Network centralization and programmability

Marco Ruffini: marco.ruffini@tcd.ie

Part II Module Overview

- Week1:
  - Lecture: Network centralization and programmability
  - Lab: Mininet API and flow tables
- Week2:
  - Lecture: Software Defined Networking and Openflow
  - Lab: OpenFlow QoS commands and POX controller
- Week 3:
  - Lecture: Working with a controller
  - Lab: Start working on assignment1
- Week4:
  - Lecture: Moving to OpenFlow hardware
  - Lab: Assignment1 marking and Start working on assignment2, moving to OpenFlow Hardware.
- Week5:
  - Test: in class test on course material and lab
  - Lab: finalise Assignment 2 and marking
- Marking:
  - 2.5 x 2 labs = 5
  - In class test = 15
  - 2 x Assignments (marked live) = 20 + 10

Preparatory material for labs

- This is homework: prepare it before the first lab on Thursday
- Basically install Virtualbox, mininet Ubuntu-based image and other tools (ssh and X11)
- Run some basic mininet commands to see that it works

- Notice that Virtualbox does not work on M1/M2 MAC chips yet.

Lecture content

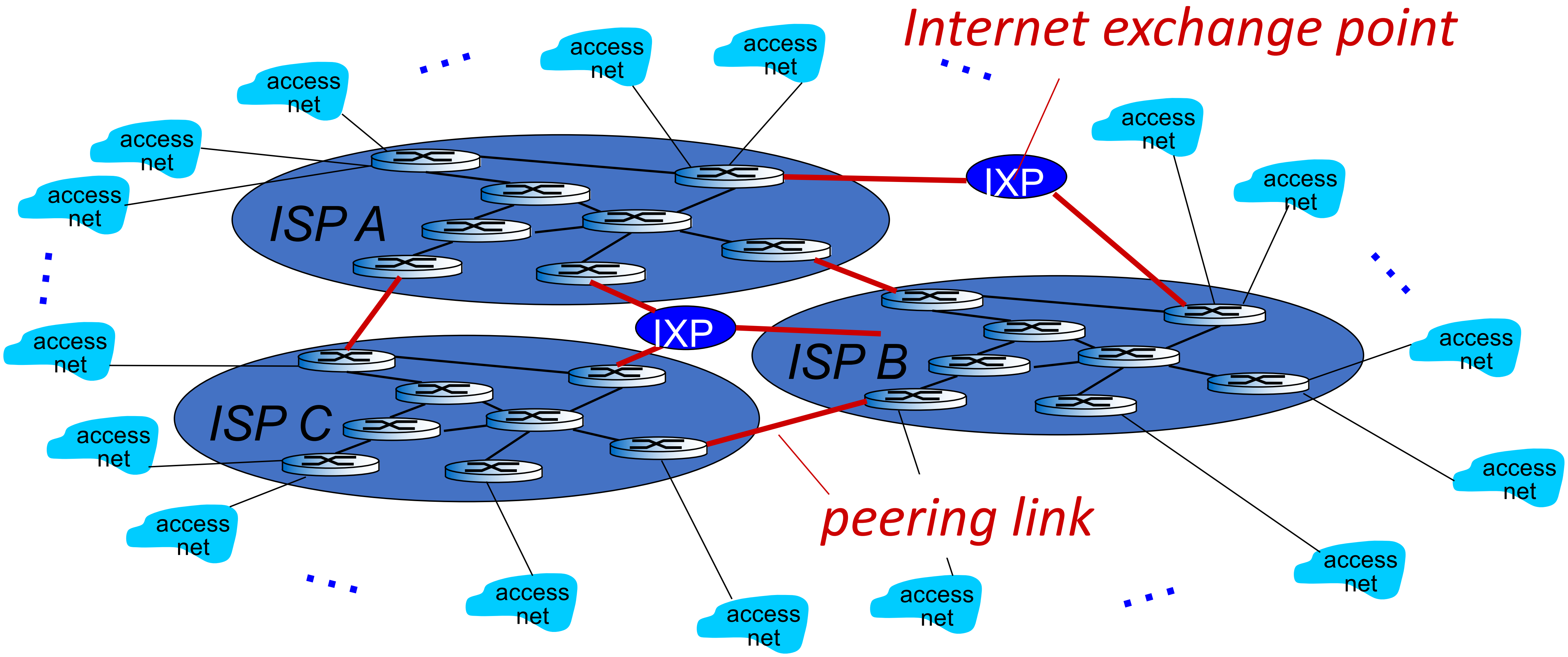
- From Distributed to centralized networks
  - Intra-As and Inter-AS routing
  - Network control and optimization
  - Protocol centralization

- Network programmability
  - Control and data plane separation
  - OpenFlow protocol
  - Software Defined Networking

From distributed to centralized networks

Internet structure: network of networks

- The Internet is a complex interconnection of heterogeneous devices.
- Interconnection of multiple independent Autonomous Systems (AS)



Intra-AS routing

- Within AS, protocols used to be distributed:
  - Open Shortest Path First (OSPF), Routing Information Protocol (RIP)

- Routers share information on the state of their link and their connectivity using two different types of protocols:
  - Link state routing: routers tell every one (broadcast) about their close neighbours
  - Path vector routing: routers tell their neighbours about routes that involve many other routers

- Decisions on actual route can be made on number of hops and link weight (i.e., can associate with it other metrics, such as physical distance, congestion, etc.)

