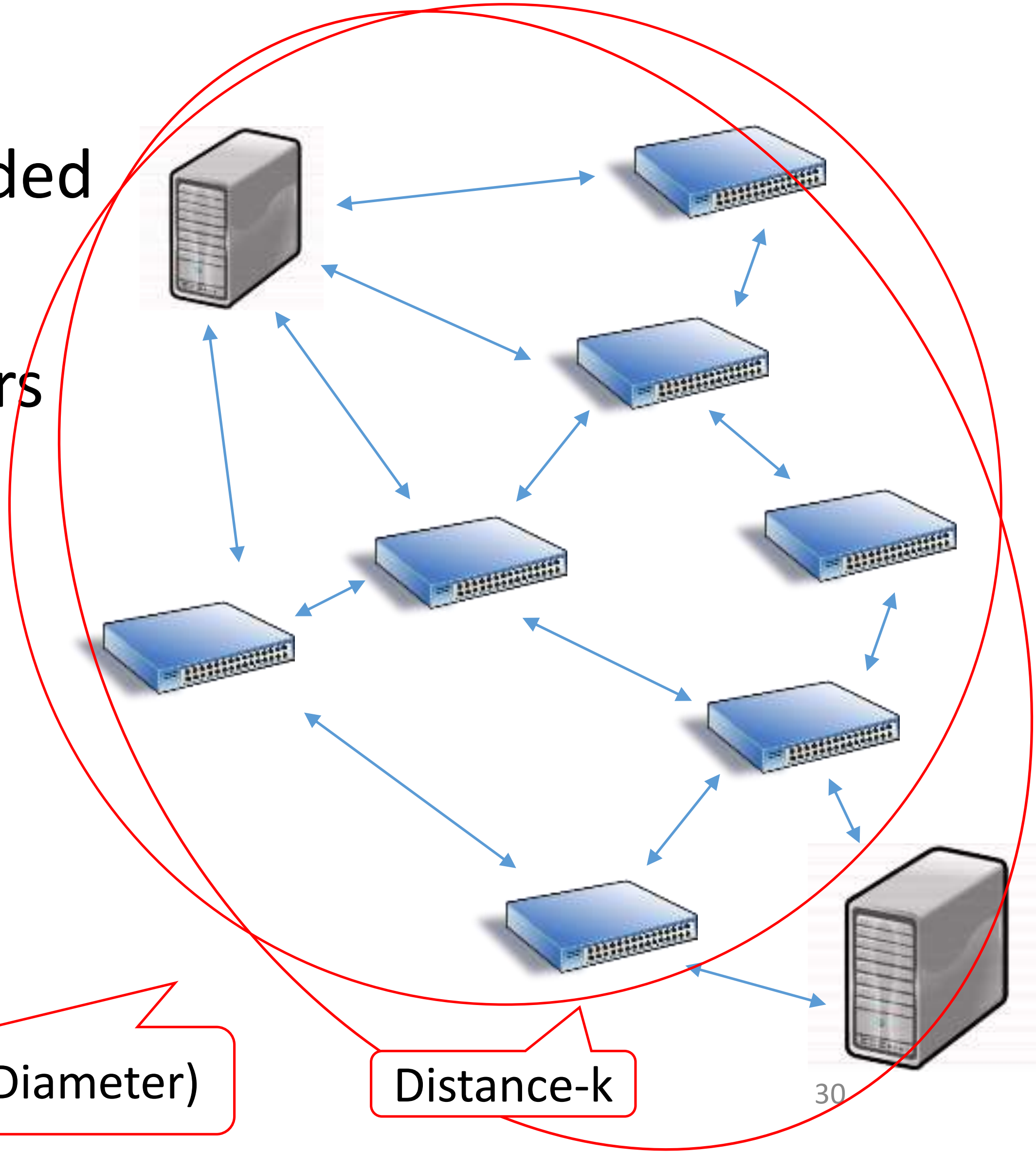
Can also tolerate topological changes after recovery in *O*(maxDiameter)

