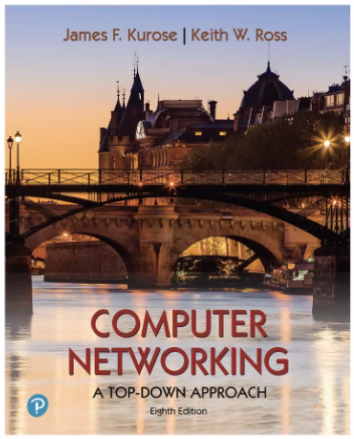
Lab 1

Lab 1

Software Defined Networks

Novella Bartolini
Viviana Arrigoni

Computer Network Performance 2021-2022



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Adapted slides from chapters 4 and 5

Vulnerability Assessment and Robust Defenses for Optimized Attacks in Dynamic SDNs

4th call: Strengthening trustworthiness and resilience of internet

NGI ATLANTIC.EU

EU partners: Novella Bartolini, Viviana Arrigoni

US partner: Ting He

Goals

1. Monitoring system for SDNs via Network Tomography

   V. Arrigoni, N. Bartolini, A. Massini and F. Trombetti "Failure Localization through Progressive Network Tomography." IEEE INFOCOM 2021

2. Cache pollution attack

   M. Yu, T. Xie, T. He, P. McDaniel and Q. K. Burke. "Flow table security in SDN: Adversarial reconnaissance and intelligent attacks," IEEE/ACM Transactions on Networking 29.6 (2021): 2793-2806

3. Dynamic Routing and Provisioning (proactive defense)

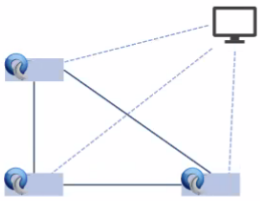
   S. Ciavarella, N. Bartolini, H. Khamfroush and T. La Porta " Progressive damage assessment and network recovery after massive failures." IEEE INFOCOM 2017

4. Adaptive rule replacement policy (reactive defense)

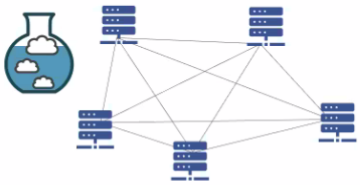
   T. Xie, T. He, P. McDaniel and N. Nambiar "Attack resilience of cache replacement policies." IEEE INFOCOM 2021

Experiments

1. Prototype



2. Distributed emulations (claudlab)



3. Local emulations (mininet)

To be continued…

Two key network-layer functions

network-layer functions:

- *forwarding:* move packets from a router's input link to appropriate router output link
- *routing:* determine route taken by packets from source to destination
  - *routing algorithms*

Network layer: data plane, control plane

Data plane:

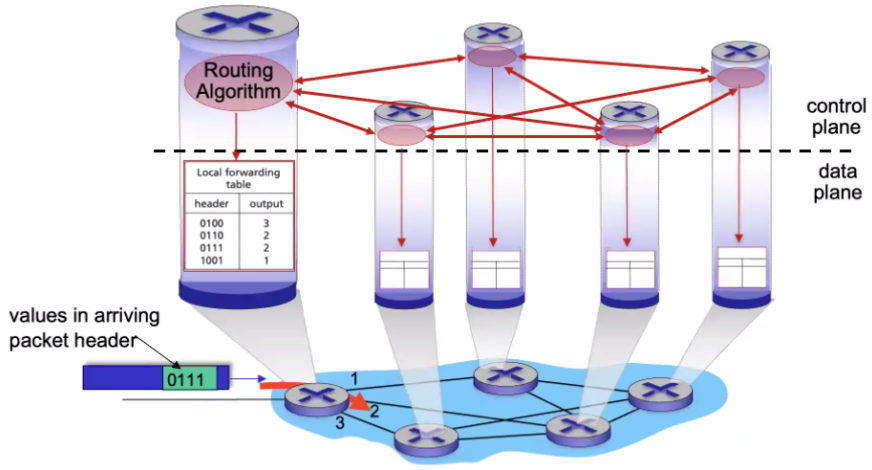
- *local*, per-router function
- determines how datagram arriving on router input port is forwarded to router output port

Control plane

- *network-wide* logic
- determines how datagram is routed among routers along end-end path from source host to destination host
- two control-plane approaches:
  - *traditional routing algorithms:* implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers

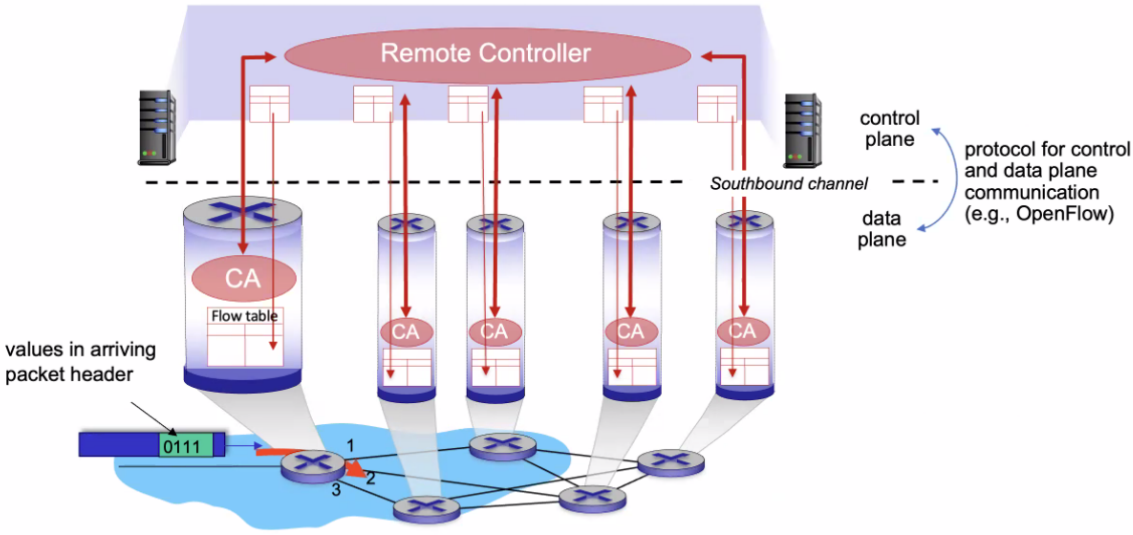
Per-router control plane

Individual routing algorithm components *in each and every router* interact in the control plane



Software-Defined Networking (SDN) control plane

Remote controller computes, installs forwarding tables in routers



data plane

Generalized forwarding: match plus action

Each router contains a flow/forwarding table

- "match plus action" abstraction: match bits in arriving packet, take action
  - *destination-based forwarding:* forward based on dest. IP address
  - *generalized forwarding*:
    - many header fields can determine action
    - many action possible: drop/copy/modify/log packet
    - allows for a very flexible definition of flows and for arbitrarily granural routing decisions

IP flow

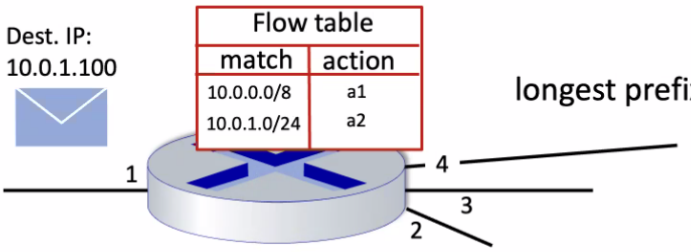
- Quintuple information

Source IP	Destination IP	Source Port Nr	Destination Port Nr	Protocol type

- It is the same for all routers

Flow table abstraction

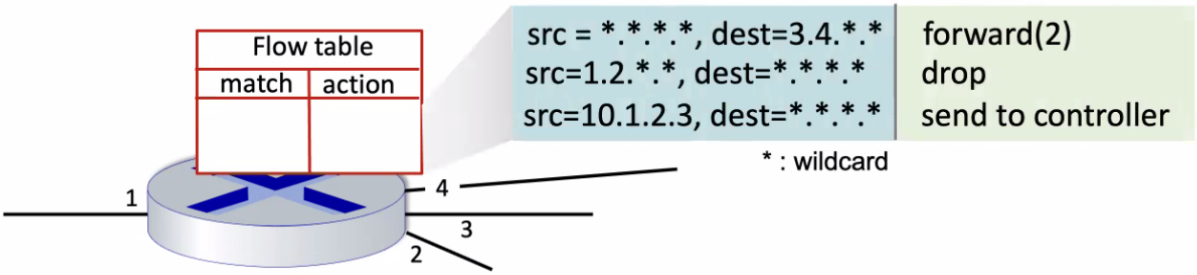
- flow: defined by header fields
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, or **send matched packet to controller**
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets



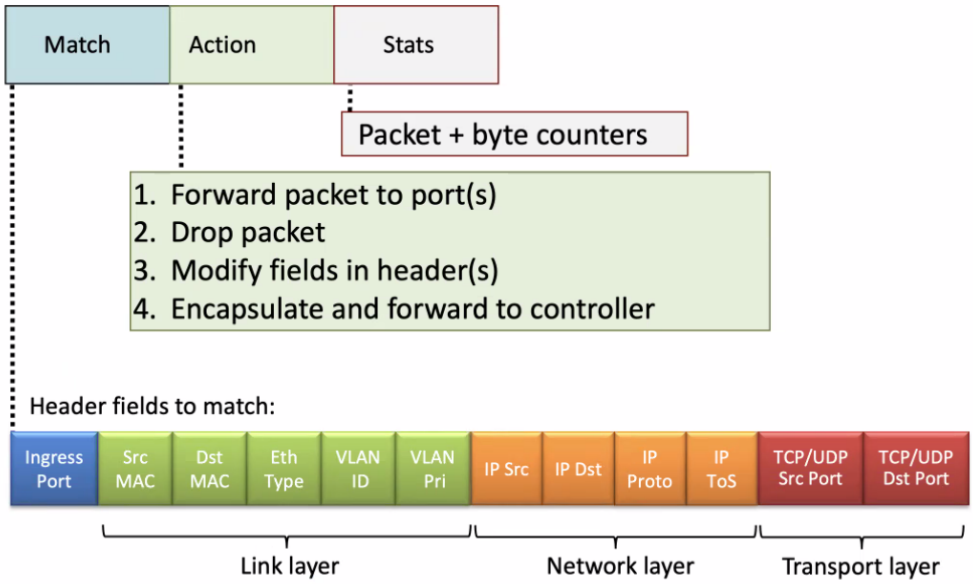
longest prefix matching for disambiguation