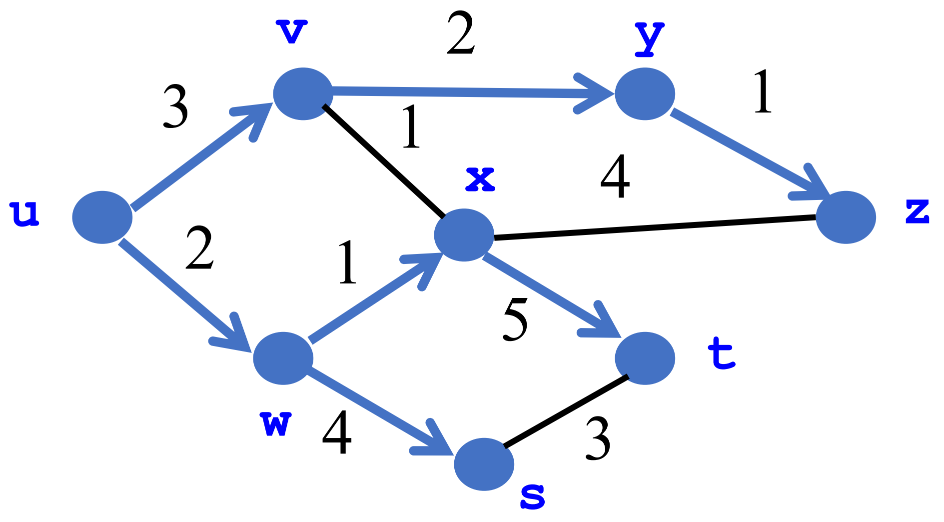
- In principle there is freedom for as As to choose the routing within its own network

Inter-AS routing

- The main Internet protocol is the Border Gateway Protocol (BGP)
- It operates routing across Autonomous Systems
- Features:
  - It’s a distributed protocol
  - AS border routers send information about addresses that are reachable through them (prefix announcements, such as 134.54.0.0/16)
  - Announcements also contain list of other AS the message has passed and next hop AS
  - Route decision based on policy first (i.e., preferred AS, etc), then on hop count

Distributed Control Plane

- Link-state routing: OSPF, IS-IS
  - Flood the entire topology to all nodes
  - Each node computes shortest paths
  - Dijkstra's algorithm



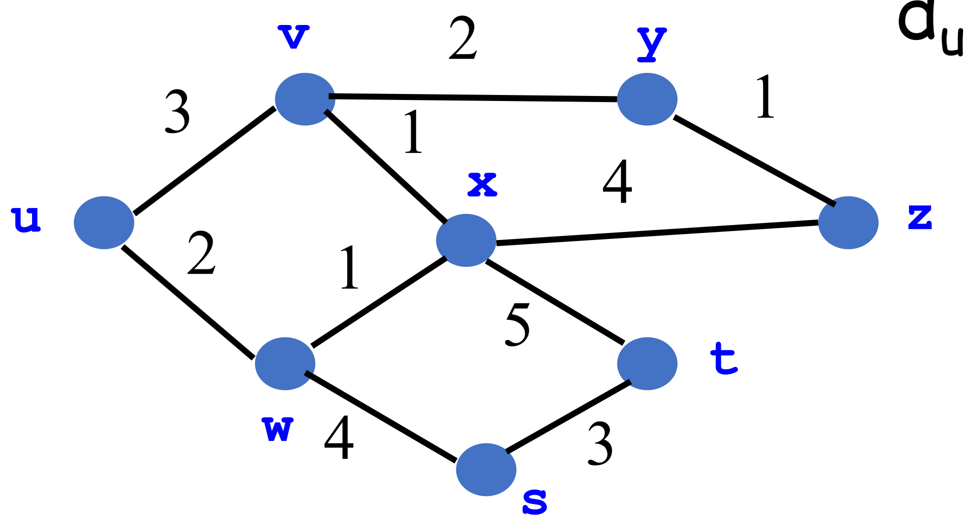
U's table

	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

9

Distributed Control Plane

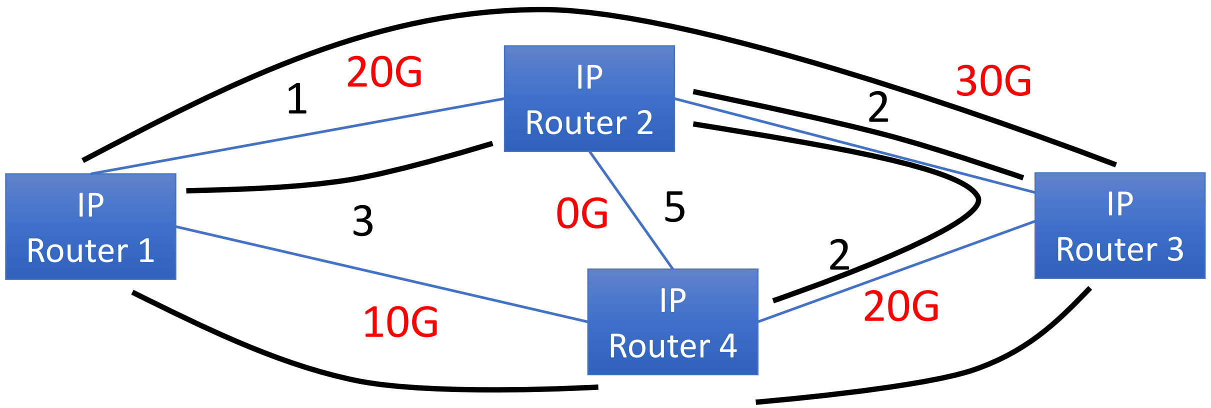
- Distance-vector routing: RIP, EIGRP
  - Each node computes path cost
  - … based on each neighbors' path cost
  - Bellman-Ford algorithm