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

#nodes = #controllers + #switches

Bounded number of illegitimate deletions

- Wipe controller memory if bounds exceeded
- FIFO updates of switch memories
- Proof by induction on distance k-neighbors from each controller:
  - Within (ck+1) rounds: no illegitimate deletions for at most distance k-neighbors
  - c: constant – depends on link capacity
  - k = maxDiameter



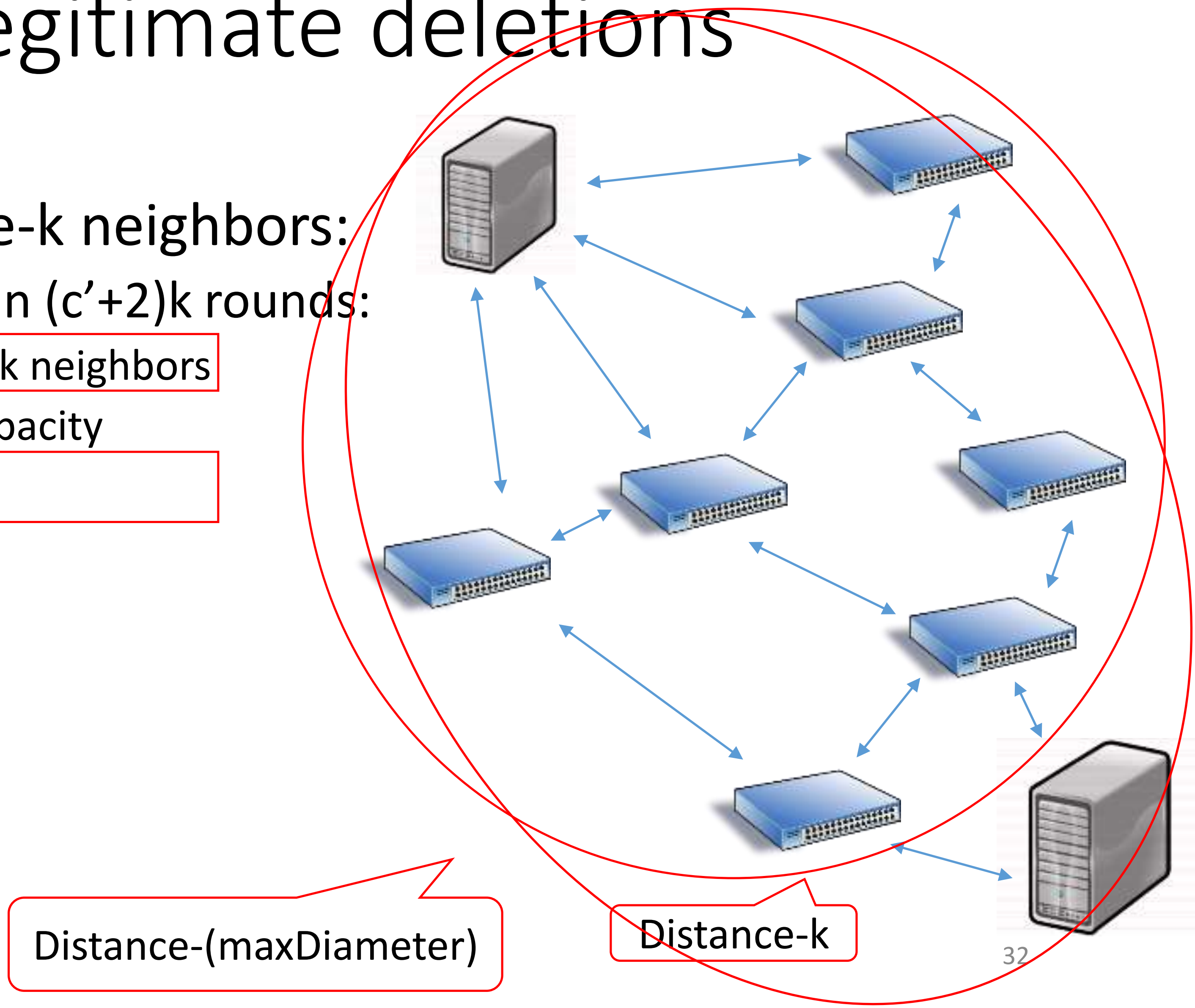
Bounded number of illegitimate deletions