Flow table abstraction

- flow: defined by header fields
- generalized forwarding: simple packet-handling rules
  - match: pattern values in packet header fields
  - actions: for matched packet: drop, forward, modify, or **send matched packet to controller**
  - priority: disambiguate overlapping patterns
  - counters: #bytes and #packets



OpenFlow: flow table entries



OpenFlow: examples

Destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	51.6.0.8	*	*	*	*	port6

IP datagrams destined to IP address 51.6.0.8 should be forwarded to router output port 6

Firewall:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	*	*	*	*	*	22	drop

Block (do not forward) all datagrams destined to TCP port 22 (ssh port #)

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	*	*	*	*	128.119.1.1	*	*	*	*	*	drop

Block (do not forward) all datagrams sent by host 128.119.1.1

OpenFlow: examples

Layer 2 destination-based forwarding:

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	VLAN Pri	IP Src	IP Dst	IP Prot	IP ToS	TCP s-port	TCP d-port	Action
*	*	22:A7:23:11:E1:02	*	*	*	*	*	*	*	*	*	port3

layer 2 frames with destination MAC address 22:A7:23:11:E1:02 should be forwarded to output port 3

OpenFlow abstraction

- match+action: abstraction unifies different kinds of devices

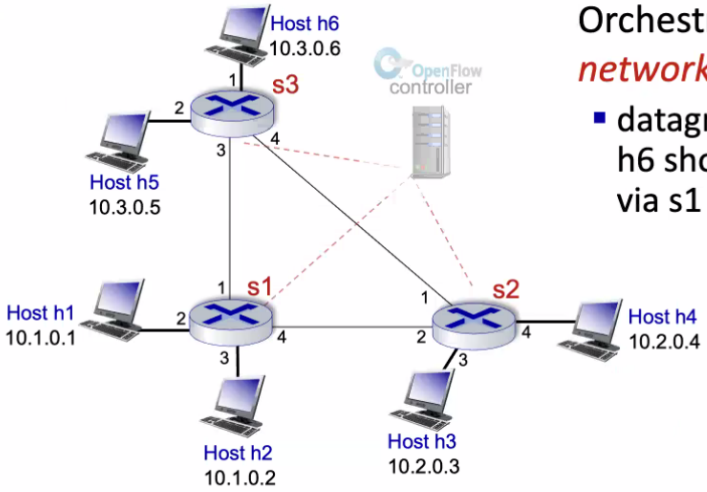
Router
- *match:* longest destination IP prefix
- *action:* forward out a link

Switch
- *match:* destination MAC address
- *action:* forward or flood

Firewall
- *match*: IP addresses and TCP/UDP port numbers
- *action:* permit or deny

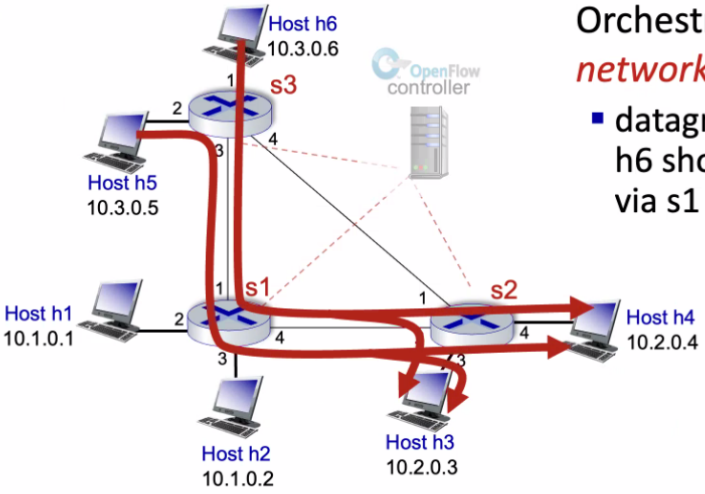
NAT
- *match:* IP address and port
- *action:* rewrite address and port

OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:
- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

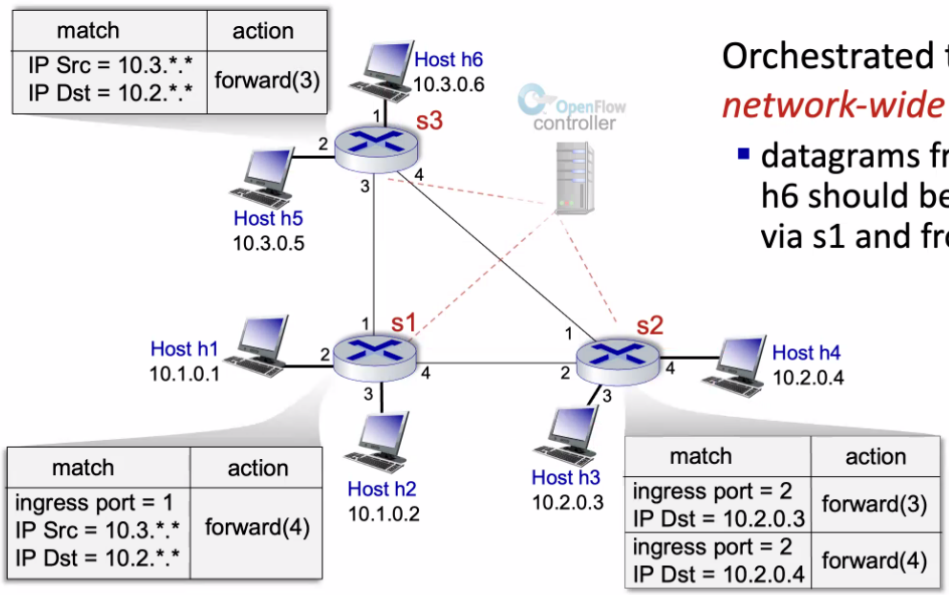
OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:

- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

OpenFlow example



Orchestrated tables can create *network-wide* behavior, e.g.,:

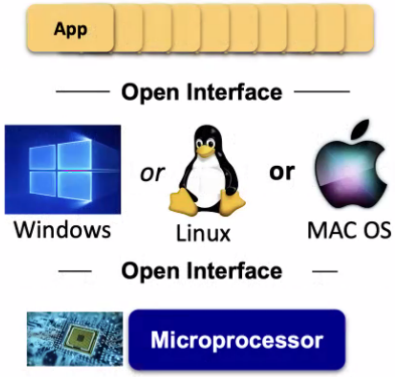
- datagrams from hosts h5 and h6 should be sent to h3 or h4, via s1 and from there to s2

control plane

Software defined networking (SDN)

- Internet network layer: historically implemented via distributed, per-router control approach:
  - *monolithic* router contains switching hardware, runs proprietary implementation of Internet standard protocols (IP, RIP, IS-IS, OSPF, BGP) in proprietary router OS (e.g., Cisco IOS)
  - different "middleboxes" for different network layer functions: firewalls, load balancers, NAT boxes, ..
- ~2005: renewed interest in rethinking network control plane

SDN analogy: mainframe to PC revolution



Vertically integrated
Closed, proprietary
Slow innovation
Small industry

Horizontal
Open interfaces
Rapid innovation
Huge industry

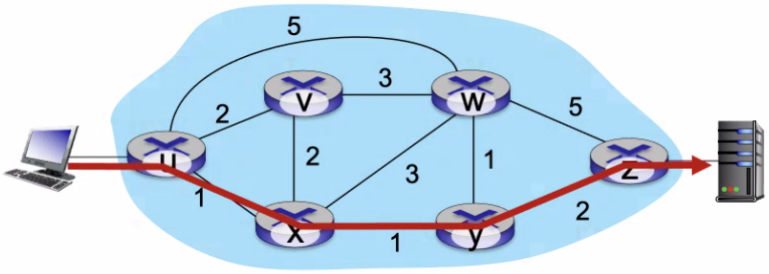
* Slide courtesy: N. McKeown

Software defined networking (SDN)

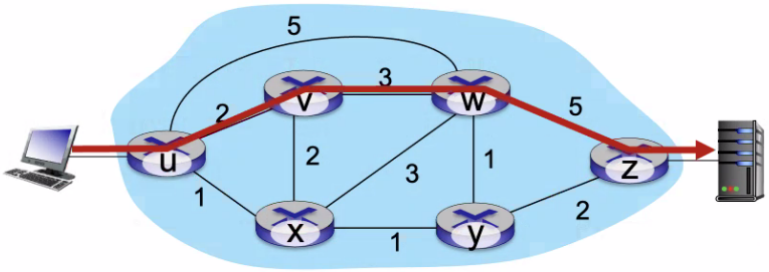
Why a *logically centralized* control plane?

- easier network management: avoid router misconfigurations, greater flexibility of traffic flows
- table-based forwarding (recall OpenFlow API) allows "programming" routers
  - centralized "programming" easier: compute tables centrally and distribute
  - distributed "programming" more difficult: compute tables as result of distributed algorithm (protocol) implemented in each-and-every router
- open (non-proprietary) implementation of control plane
  - foster innovation

Traffic engineering: difficult with traditional routing



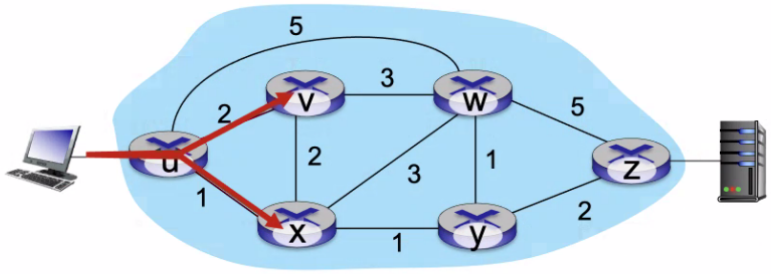
Traffic engineering: difficult with traditional routing



Q: what if network operator wants u-to-z traffic to flow along *uvwz*, rather than *uxyz*?

A: need to re-define link weights so traffic routing algorithm computes routes accordingly (or need a new routing algorithm)!