$$d_u(z) = \min\{c(u,v) + d_v(z),\ c(u,w) + d_w(z)\}$$

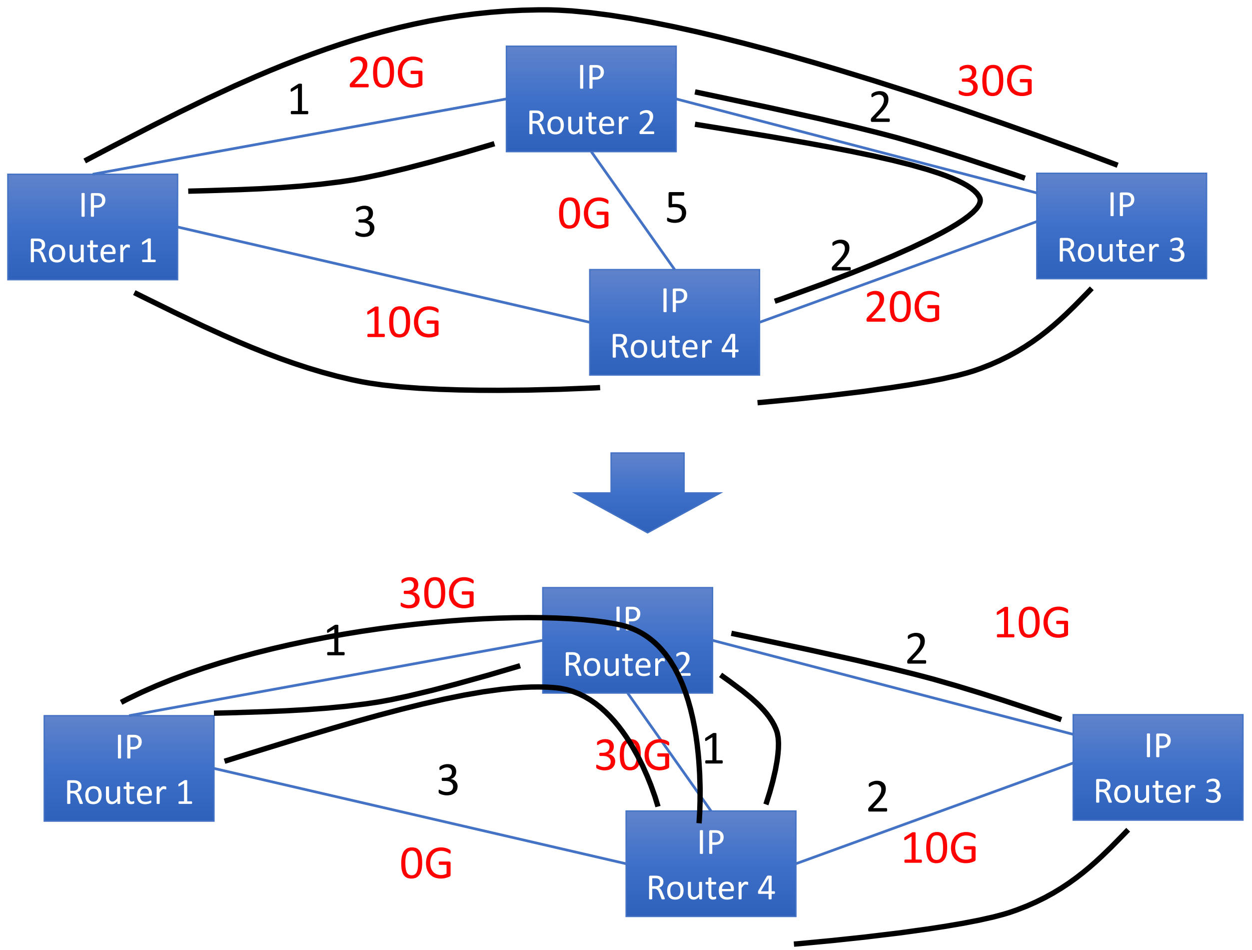
Network optimisation issue

- Today an operator can run complex centralized network optimization algorithms to spread load across network, minimise latency, packet loss, etc.
- If the protocol is distributed, then the route is decided by a distributed algorithm, that only considers hop count and weight
- The operator could tweak the weight metric, but this method is indirect and can cause unforeseen issues..



- Imagine links max capacity per link is 30Gb/s, before congestion occurs
- Imagine each flow (black line) carries 10G traffic
- Say I want IP2 to take a shorter route towards IP4, and avoid link IP2->IP3 that is congested
- I can reduce the weight of link IP2->IP4, which has no traffic

Problem optimizing with distributed algorithms



- I now end up with two congested links

Protocol centralisation

- It thus make sense to centralise the protocol:
  - The optimal routes are centrally calculated by an optimization algorithm (centralization = global view and global decision making)
  - The decision mechanism uses a protocol to distributed routing information