Intuition: at the k-th roundtrip a controller discovers nodes of distance k+1

- k=1: controller knows distance 1 topology – 1 roundtrip for query return
- k to k+1: when query from distance-k neighbor returns, controller learns about distance k+1 neighbors

Recovery from transient faults, in the absence of illegitimate deletions

- Transient fault recovery,
  proof by induction on distance-k neighbors:
  - If no illegitimate deletion, within (c'+2)k rounds:
    - correctness for at most distance-k neighbors
    - c': constant – depends on link capacity
    - k = maxDiameter



Recovery from transient faults, in the absence of illegitimate deletions

- k=1: first roundtrip is for the query, second is for installing the correct state
- k to k+1: as in base case
- If illegitimate deletion: switch has again an incorrect state!

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)

Setup

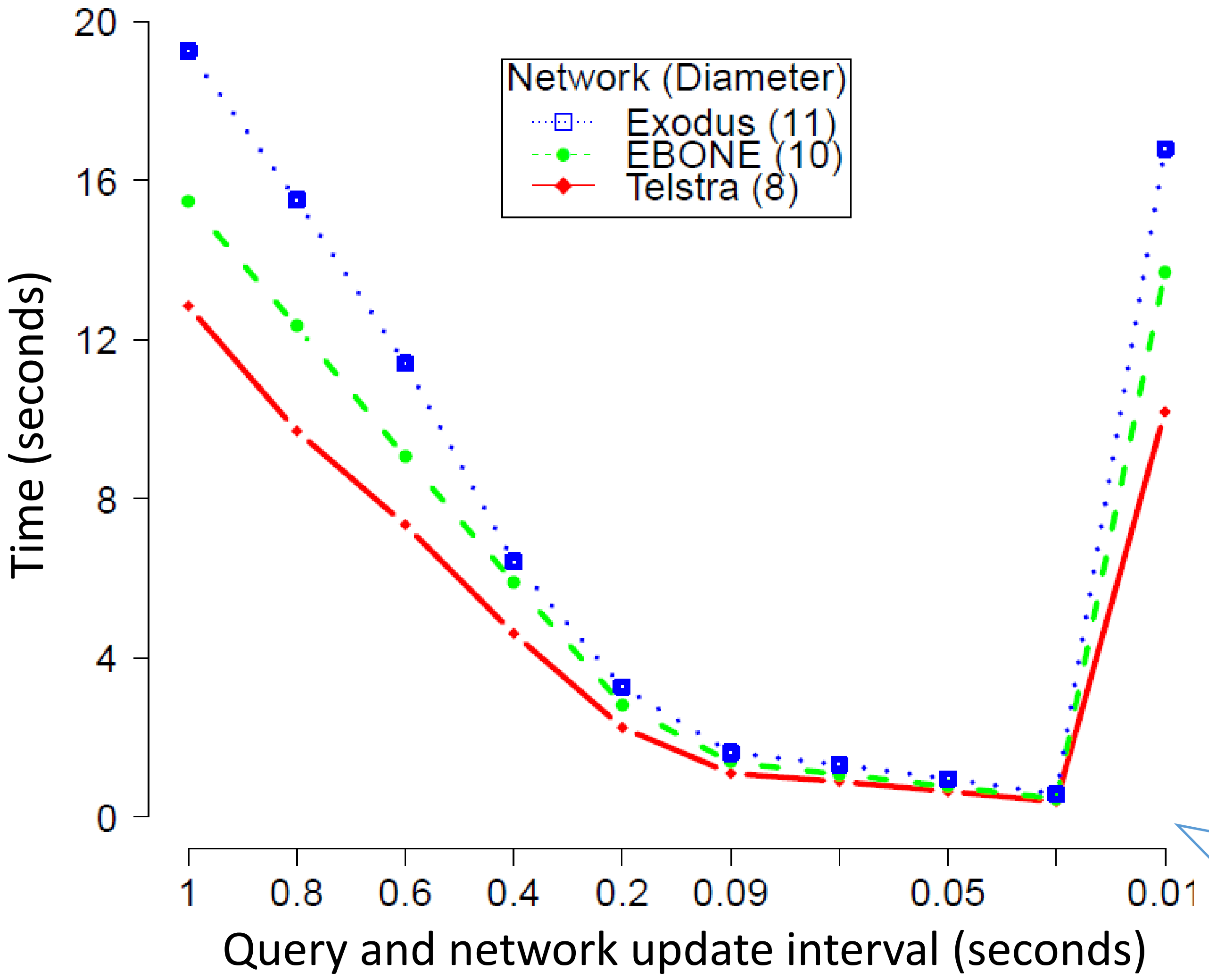
- PCs with Ubuntu 16.04.1 OS, Intel Core i7-2600K CPU at 3.40GHz (8CPUs) with 16GB RAM.

