UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

Software-Defined Networking (SDN) & Network Function Virtualization (NFV)

Future Internet Networks (FINE)

Master in Innovation and Research in Informatics (MIRI)

Jordi Perelló Muntan, academic year 2017-2018

What is Software-Defined Networking (SDN)?

- Term initially used in year 2009 in a scientific article about the OpenFlow Project @ Stanford University
- Its most accepted definition was elaborated later on by the Open Networking Foundation (ONF), a large non-profit operator-lead consortium dedicated to accelerating the adoption of SDN & NFV:
  - *"SDN is the **physical separation** of the **network control plane** from the **forwarding plane**, and where a **control plane controls several devices**"*
- In other words, ONF suggests for SDN to pull the intelligence off the network devices (switches, routers, ...), logically centralizing it on a separate device (i.e., SDN controller), able to control multiple network devices
- The ultimate goal behind SDN is to make network control directly *programmable*, while abstracting the details of the physical infrastructure from network services and applications

How do current network devices operate?

- They integrate all forwarding and control functions (i.e., planes) necessary to deliver frames/packets to destination:
  - **Forwarding functions**: responsible for moving frames/packets from I/O interfaces (e.g., packet buffering, forwarding table lookup, MAC address table lookup, actual packet forwarding)
  - **Control functions**: responsible for deciding how frames/packets should be moved (e.g., routing protocols, IP table, ARP)
- Forwarding functions must be executed very fast (on a per packet basis!), hence implemented in hardware (ASICs). Control functions do not have such stringent execution time requirements, hence implemented in CPU
- Additionally, network devices offer management functions to enable the configuration of control plane functions (i.e., management plane), although always limited to the specific vendor solution capabilities

(Some) Limitations of traditional networking

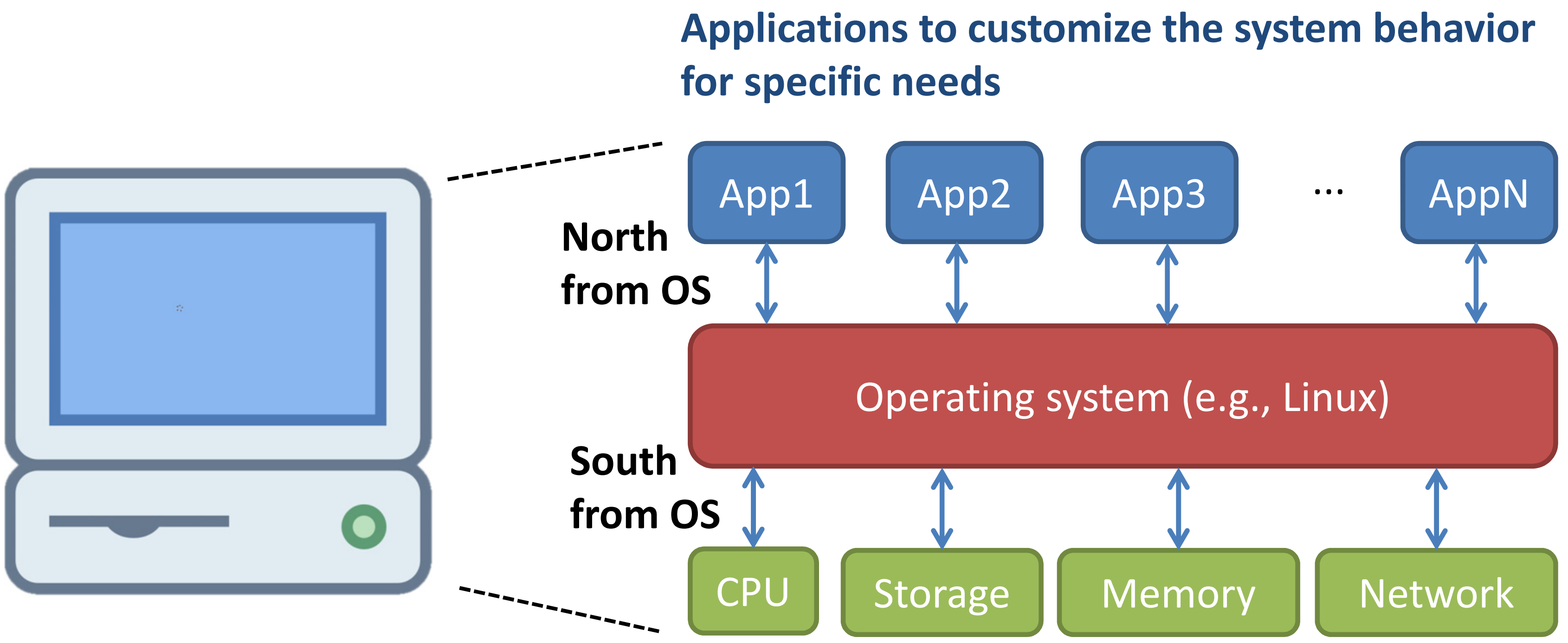
1. Vendor-specific configuration procedures: problematic as networking devices from different vendors are deployed
2. Each networking device has to be configured manually and independently: tedious and error prone task as the network size grows up
3. Distributed control protocols (e.g., routing protocols) can easily lead to inconsistencies as traffic dynamics raise up in the network
4. Too static to face unpredictable traffic patterns that require additional capacity only at certain times (only overprovisioning is possible)
5. Limited reusability of dedicated network appliances when the network scenario characteristics change
6. Very limited (or inexistent) possibilities to alter control protocols behavior to improve or modify it, i.e., most innovation is mission impossible

Can these limitations be solved with SDN?

1. SDN introduces standardized interfaces (e.g., OpenFlow protocol) to interact with network devices from different vendors in a unified way, enabling the configuration of a wide range of network features (TE, QoS, security, etc.)
2. SDN centralizes equipment configuration decisions in the so-called SDN controller, which can be automated based on software (network apps)
3. The SDN controller enjoys a global network view, which simplifies network decisions and prevents inconsistencies typical from distributed protocols
4. The SDN controller can trigger the automatic setup of network services on-demand, e.g., when additional bandwidth is required (and release them when not needed anymore)
5. SDN with NFV enables the mobility of VNFs and the reusability of the hardware when not needed anymore (introduced later on!)
6. SDN enables the programmability of new network applications, so as to define its behavior as desired by the network operator

SDN in analogy with a computer OS (1/2)

- The computer operating system model can be drawn in three basic layers: hardware, operating system and applications

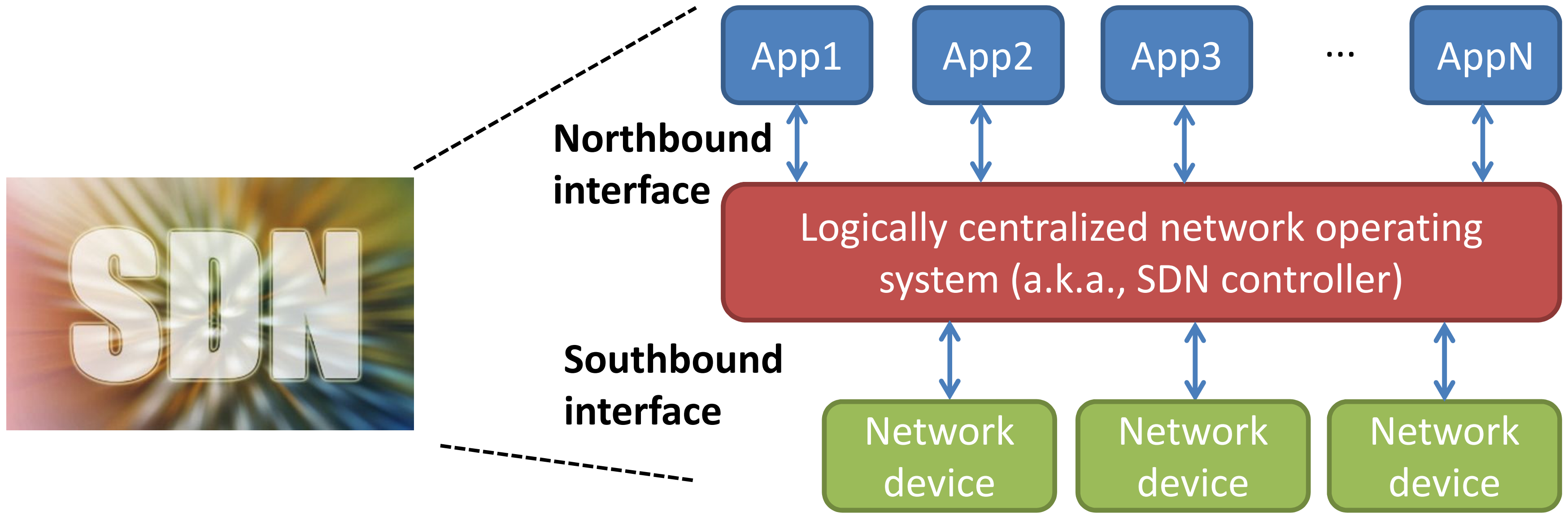

Applications to customize the system behavior for specific needs
App1
App2
App3
...
AppN
North from OS
Operating system (e.g., Linux)
South from OS
CPU
Storage
Memory
Network

The OS can be seen as a middleware permitting applications to access HW resources, based on policies

SDN in analogy with a computer OS (2/2)

- Similarly, the SDN model can also be split into 3 different layers:

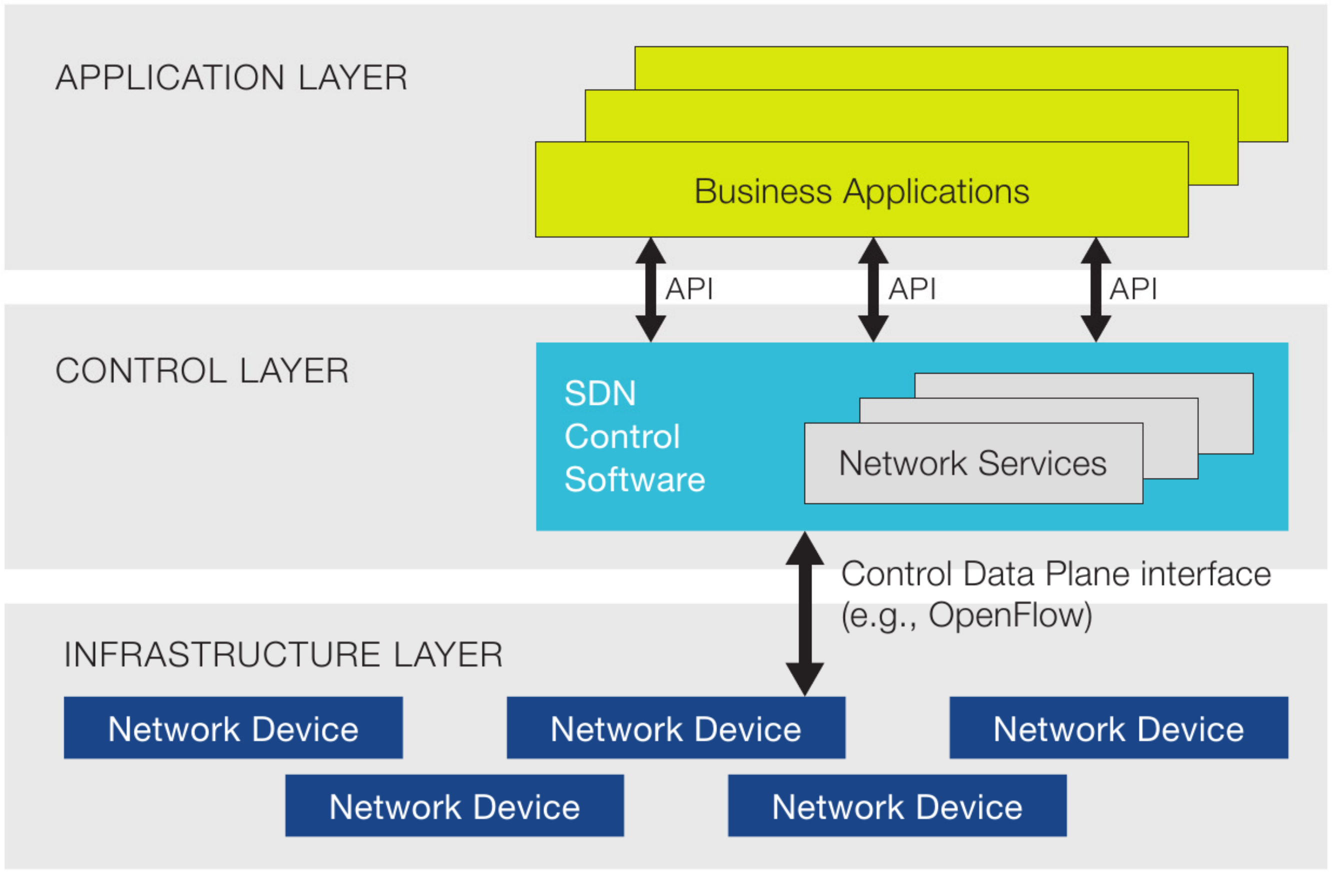
Applications to customize the network behavior for specific needs



The SDN controller provides services to automatically manage network devices, as well as a programmable interface (API) to the network applications

SDN architecture (1/2)

- As can be seen, the SDN architecture proposed by the ONF has a lot in common with previous pictures!