➔ The decision is made centrally

➔ The protocol only distributes decision information to the routers

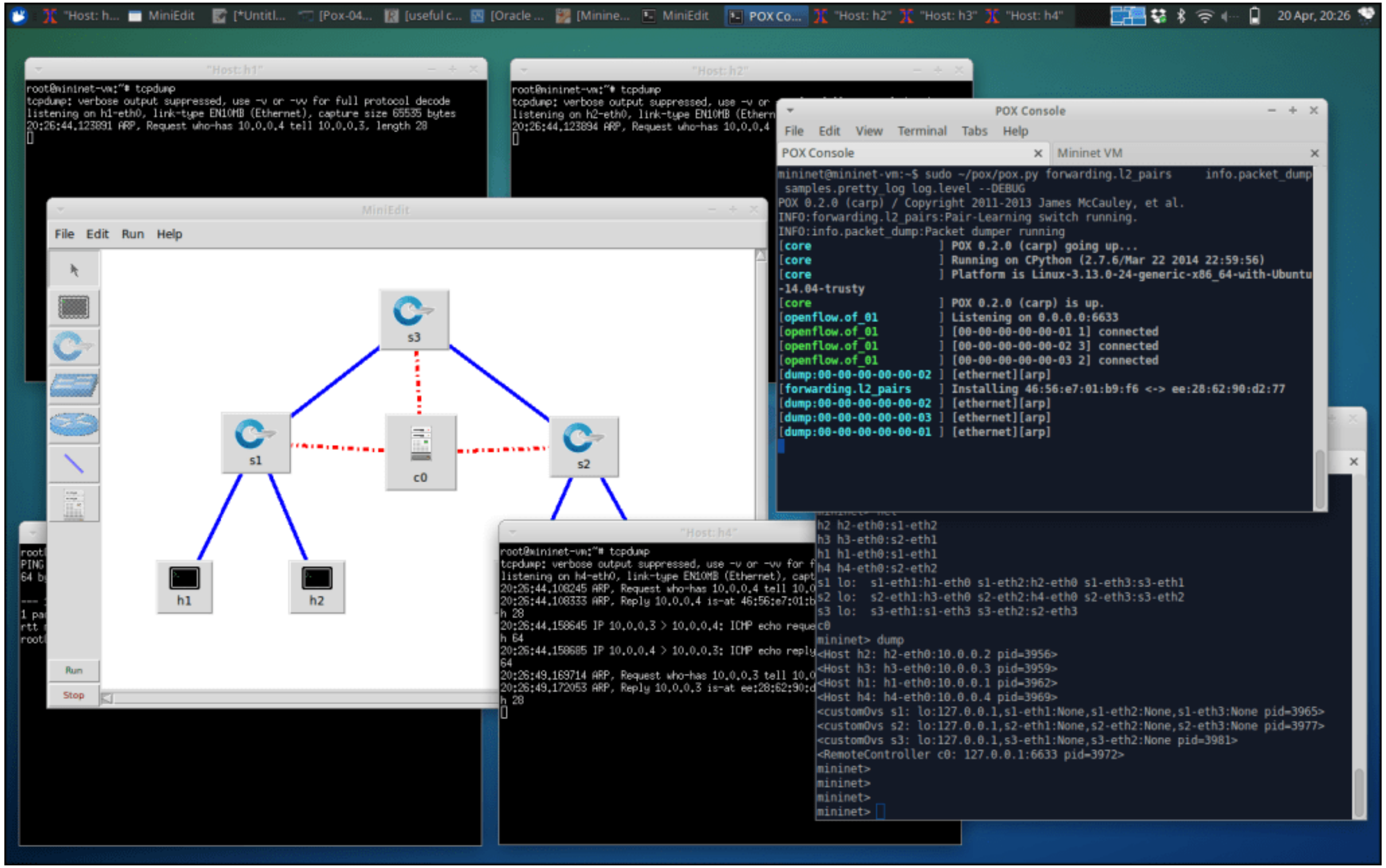
➔ Routers don’t make any decision, just update their routing table (except for cases of fast protection against failures).

Examples: MPLS, Carrier Ethernet, etc.

Question

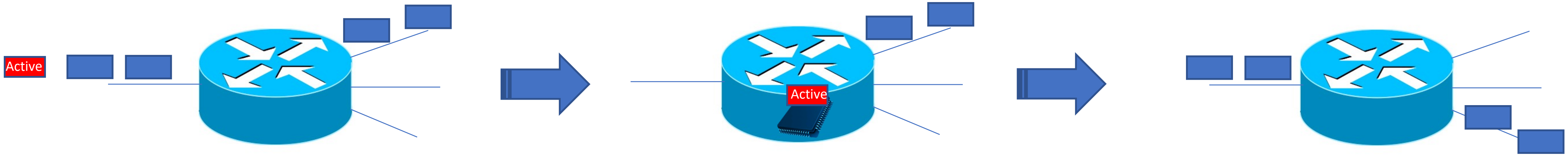
- If centralisation has strong advantages, why is inter-AS routing (BGP) still distributed?

Network Programmability



History of programmable networks: 1) Active Networks

- Ideas on programmable networks date back to the late ’90
- Active networks was one of the first (SwitchWare, Smart Packets, NetScrpt):
  - Network nodes would expose resources through programming interfaces
  - Packets entering nodes could carry code that would change the behaviour of the router
  - This could be achieved in band (capsule model) or out of band (programmable router model)