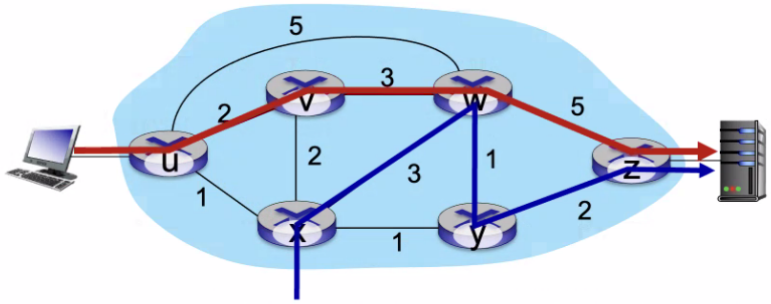
Traffic engineering: difficult with traditional routing



Q: what if network operator wants to split u-to-z traffic along uvwz *and* uxyz (load balancing)?

A: can't do it (or need a new routing algorithm)

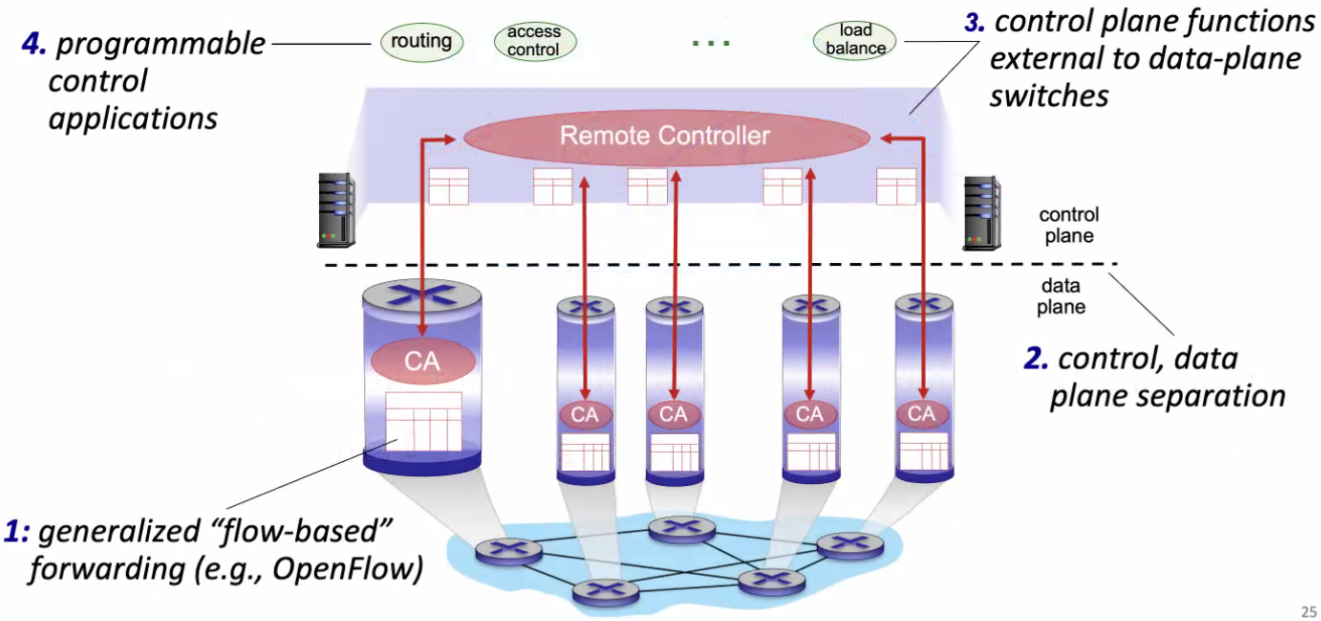
Traffic engineering: difficult with traditional routing



Q: what if w wants to route blue and red traffic differently from w to z?

A: can't do it (LS, DV routing)

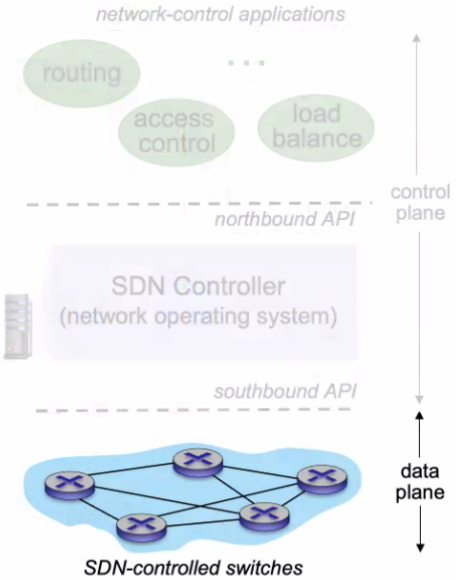
Software defined networking (SDN)



Software defined networking (SDN)

Data-plane switches:

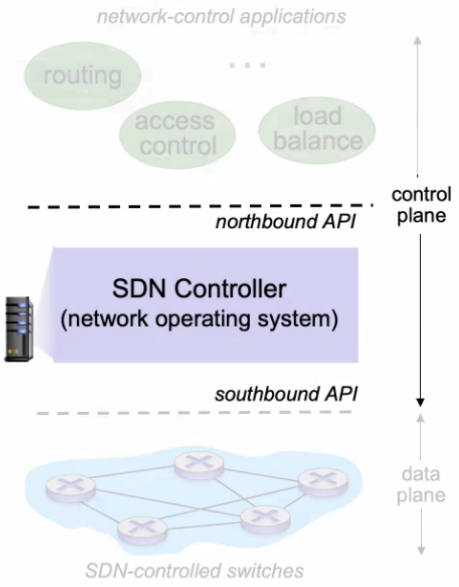
- fast, simple, commodity switches implementing generalized data-plane forwarding in hardware (TCAM)
- flow (forwarding) table computed, installed under controller supervision
- API for table-based switch control (e.g., OpenFlow)
  - defines what is controllable, what is not
- protocol for communicating with controller (e.g., OpenFlow)



Software defined networking (SDN)

SDN controller (network OS):

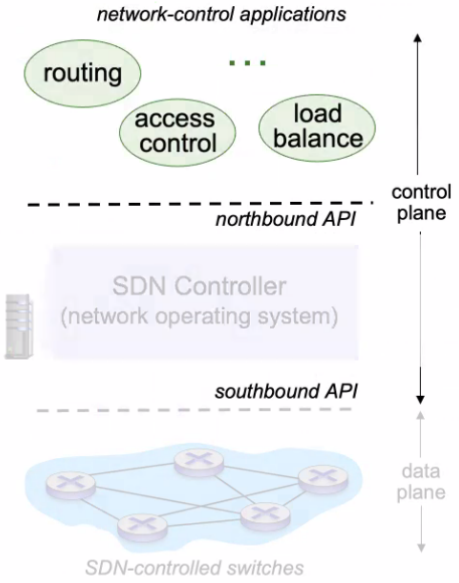
- maintain network state information
- interacts with network control applications "above" via northbound API
- interacts with network switches "below" via southbound API
- implemented as distributed system for performance, scalability, fault-tolerance, robustness



Software defined networking (SDN)

network-control apps:

- "brains" of control: implement control functions using lower-level services, API provided by SDN controller
- *unbundled:* can be provided by 3rd party: distinct from routing vendor, or SDN controller

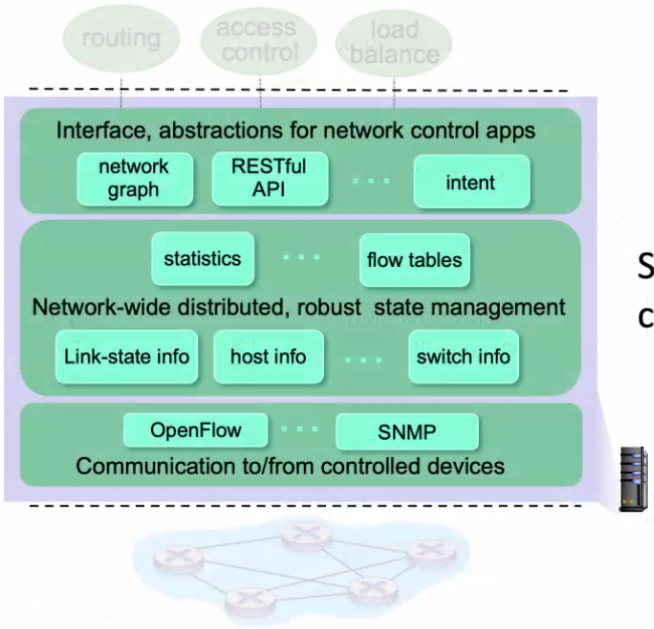


Components of SDN controller

interface layer to network control apps: abstractions API

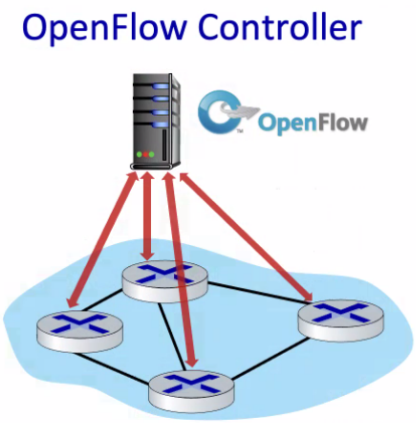
network-wide state management : state of networks links, switches, services: a *distributed database*