SDN architecture (2/2)

- Application layer (**i.e., extensions/plug-ins to the network brain**)
  - Contains SDN applications that decide on the network behavior, communicating it to the SDN controller via the northbound interface
  - Applications typically work upon (easier to manage!) abstracted network information exposed to them by the SDN controller
- Control layer (**i.e., the network brain**)
  - Contains the logically centralized SDN controller
  - The SDN controller receives instructions from the SDN applications and forwards them to the network devices via the southbound interface
  - The SDN controller also runs core network services (e.g., topology discovery, network resource inventorying, simple path computation, etc.)
- Infrastructure layer (**i.e., dumb network equipment**)
  - Implemented by the network devices that receive and forward data from source to destination on a per-flow basis

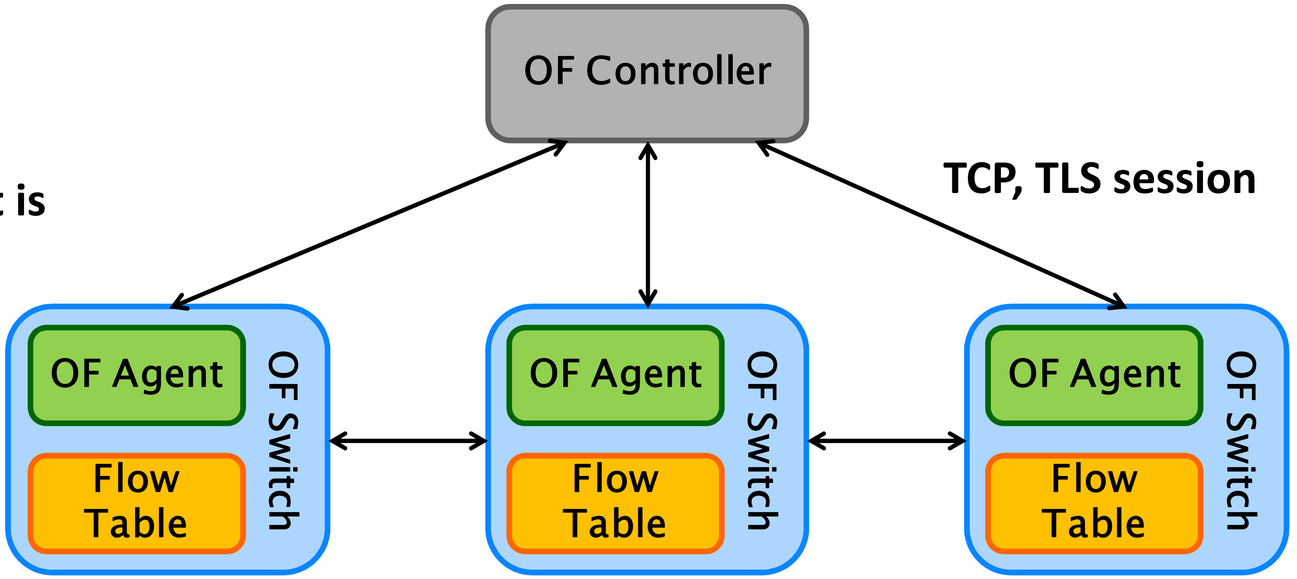
OpenFlow

- OpenFlow is considered one of the first SDN standards initially proposed by Stanford University. First OpenFlow article:
  - N. McKeown, et al., “OpenFlow: enabling innovation in campus networks”, ACM SIGCOMM Computer Communication Review, 38(2), Apr. 2008
- Since that moment, the ONF has remained in charge of the OpenFlow standardization for implementing the SDN controller southbound interface, right from its initial version (v1.0) in 2009 (current version v1.5, 2014)
- From its very beginning, OpenFlow already fostered the physical separation between network devices and their control functions, adopted later on in the SDN architecture:
  - OpenFlow Ethernet switches, whose physical characteristics are abstracted by a flow table
  - OpenFlow controller that configures flow tables of OpenFlow switches

OpenFlow architecture

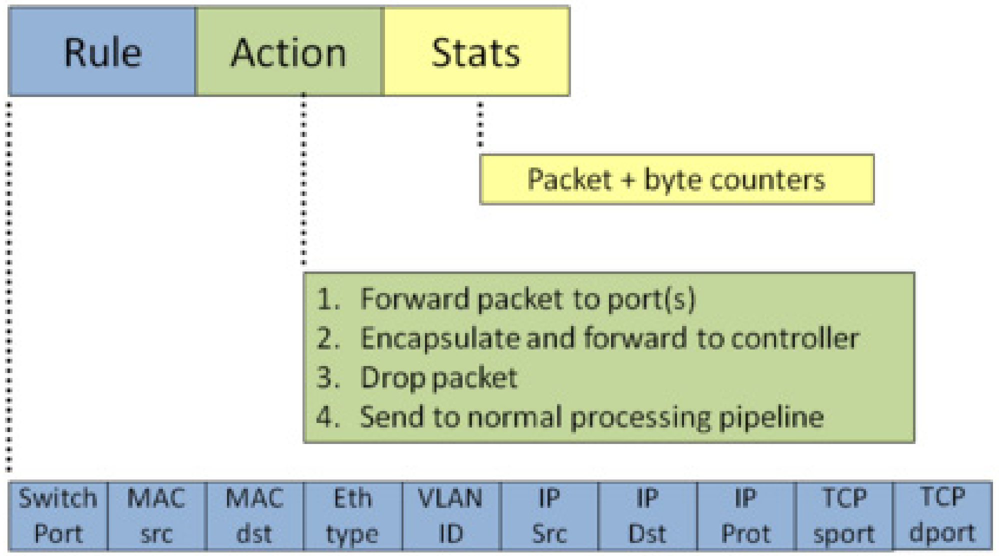
- In an OpenFlow-enabled network, OpenFlow switches communicate with the OpenFlow controller via the OpenFlow protocol
- This protocol permits to control the flows by deciding the routes their packets follow and the processing they receive
- The datapath of an OpenFlow Switch consists of a Flow Table, where an action is associated with each flow entry

The OF Agent is the one that understands OpenFlow messages



OpenFlow flow table (1/2)

- The flow table(s) in an OpenFlow switch can be populated with multiple flow-entries (entry = flow), where each one has 3 fields:
  - The rule defining the flow (to identify packets that belong to the flow)
  - The action that defines how flow packets are processed, basically:
    - ➢ Forward packets of the flow to a given port(s)
    - ➢ Encapsulate packets of the flow and send them to the controller
    - ➢ Drop packets of the flow (e.g., for security reasons)
  - Statistics to keep track of the flow behavior (e.g., to remove inactive ones)



OpenFlow flow table (2/2)

- An OpenFlow switch permits L2 switching (MAC address or VLAN), L3 routing (IP address) and even flow switching:

Switch port	MAC source	MAC dest	Ether type	VLAN ID	IP source	IP dest	IP prot	TCP sport	TCP dport	Action
*	*	**00:1f...**	*	*	*	*	*	*	*	**to Port3**

Switch port	MAC source	MAC dest	Ether type	VLAN ID	IP source	IP dest	IP prot	TCP sport	TCP dport	Action
*	*	*	*	*	*	**80.58...**	*	*	*	**to Port2**

Switch port	MAC source	MAC dest	Ether type	VLAN ID	IP source	IP dest	IP prot	TCP sport	TCP dport	Action
Port8	*	*	*	**vlan3**	*	*	*	*	*	**to Port6**

Switch port	MAC source	MAC dest	Ether type	VLAN ID	IP source	IP dest	IP prot	TCP sport	TCP dport	Action
Port2	**00:1f...**	**00:2a...**	**0800**	**vlan1**	**80.58..**	**92.54...**	**4**	**2789**	**80**	**to Port12**

Reactive vs. proactive flow-entries in OpenFlow

- **<u>Reactive flow-entries</u>**: configured in the OpenFlow switches by the controller upon arrival of the first packet of a new flow
  - ➢ It is assumed that the OpenFlow controller has previously learnt the network topology, i.e., where devices are located (detailed later on)
  - ➢ Packets of the flow have to be stored at an input buffer until the flow-entry is properly added in the flow table
  - ➢ The controller must be located near the switches to avoid excessive latency during flow setup (RTTs between switches-controller)
- **<u>Proactive flow-entries</u>**: configured in the OpenFlow switches by the controller before the traffic of the flow is actually transmitted
  - ➢ When packets of the flow arrive at the switch, they can be directly forwarded to the specific output port without any controller intervention
  - ➢ The controller does not have to be located near the switches
  - ➢ But the network looses automation ...