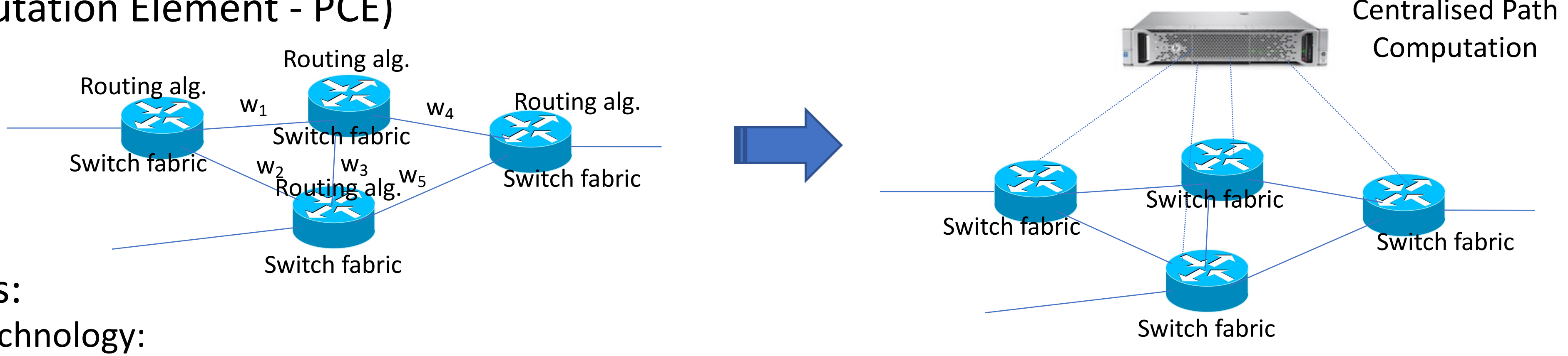


- Interesting to examine drivers (compare them to SDN):
  - Technology: cost reduction of processing power in routers, new programming languages
  - Usage: reduce long provisioning times of legacy technology, unify control of middle boxes (firewalls, proxies,…)

Source: N. Feamster, J. Rexford and E. Zegura. The Road to SDN. ACM Networks, 11(12), Dec. 2013

History of programmable networks: 2) Separation of control and data planes

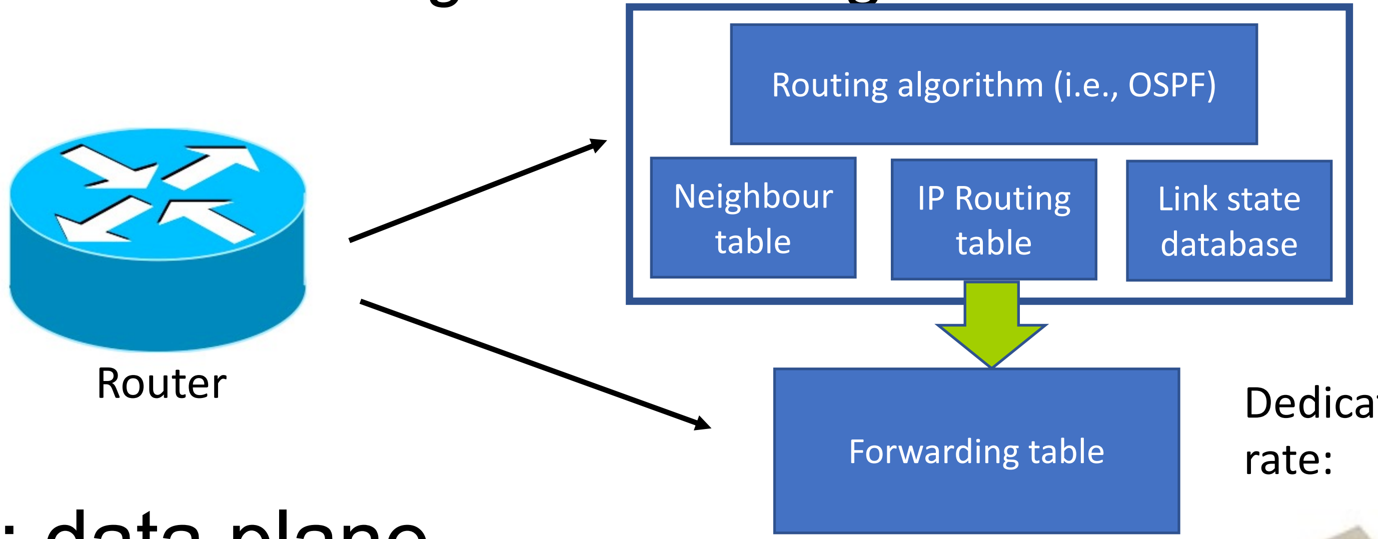
- This occurred at the beginning of decade 2000s
- Development of <u>open interfaces</u> between control and data plane (e.g., IETF Forwarding and Control Element Separation - ForCES)
- Development of <u>centralized network control</u> (Routing Control Platform - RCP, SoftRouter, Path Computation Element - PCE)



- Drivers:
  - Technology:
    - hardware (ASIC) data plane implementation already started separation with software control plane;
    - Commodity GPP has more memory and power than dedicated router processors
  - Use: demand for higher capacity leads operators to seek more control for traffic engineering
  - Users requiring services such as VPNs..
    - Legacy techniques of changing link weight to influence distributed routing protocols suboptimal and unpredictable
    - MPLS being developed to create tunnels that could be centrally organized

Forwarding vs. Routing

- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router *creating* a forwarding table



Typically software running on CPU

Dedicated chip running at line rate:

Broadcom's Jericho2c+ provides 14.4 Tb/s

- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table

History of programmable networks: 3) OpenFlow

- OpenFlow succeeded in trade-off between full programmability and real-world deployment
  - Use of existing switch hardware (merchant silicon)
    - Fast moving move from academic to industry/application environment
  - API between control plane software (opensource as NOX, POX) and switch hardware:
    - Allowed basic functionalities such as: Pattern matching, counters, actions…
    - Several developments since first release in 2009, to include meter tables, pipeline of tables, addition of optical interfaces, etc.