- 1 controller req or flow installation per second paths according to BFS OpenFlow fast-failover groups for backup paths

- Hosts for traffic and RTT (round-trip delay time) evaluation are placed such that their distance is max

- Standard SDN topologies (B4, Clos, Rocketfuel networks). B4 and Clos 3 controllers, Rocketfuel up to 7 controllers, up to ~200 nodes in total.

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

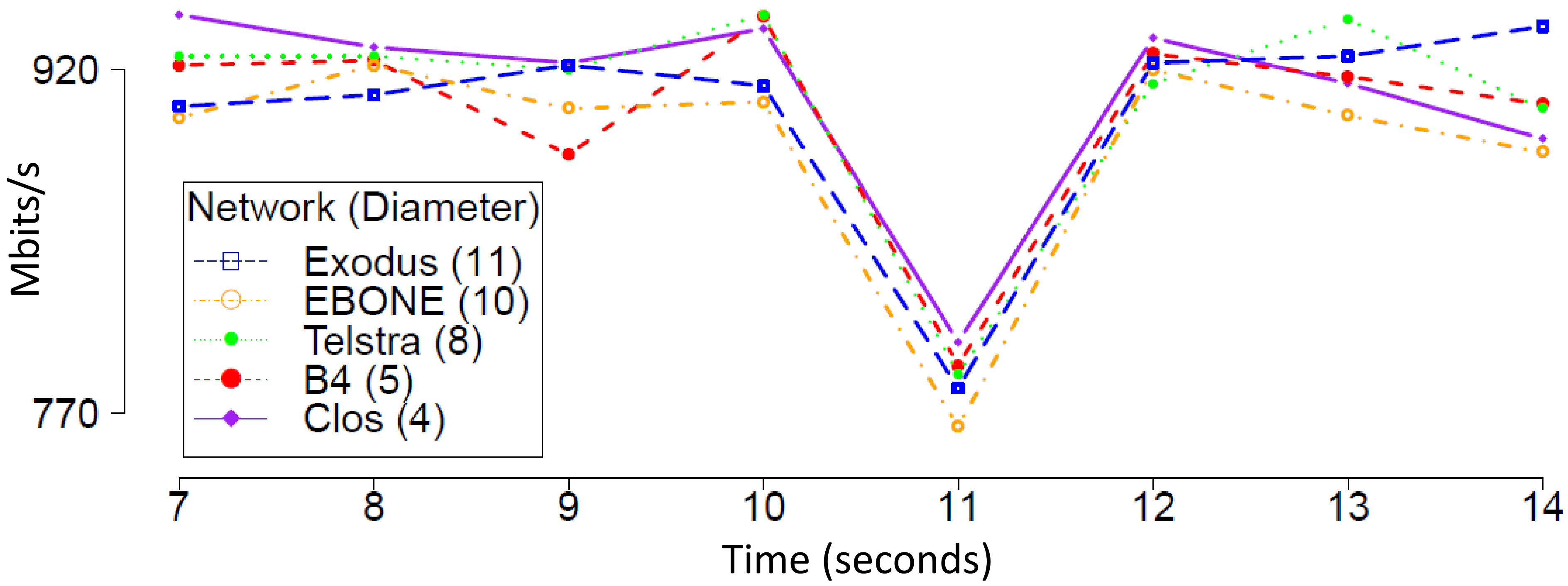
How efficiently does Renaissance recover in the presence of link and node failures?

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

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