Reactive flow setup example (1/2)

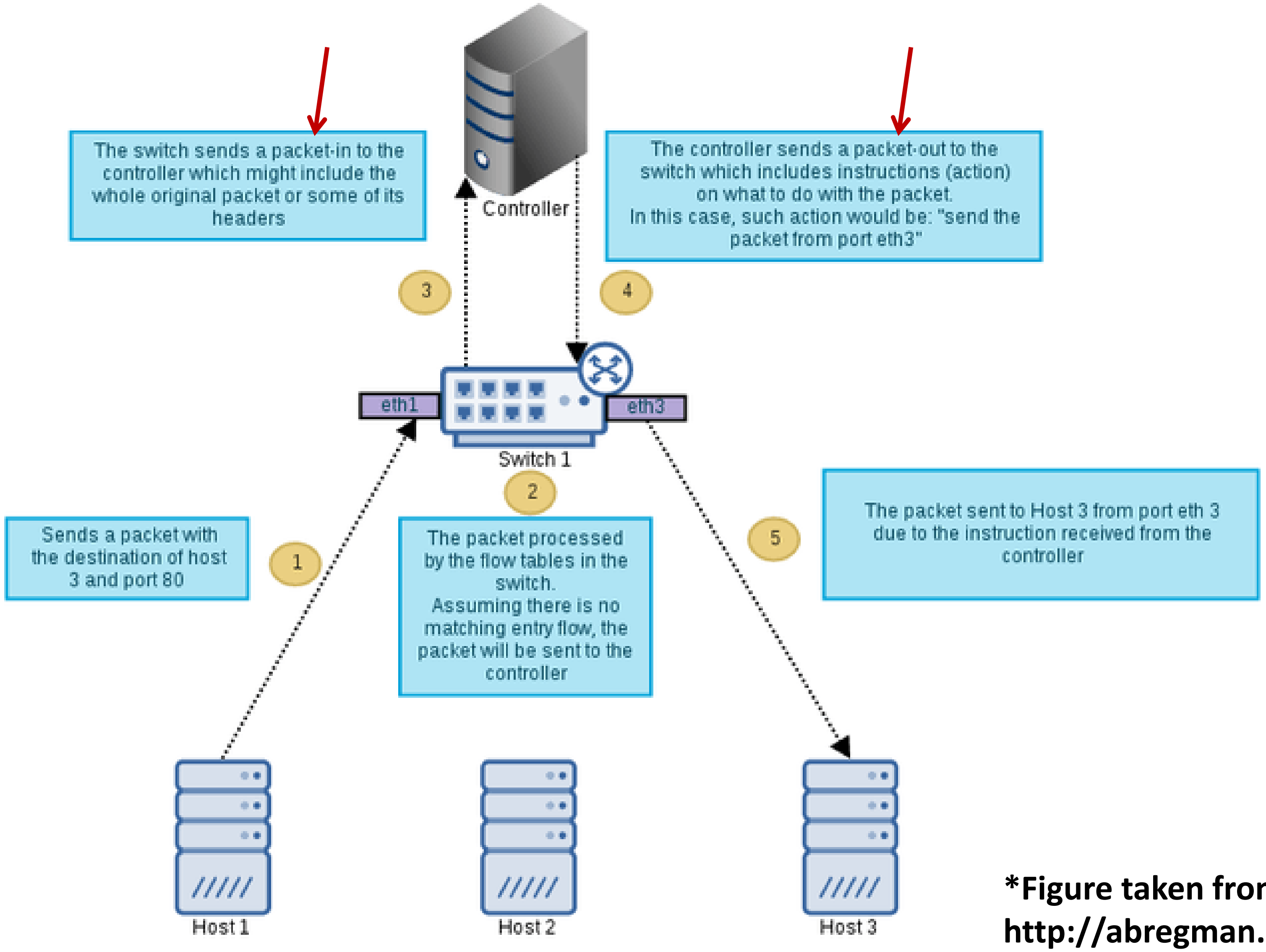

The switch sends a packet-in to the controller which might include the whole original packet or some of its headers
Controller
The controller sends a packet-out to the switch which includes instructions (action) on what to do with the packet.
In this case, such action would be: "send the packet from port eth3"
3
4
eth1
eth3
Switch 1
2
Sends a packet with the destination of host 3 and port 80
1
The packet processed by the flow tables in the switch.
Assuming there is no matching entry flow, the packet will be sent to the controller
5
The packet sent to Host 3 from port eth 3 due to the instruction received from the controller
Host 1
Host 2
Host 3

***Figure taken from http://abregman.com**

Reactive flow setup example (2/2)

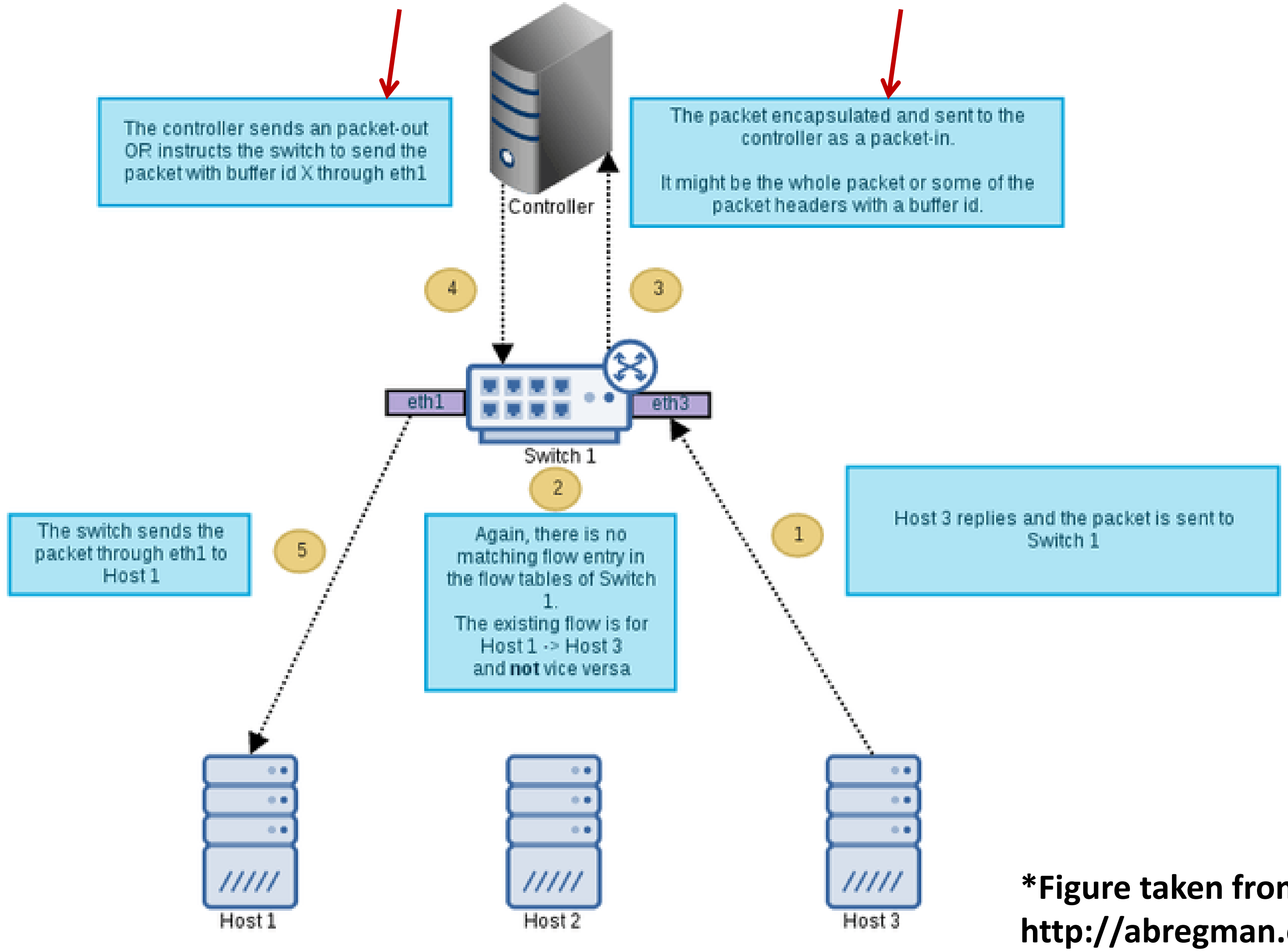
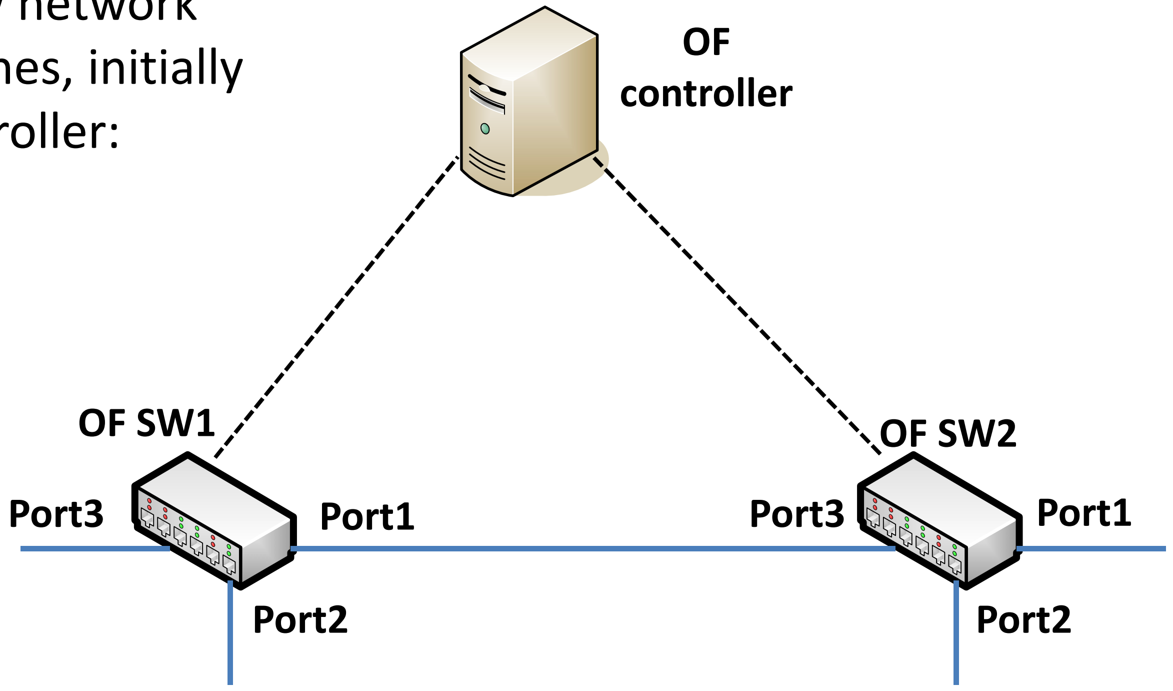

The controller sends an packet-out OR instructs the switch to send the packet with buffer id X through eth1
Controller
The packet encapsulated and sent to the controller as a packet-in.
It might be the whole packet or some of the packet headers with a buffer id.
4
3
eth1
eth3
Switch 1
2
The switch sends the packet through eth1 to Host 1
5
Again, there is no matching flow entry in the flow tables of Switch 1.
The existing flow is for Host 1 -> Host 3 and **not** vice versa
1
Host 3 replies and the packet is sent to Switch 1
Host 1
Host 2
Host 3

***Figure taken from http://abregman.com**

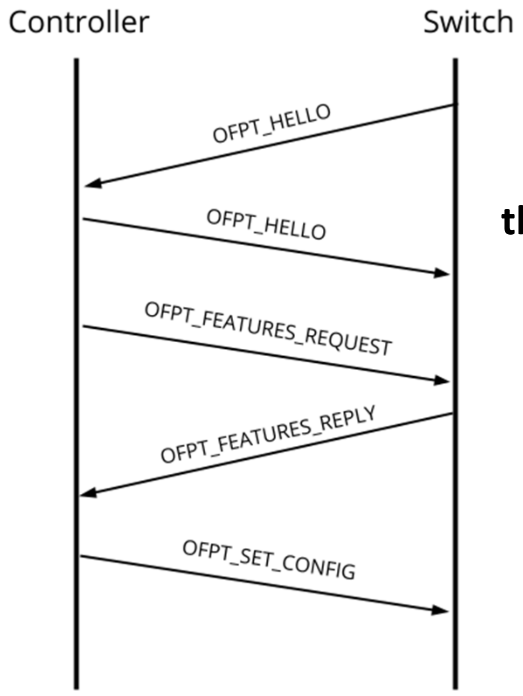
OpenFlow discovery protocol (1/4)

- OpenFlow Discovery Protocol (OFDP) is the de-facto protocol used by the controllers to discover the underlying topology, which makes use of Link Layer Discovery Protocol (LLDP) frames used in Ethernet
- Imagine an OpenFlow network scenario with 2 switches, initially unknown by the controller:

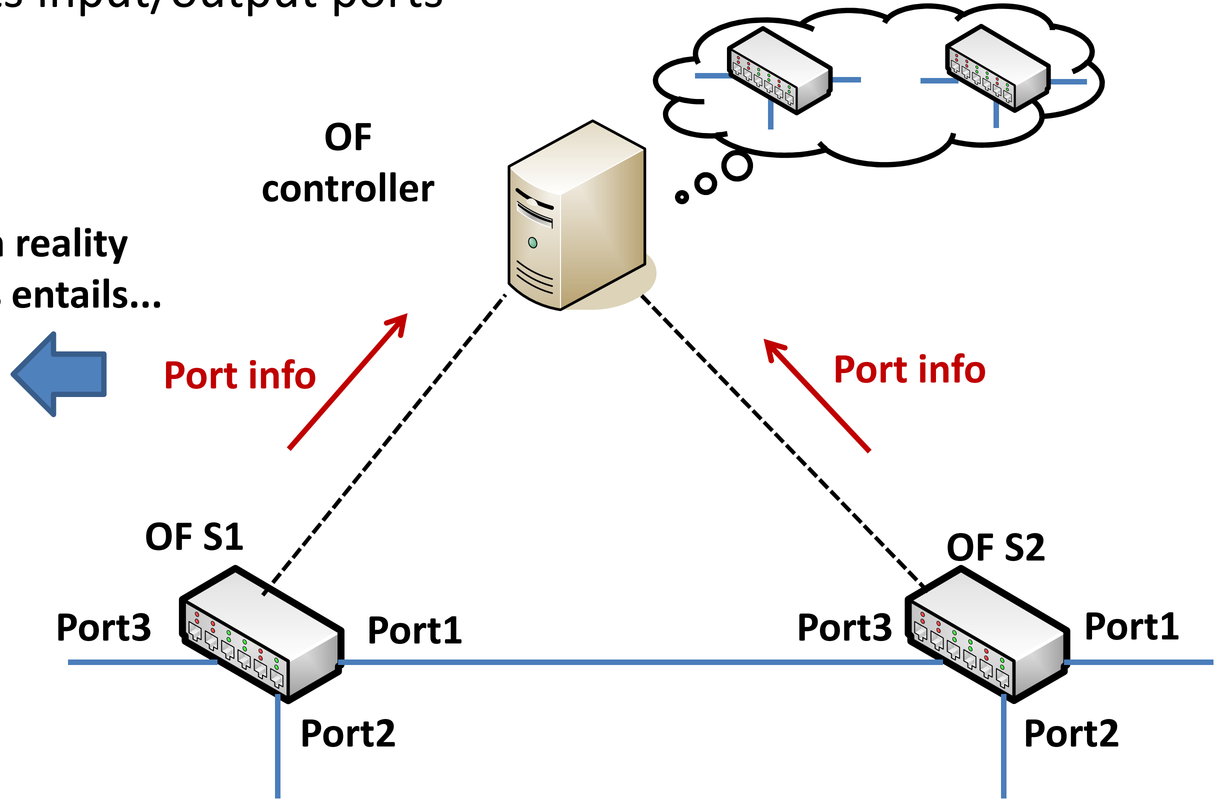


OpenFlow discovery protocol (2/4)

- When an OpenFlow switch is pre-configured (not yet operational), the switch IP address, controller IP address and default gateway are introduced
- Then, during network bootstrap, it connects to the controller providing information about its input/output ports

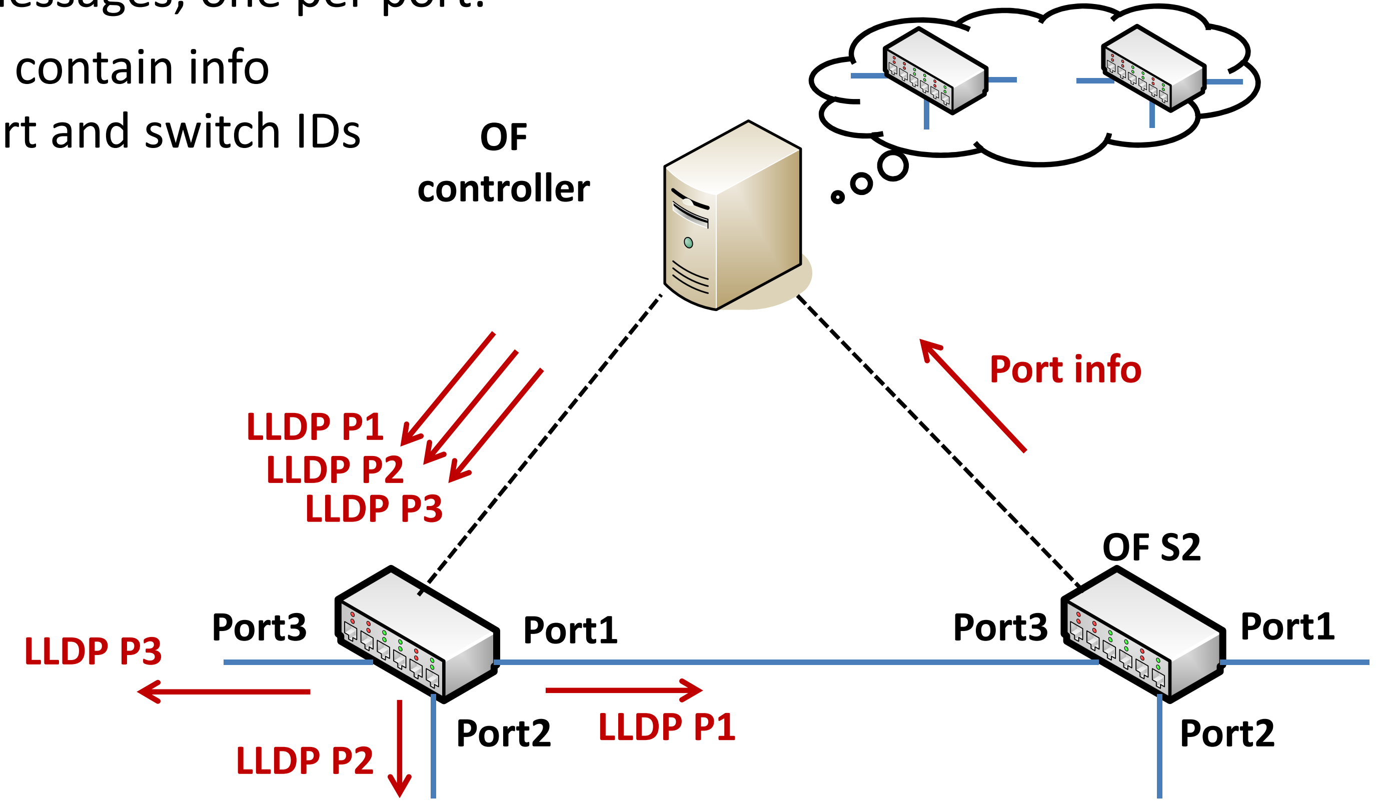


In reality this entails...



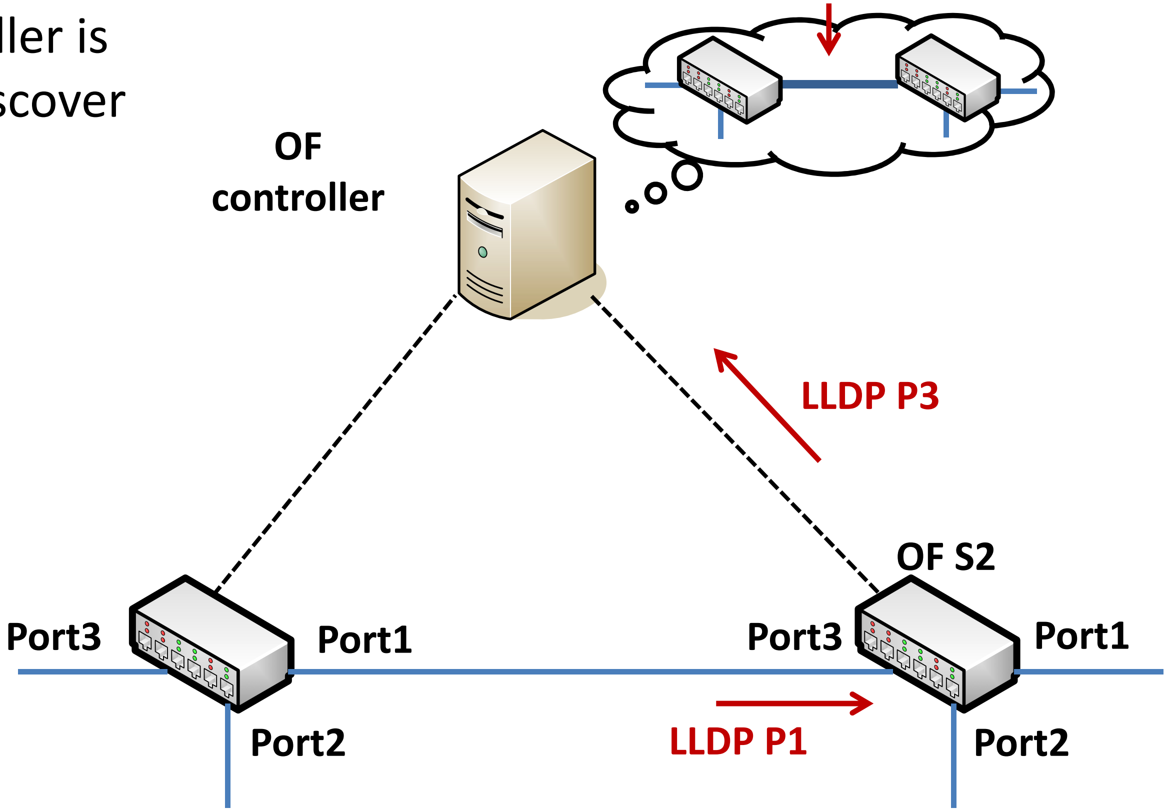
OpenFlow discovery protocol (3/4)

- Moreover, new rule in the forwarding table: any incoming LLDP frame has to be forwarded to the controller it via Packet-in message
- Then, the controller sends LLDP packets to every OpenFlow switch via Packet-out messages, one per port:
- LLDP packets contain info about the port and switch IDs



OpenFlow discovery protocol (4/4)

- Upon receipt of an LLDP packet by a neighboring OpenFlow switch, it sends the packet to the controller, together with information about the input port where received
- In this way, the controller is able to dynamically discover the network topology

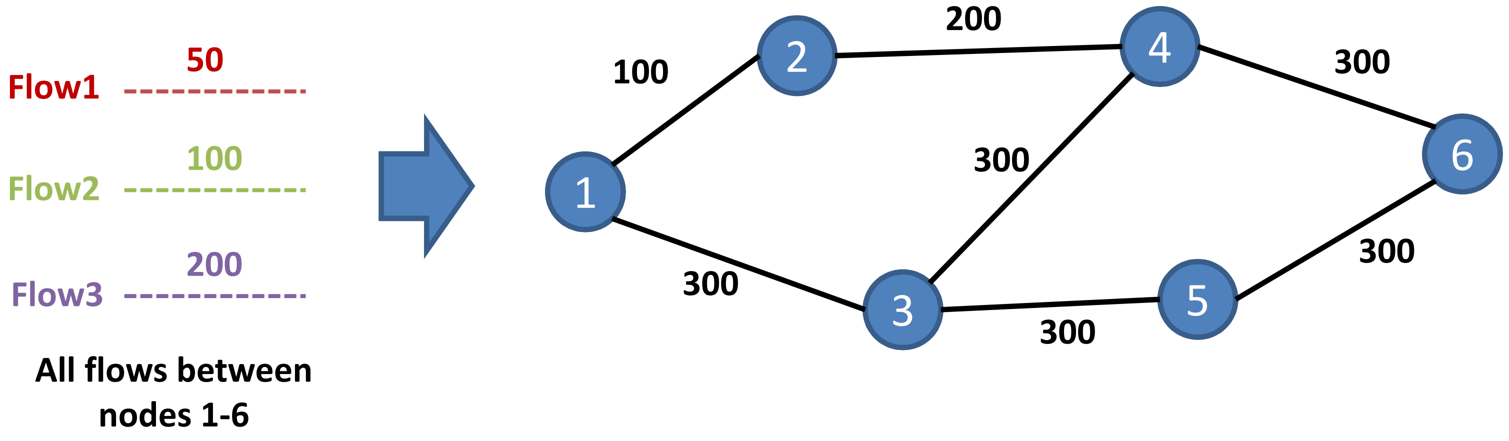


Traffic Engineering (TE) strategies in SDN

- Taking advantage of its global network view, a network operator can easily run customized TE strategies, either as core services in the SDN controller or as SDN applications connected to the northbound interface:
  - **Offline TE strategies** (a.k.a., network planning):
    - ➢ Executed to decide on the paths over which traffic flows will be transmitted before the network enters into operation
    - ➢ Based on (aggregated) traffic estimations
    - ➢ Typically entail joint optimization procedures
    - ➢ Execution time initially not an issue
  - **Online TE strategies**:
    - ➢ Executed to allocate a new flow request during the network operation
    - ➢ They take into account the current network state (capacity already used)
    - ➢ The goal is to optimally allocate that single flow request, although care may be taken to maximize the possibility to allocate future flows
    - ➢ Execution time becomes critical