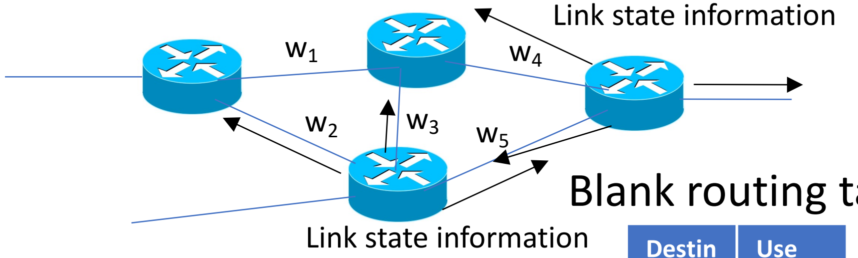
- Drivers:
  - Technology:
    - availability of merchant silicon (same chip for several manufacturers)
    - OpenFlow initially only required firmware upgrade
  - Use: Data centre community could use software engineers to write network control code (gained more control and quicker development time)
    - Controlled environment (homogeneous network, one domain, controlled traffic)

Source: N. Feamster, J. Rexford and E. Zegura. The Road to SDN. ACM Networks, 11(12), Dec. 2013

OpenFlow example

- Distributed approach:

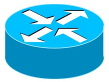
Link state broadcast



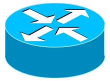
Blank routing table

Destination	Use port

Shortest path calculation



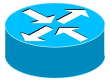
Shortest path calculation



function Dijkstra(*Graph*, *source*):

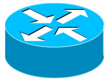
create vertex set Q 4 5
for each vertex *v* in *Graph*:
……………..