communication: communicate between SDN controller and controlled switches

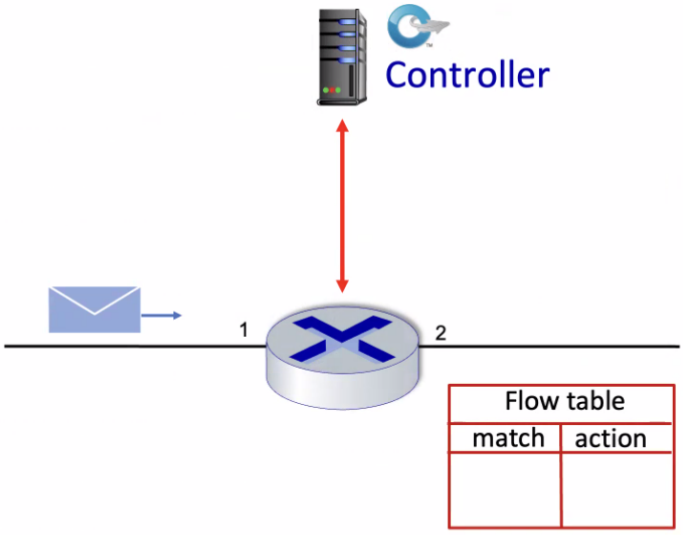


OpenFlow protocol

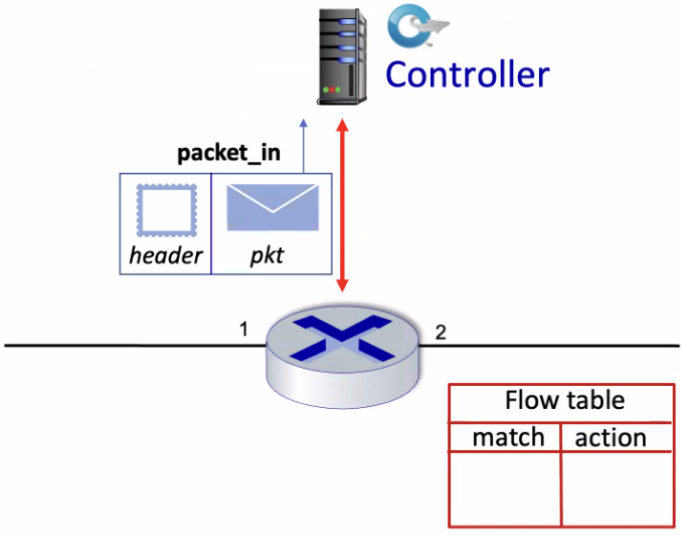
- operates between controller and switch
- TCP used to exchange messages
  - optional encryption (TLS)
- three classes of OpenFlow messages:
  - controller-to-switch
  - asynchronous (switch to controller)
  - symmetric (misc.)