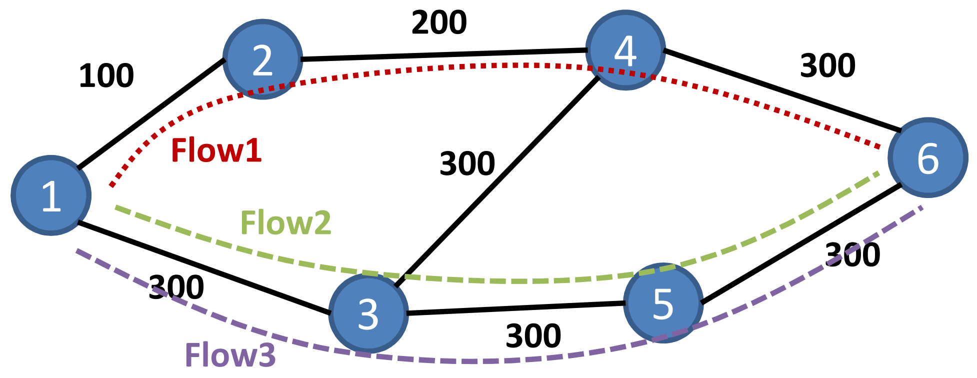
Multi-commodity flow problem (1/5)

- Offline TE strategies generally solve any variation of the generally called "Multi-Commodity Flow (MCF) problem", targeting the specific optimization goal of the network operator
- In a MCF problem, a set of commodities (=flows) have to be optimally allocated in the network, given a certain optimization goal (e.g., maximize number of allocated commodities) and a set of constraints (e.g., link capacities cannot be exceeded)
- Example:

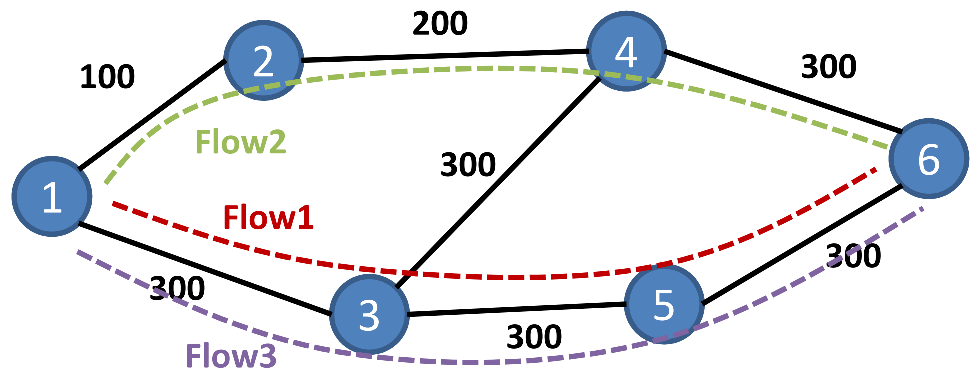


Multi-commodity flow problem (2/5)

- If the optimization goal is to maximize the number of allocated flows:



- If the optimization goal is to minimize the most loaded network link (e.g., with load-balancing purposes):



Multi-commodity flow problem (3/5)

- A typical approach to model the MCF problem is following a link-path Integer Linear Programming (ILP) formulation:

Input parameters:

$G(N,E)$: Network topology graph; N: Set of network nodes; E: Set of network links

C_e: Capacity of network link $e \in E$

D: Set of offered demands (i.e., flows) to be allocated in the network

Δ_d: Capacity required for demand $d \in D$

P_d: Set of precomputed candidate paths for $d \in D$ **Using any path computation algorithm (e.g., all distinct routes)**

Decision variables:

x_{dp}: Binary variable; equal to 1 if demand $d \in D$ over path $p \in P_d$; 0 otherwise

Objective function:

$$maximize \sum_{d \in D} \sum_{p \in P_d} x_{dp}$$

Maximize the number of allocated demands

subject to:

$$\sum_{p \in P_d} x_{dp} \leq 1 \,;\, \forall d \in D$$

$$\sum_{d \in D} \sum_{p \in P_d : e \in p} \Delta_d \cdot x_{dp} \leq C_e \;\; ; \;\; \forall e \in E$$

Multi-commodity flow problem (4/5)

- Linear programming and its variants (e.g., ILP) is a widely used optimization technique able to deliver the optimal solution of the modeled problem using commercial solvers (e.g., IBM CPLEX)
- If we would like to minimize the most loaded link in the network:

Additional decision variables:

y_e: Integer variable; capacity used in link $e \in E$

Y: Integer variables; capacity used in most used network link

Objective function:

$$minimize\ (|D| - \sum_{d \in D} \sum_{p \in P_d} x_{dp}) + \varepsilon \cdot Y$$

Secondary optimization goal: capacity used in most loaded link

subject to:

$$\sum_{p \in P_d} x_{dp} \leq 1\ ;\ \forall d \in D$$

$$\sum_{d \in D} \sum_{p \in P_d : e \in p} \Delta_d \cdot x_{dp} = y_e\ \ ;\ \ \forall e \in E$$

$$y_e \leq C_e\ \ ;\ \ \forall e \in E$$

$$y_e \leq Y\ \ ;\ \ \forall e \in E$$

Multi-commodity flow problem (5/5)

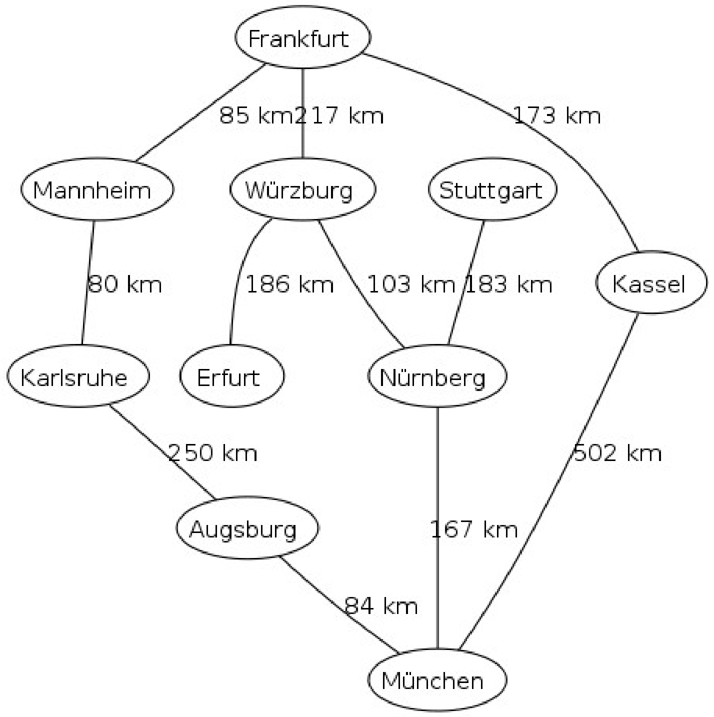
- ILP techniques succeed in providing the optimal solution to the targeted problem (e.g., offline TE problem). However, their computational complexity is very high, which makes it impossible to solve large problem instances with realistic execution times
- In such cases, it may be more appropriate to use heuristic algorithms able to provide good enough sub-optimal solutions with lower execution times:
    - Greedy heuristics
    - Genetic algorithms
    - Tabu search
    - Simulated annealing
    - Ant colony optimization
    - Iterated local search
    - etc.

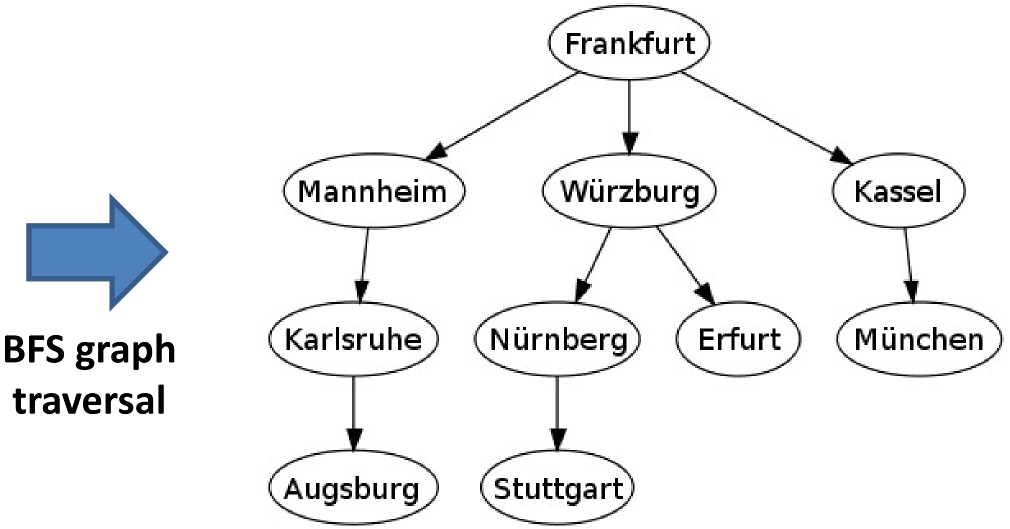
Constrained shortest path computation (1/4)

- Compared to offline scenarios, flow requests in an operational network arrive one after another, and must be served taking into account the current network state (e.g., unused available capacity at network links)
- In traditional IP-based networks, traffic demands follow the shortest paths in terms of hops from source to destination (as suggested by the routing protocol, like RIP or OSPF)
  - ➢ This practice tends to congest certain network regions, leaving others underutilized (typically, central network part vs. border regions)
- To avoid congestion and, thus, eventually delivering desirable Quality of Service (QoS) to the supported flows, a Constrained Shortest Path (CSP) computation becomes crucial
  - ➢ Most online TE strategies rely on this technique

Constrained shortest path computation (2/4)

- In network scenarios where the number of hops from source to destination is assumed as the path distance metric, the shortest path from source to destination can be obtained with a simple Breath First Search (BFS) :
  - ➢ BFS starts at the tree root (=flow source node) and explores the neighbor nodes first, before moving to the next level neighbors