Routing table update



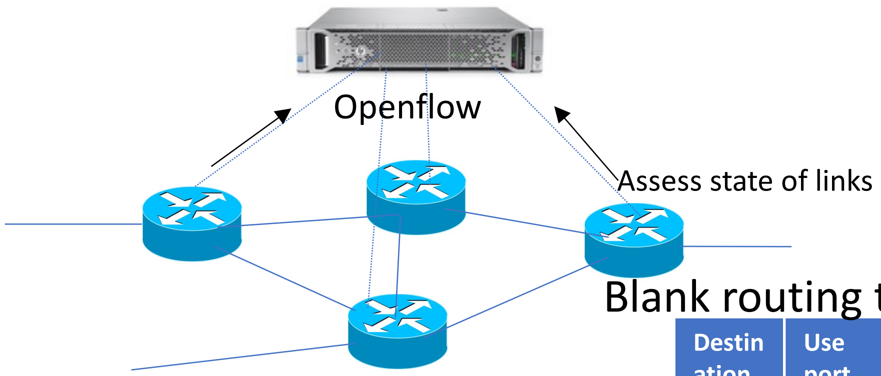
Destination	Use port
A	1
B	2
C	2

Routing table update



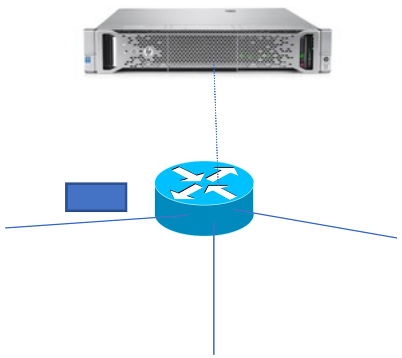
Destination	Use port
A	3
B	3
C	1

- Openflow centralized (reactive) approach

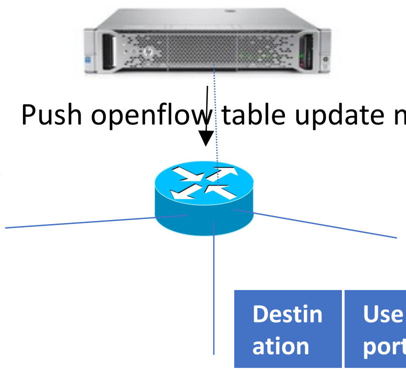


Blank routing table

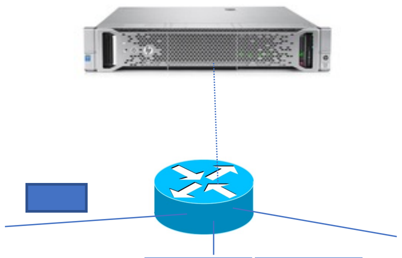
Destination	Use port



Push openflow table update message



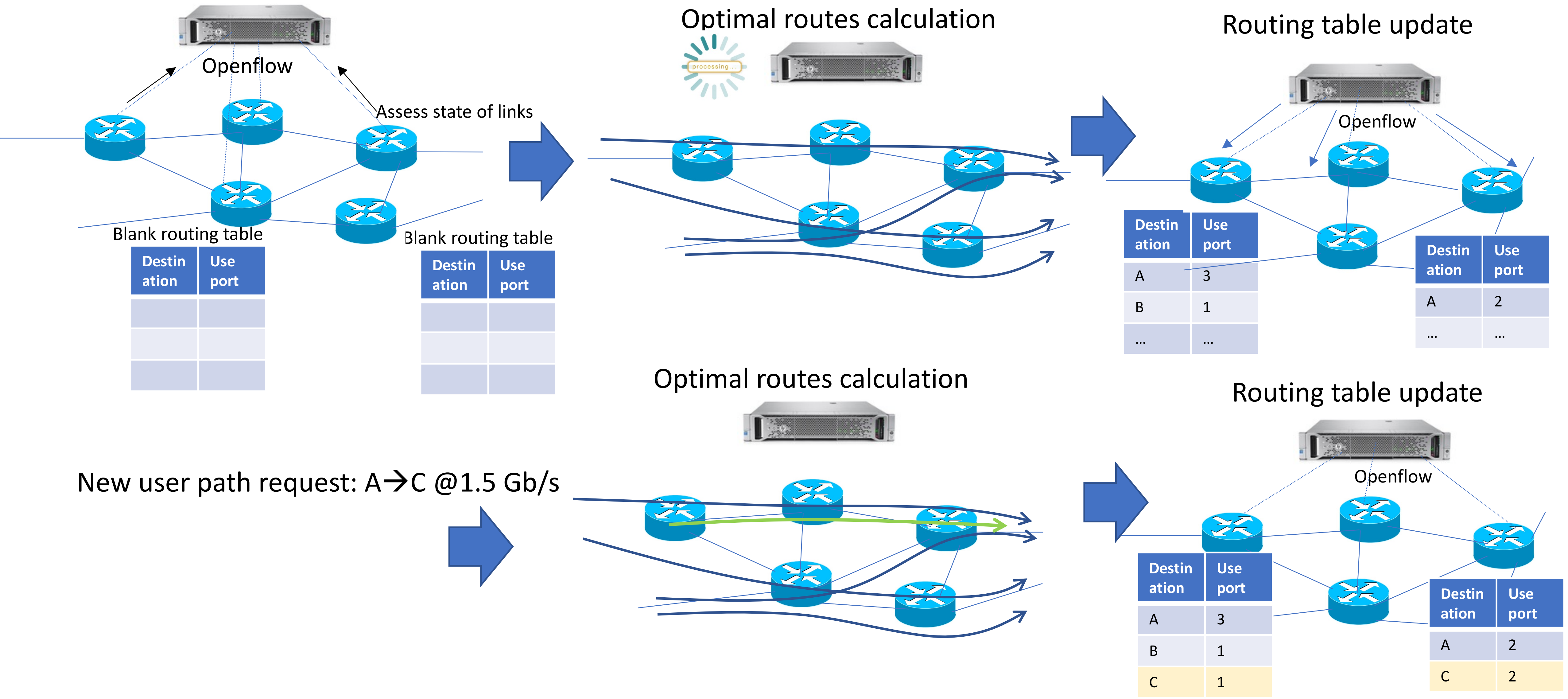
Destination	Use port
A	3



Destination	Use port
A	3

OpenFlow example

- Openflow centralized (proactive) approach



Reactive vs. Proactive (pre-populated)