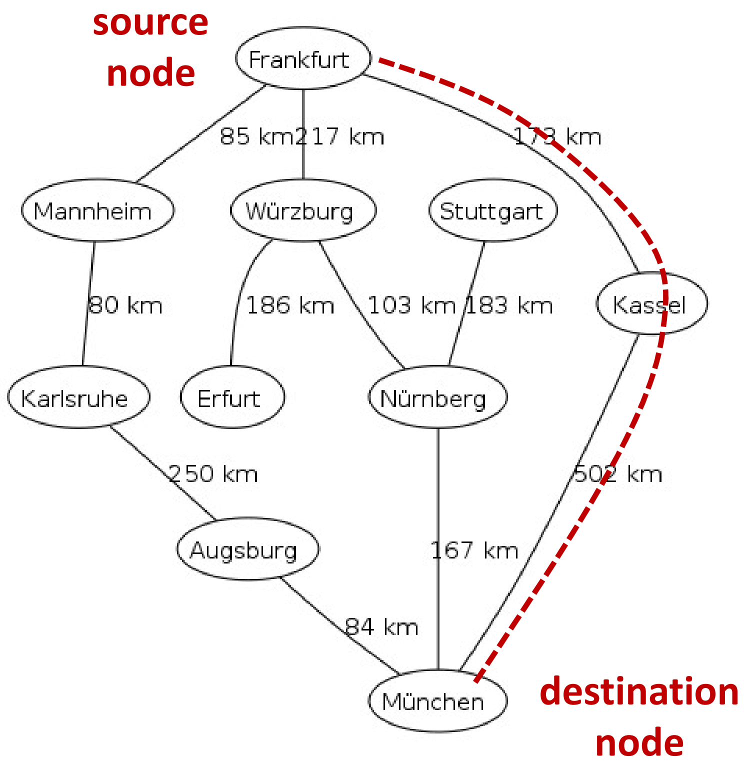


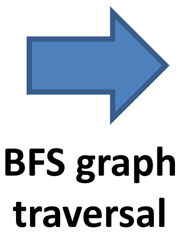


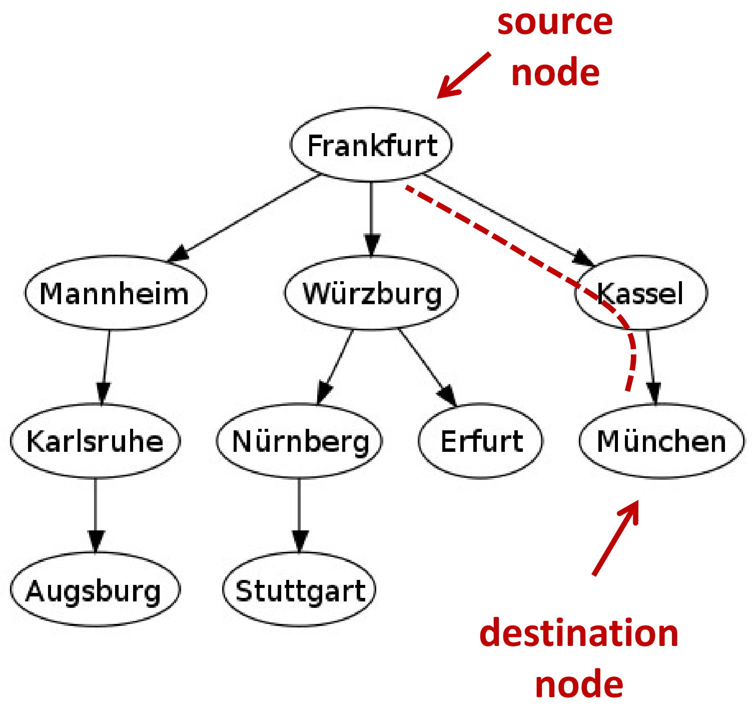
***Figures taken from https://en.wikipedia.org**

Constrained shortest path computation (3/4)

- Imagine in the previous example that we have to allocate a new flow between Frankfurt and München
  - ➢ By following the tree from the destination node up to the root node we can construct the shortest path across the network

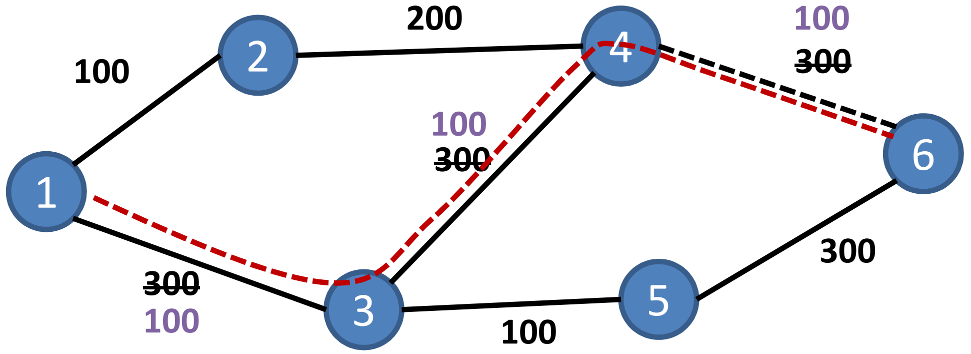




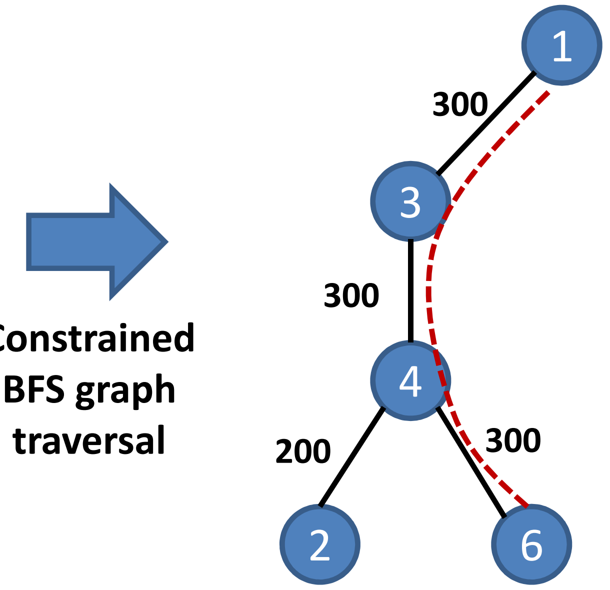


Constrained shortest path computation (4/4)

- Furthermore, we can constrain the BFS graph traversal, avoiding those links (or neighboring nodes) that do not accomplish certain characteristics, thus obtaining a (**very basic**) CSP computation
  - For example: avoid links without enough available capacity

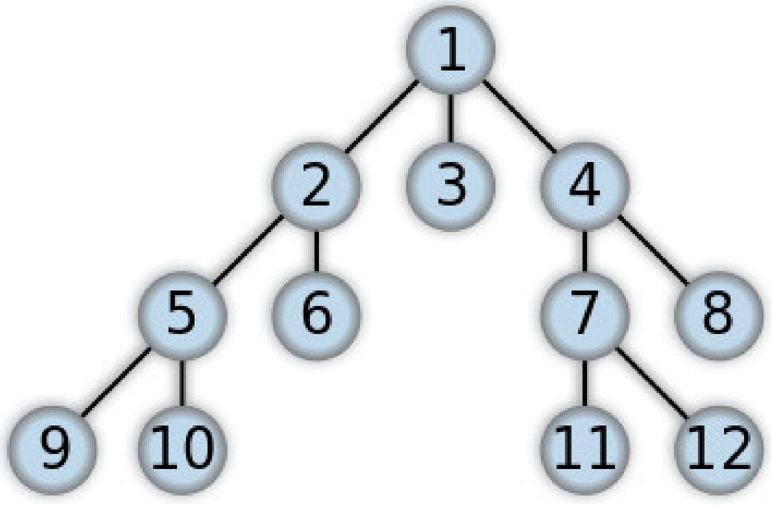


New flow beween 1-6 requiring a capacity = 200 must be established

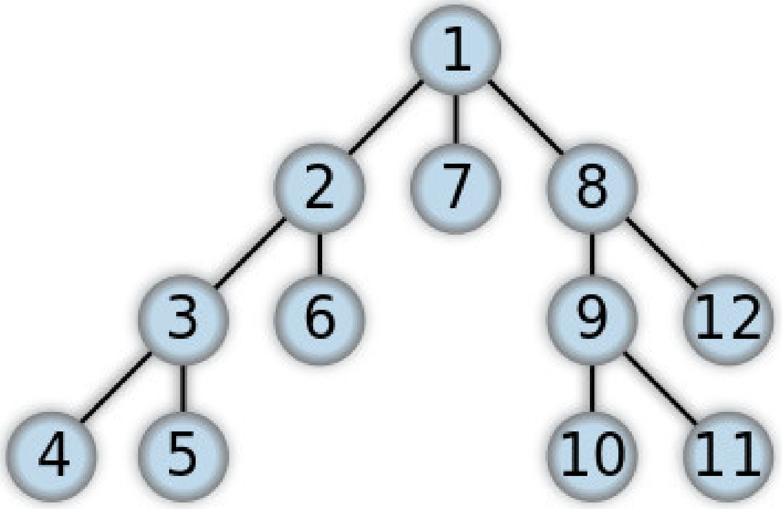


Bonus track! All distinct paths algorithm (1/6)

- In contrast to the previous CSP computation algorithm, this one bases on a Depth First Search (DFS) graph traversal
  - Instead of traversing the network graph level by level like BFS, DFS does it depth-wise, reaching the farthest node from the root and then backtracking



BFS

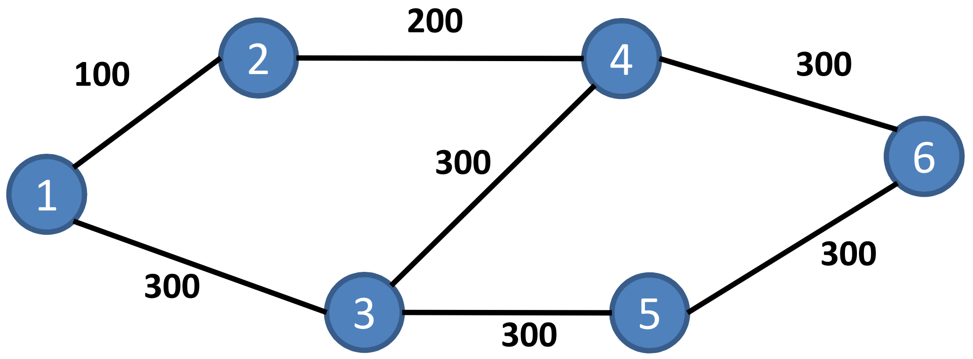


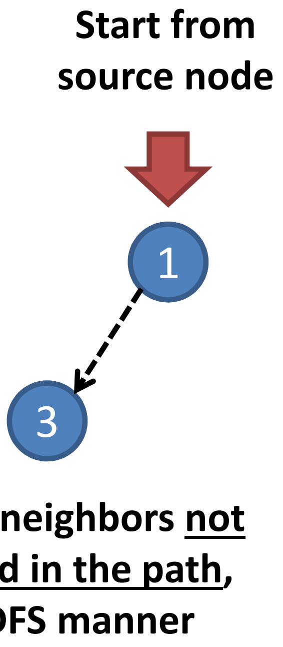
DFS

***Figures taken from https://en.wikipedia.org**

Bonus track! All distinct paths algorithm (2/6)

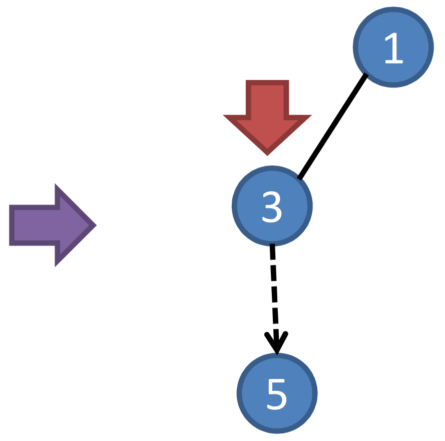
- Compute all distinct paths from nodes 1 to 6:





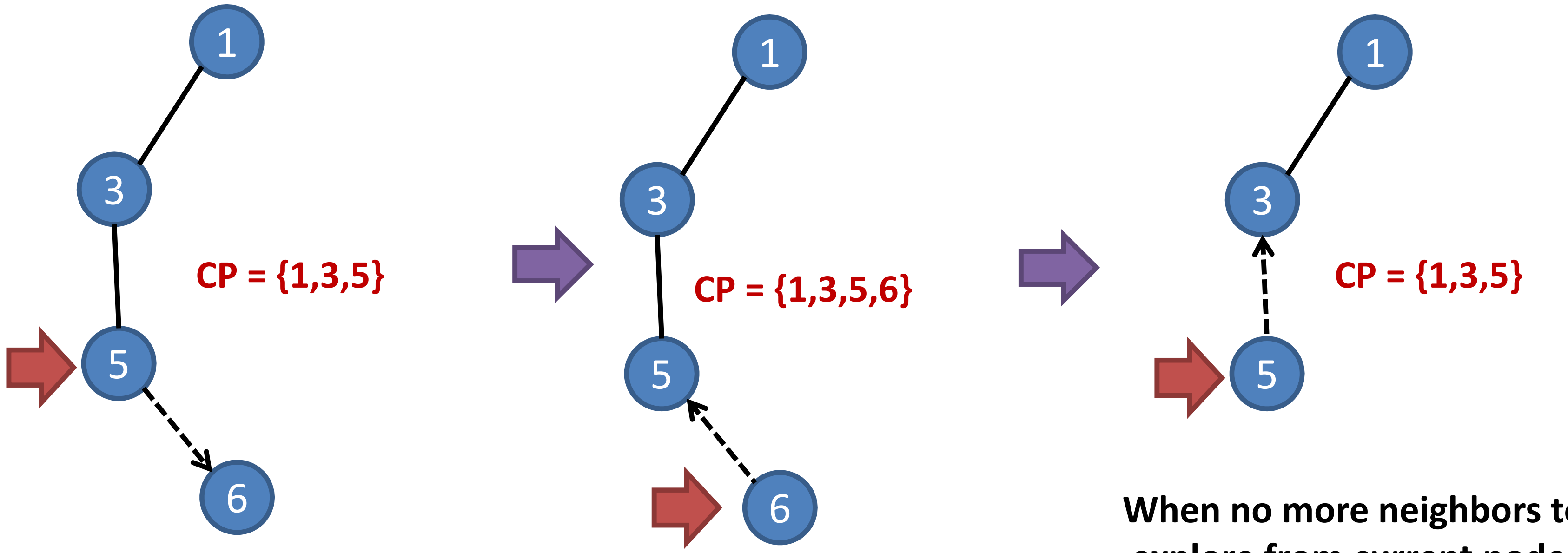
Add current node in the path under construction

CP = {1}



CP = {1,3}

Bonus track! All distinct paths algorithm (3/6)

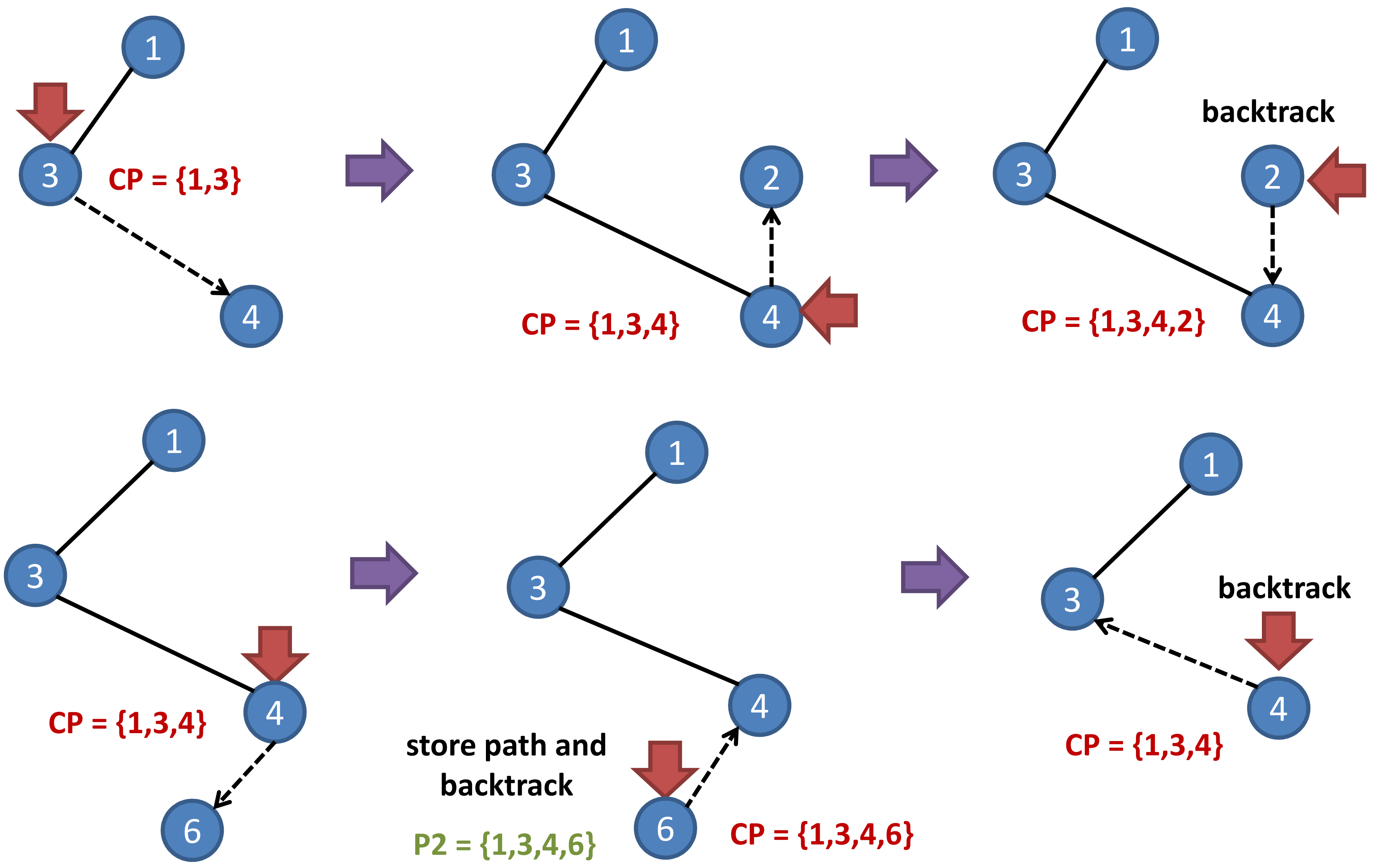


When no more neighbors to explore from current node, backtrack

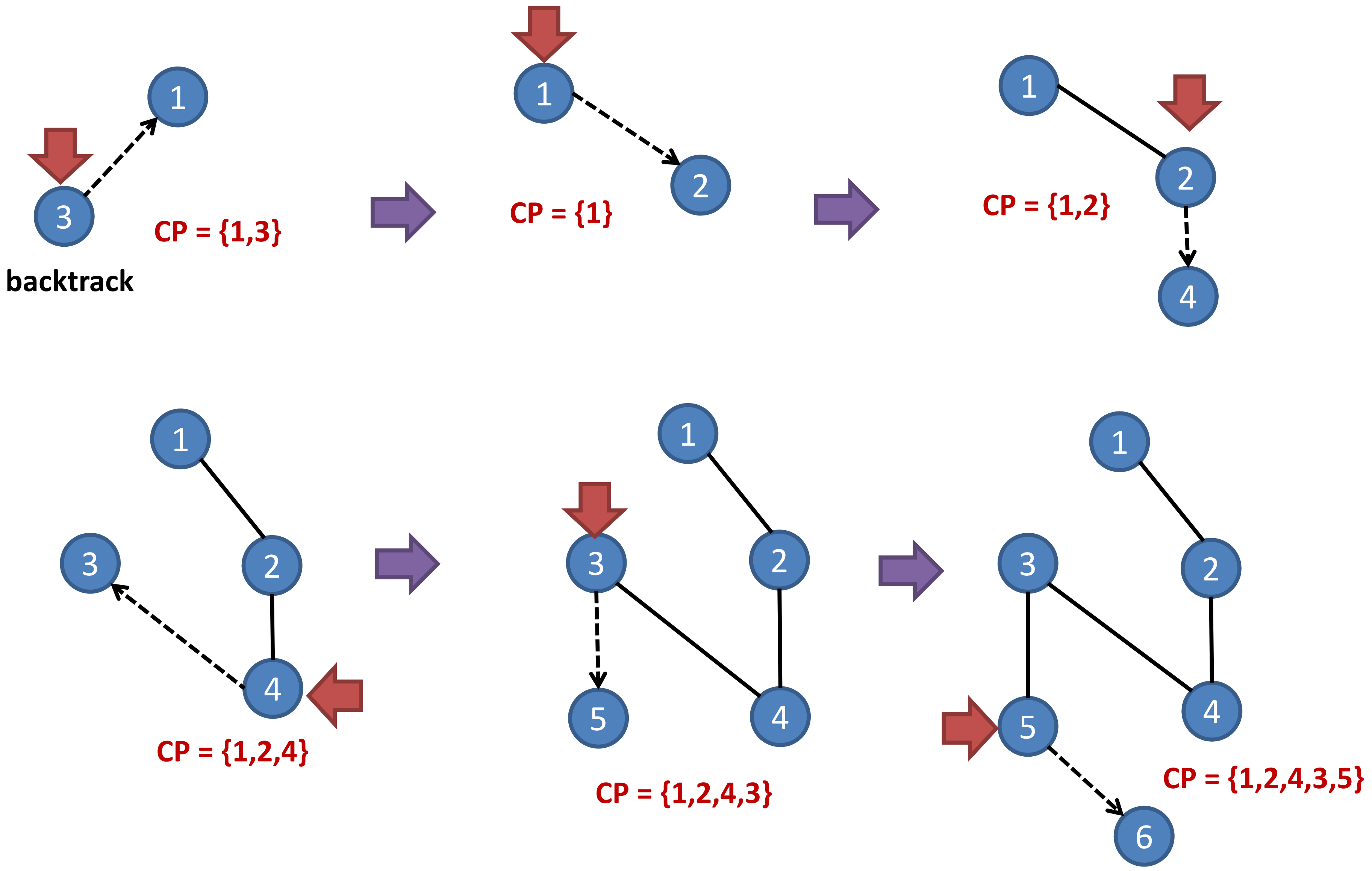
If current node = destination, store path and backtrack (remove last node in the path)

P1 = {1,3,5,6}

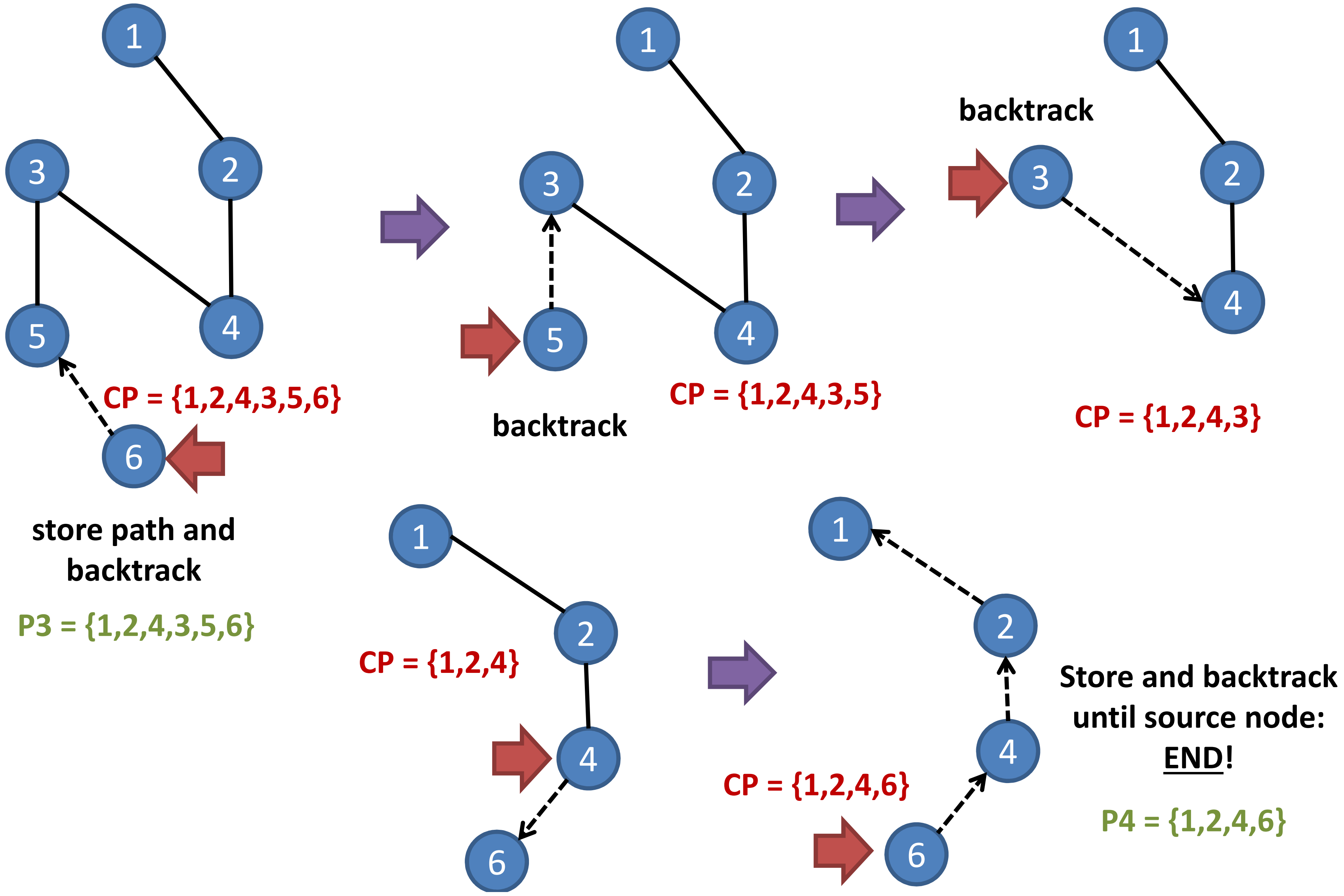
Bonus track! All distinct paths algorithm (4/6)



Bonus track! All distinct paths algorithm (5/6)



Bonus track! All distinct paths algorithm (6/6)



Network Functions Virtualization (NFV)

- Concept of replacing specialized network appliances (like firewalls, load balancers, NAT, routers, etc.) with software running on VMs in servers
- SDN and NFV can complement each other: Virtual Network Functions (VNFs) running on VMs in different servers can be connected over an SDN network to build a end-to-end service chain (i.e., a more complex network function by the sequencing of several VNFs)
- NFV benefits to network operators:
  - ➢ Hardware flexibility and reusability
  - ➢ Faster service provisioning (e.g., in an SDN network environment)
  - ➢ Network function scalability (no entire hardware replacement needed)
  - ➢ Easier innovation for a reduced time to market
  - ➢ Capital expenses reduction
- Under standardization by the European Telecommunications Standards Institute (ETSI) from 2012

The NFV framework

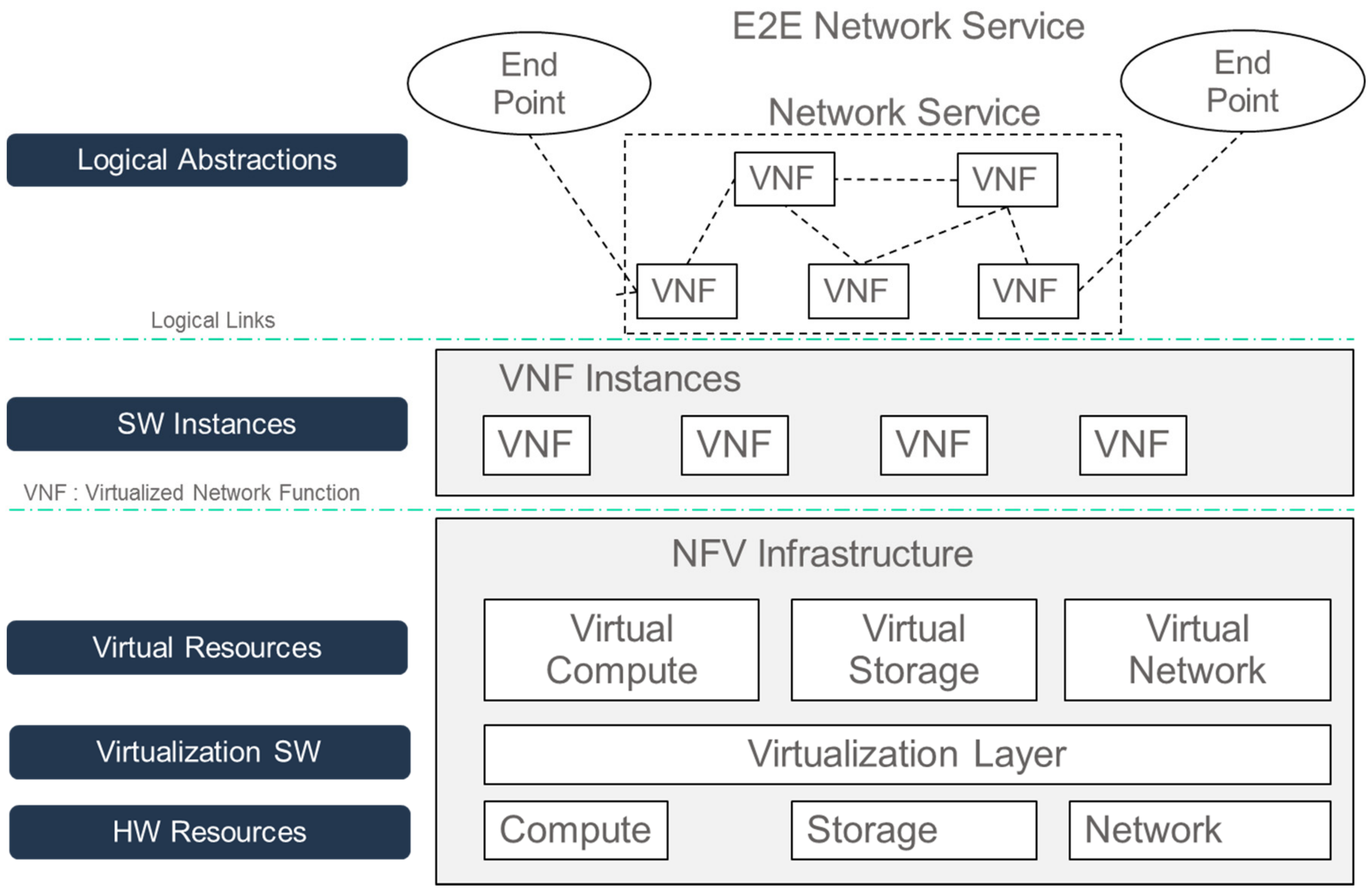
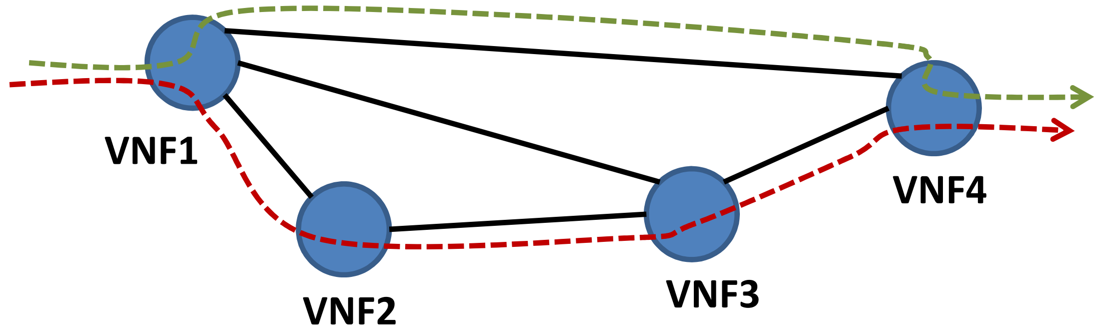


***Figure extracted from D.R. López, "When RINA meets NFV", presentation @ TNC 2015 Conference**

NFV service chain

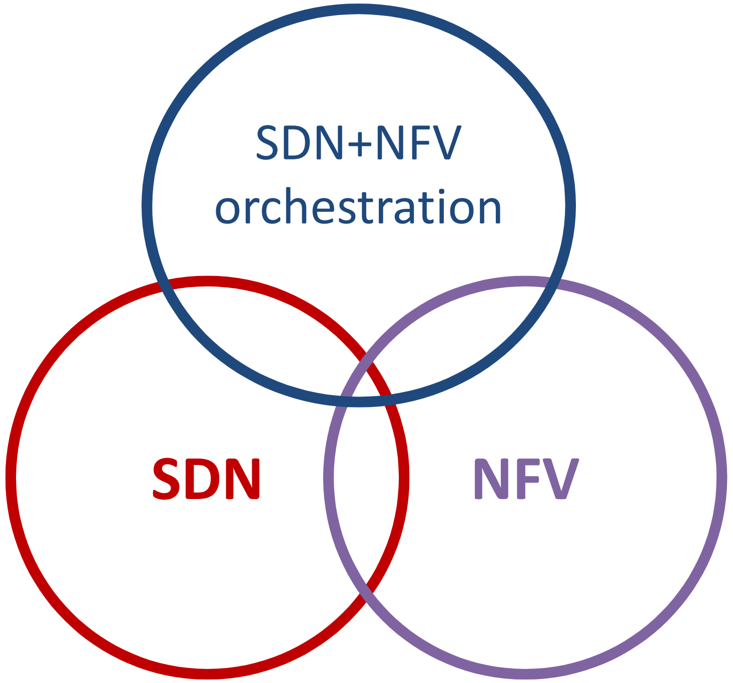
- Imagine that we have a “service graph”, that is, a network where each node implements a VM running one VNF (or several ones) and links offer connectivity (i.e., capacity) between them
- Depending on the end-user requirements, we can route its traffic flow across nodes of the service graph, applying the VNFs running on them
  - ➢ This describes an NFV service chain
- Of course, an adequate VNF placement will be crucial for proper NFV performance (low latencies, sufficient computational capacity, etc.)



VNF1 = Firewall
VNF2 = Video optimizer
VNF3 = Rate limiting
VNF4 = NAT

NFV and SDN together

- An SDN/NFV scenario encompasses both network and IT worlds
  - ➢ SDN fosters network resource virtualization & separation of control and forwarding plane functions at network devices
  - ➢ NFV fosters server virtualization to separate network functions from specialized hardware
- Therefore, upper-level orchestration (e.g., based on OpenStack) is needed to control the entire scenario and make it work efficiently
  - ➢ Dynamic setup of NFV service graphs and chains



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

The end

Future Internet Networks (FINE)

Master in Innovation and Research in Informatics (MIRI)