Both models are possible with OpenFlow

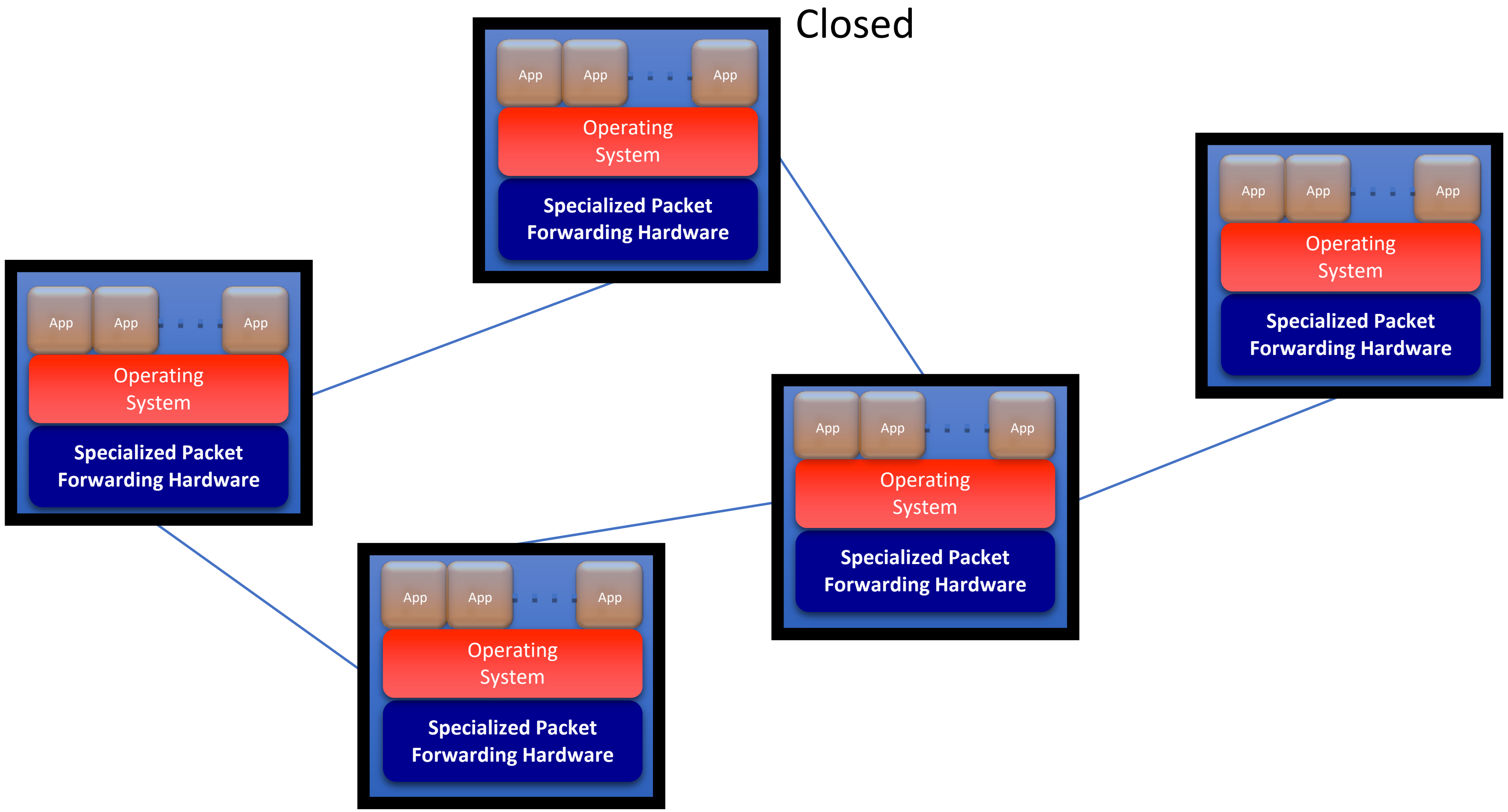
Reactive

- First packet of flow triggers controller to insert flow entries
- Efficient use of flow table
- Every flow incurs small additional flow setup time
- If control connection lost, switch has limited utility

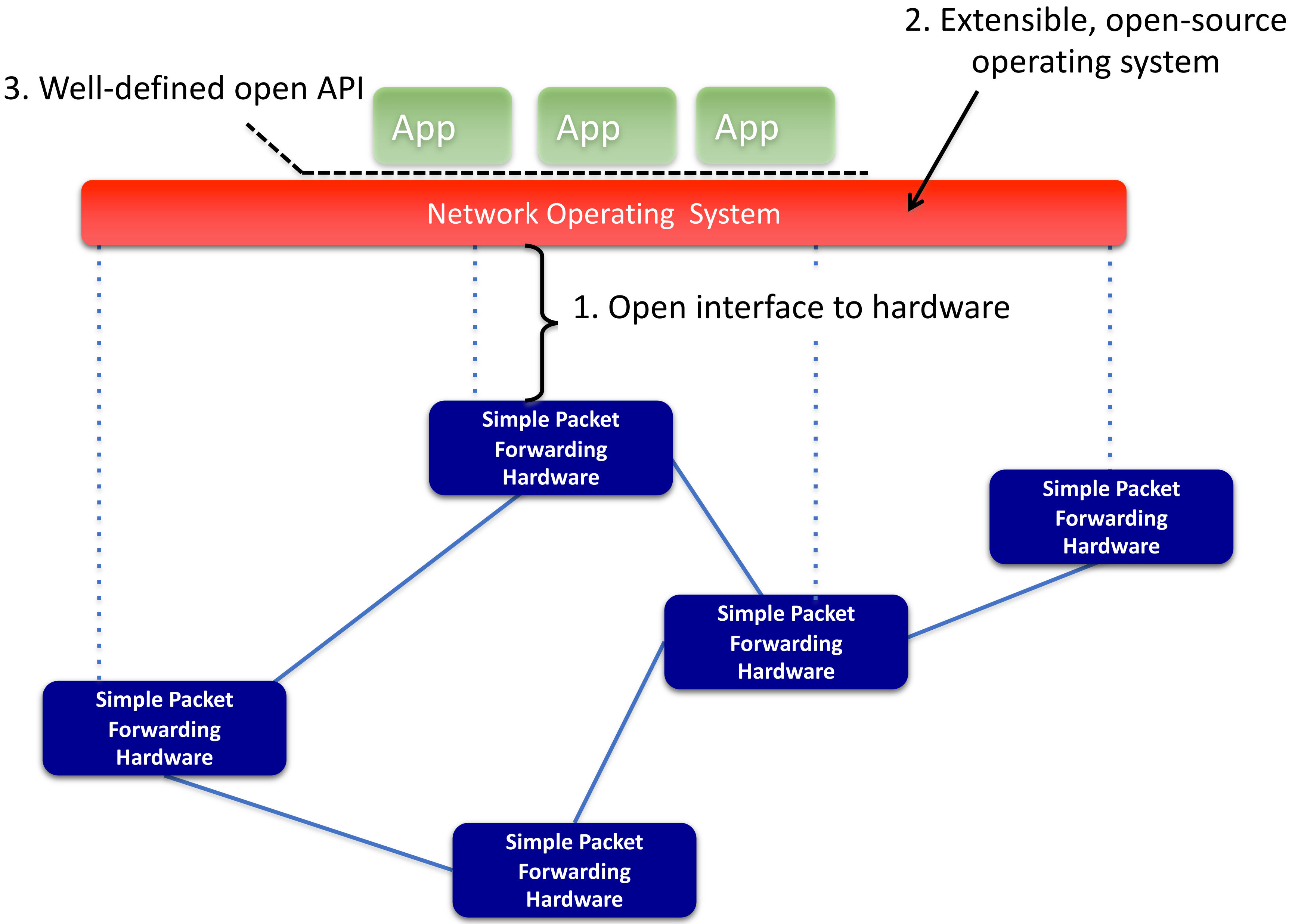
Proactive

- Controller pre-populates flow table in switch
- Zero additional flow setup time
- Loss of control connection does not disrupt traffic
- Essentially requires aggregated (wildcard) rules

SDN means going from here…



… to here



2. Extensible, open-source operating system
3. Well-defined open API
App
App
App
Network Operating System
1. Open interface to hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware
Simple Packet Forwarding Hardware