Switch & controller interaction example

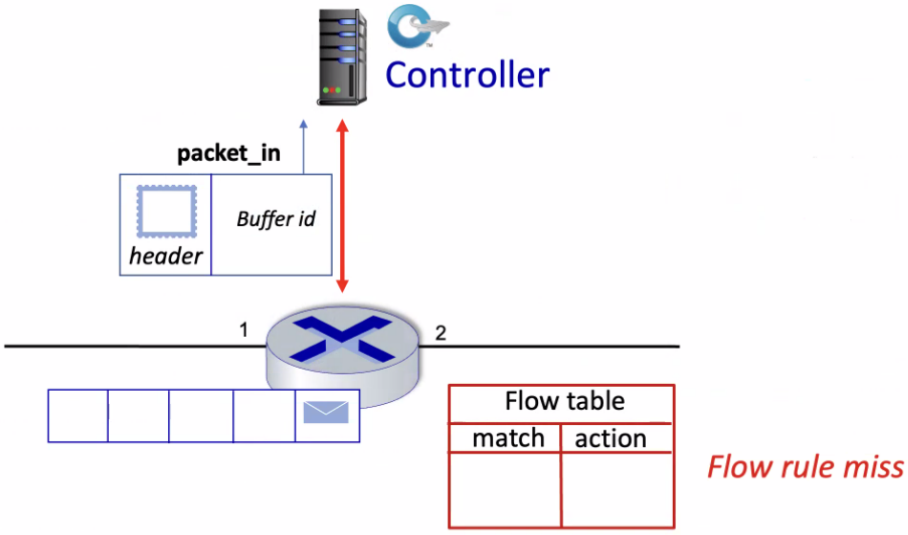


Switch & controller interaction example



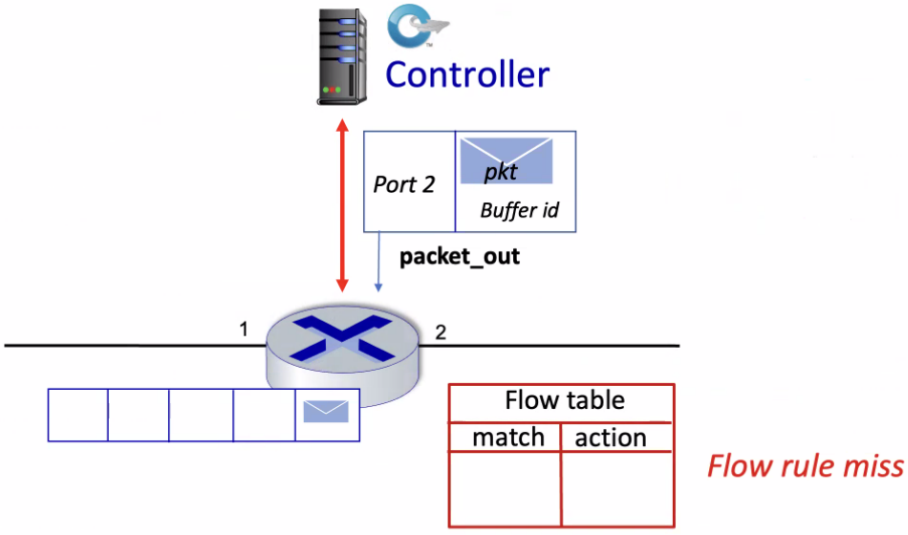
Flow rule miss

Switch & controller interaction example



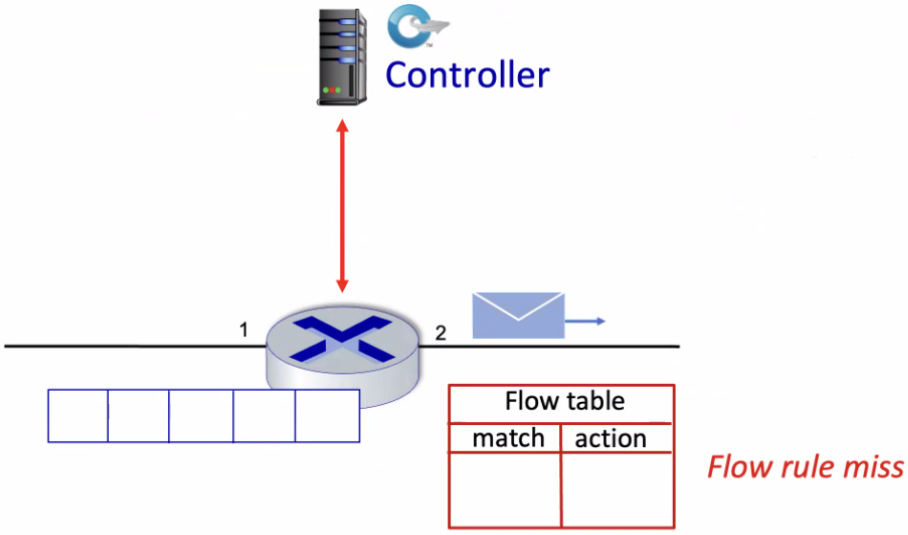
Flow rule miss

Switch & controller interaction example



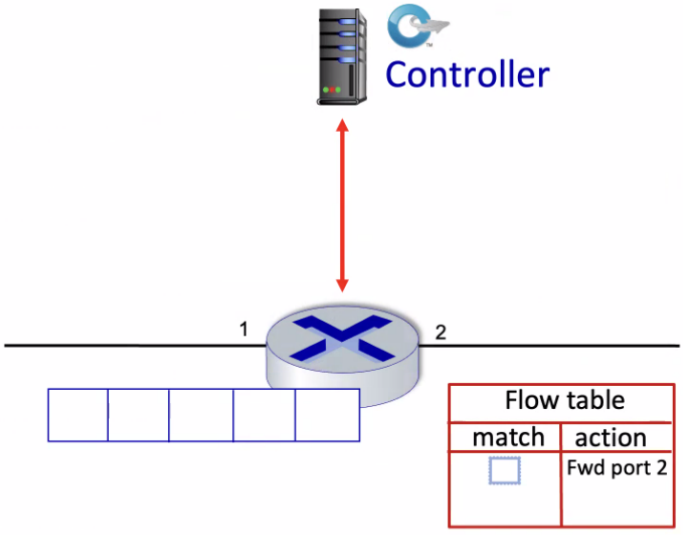
Flow rule miss

Switch & controller interaction example



Flow rule miss

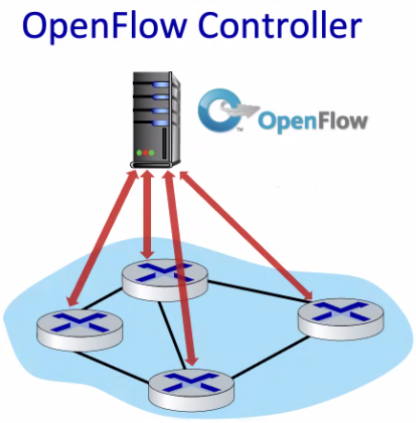
Switch & controller interaction example



OpenFlow: controller-to-switch messages

Key controller-to-switch messages

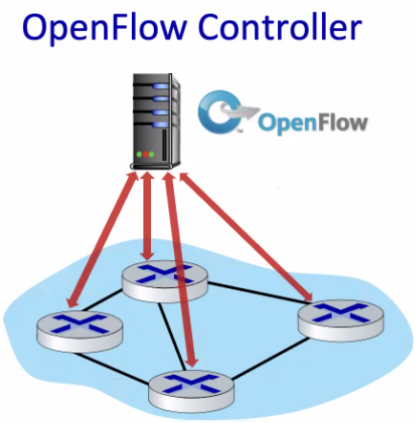
- *features:* controller queries switch features, switch replies
- *configure:* controller queries/sets switch configuration parameters
- *modify-state:* add, delete, modify flow entries in the OpenFlow tables
- *packet-out:* controller can send this packet out of specific switch port



OpenFlow: switch-to-controller messages

Key switch-to-controller messages

- *packet-in:* transfer packet (and its control) to controller. See packet-out message from controller
- *flow-removed:* flow table entry deleted at switch
- *port status:* inform controller of a change on a port.



SDN usage

- Data centres
- 5G wireless networks
- Combination with Network Function Virtualization (NFV) and Network Virtualization (NV)
  - *NFV:* network functions are executed on virtual machines at top of servers. Unload physical network nodes.
  - *NV:* allows different proprietors to share physical devices and to build different topologies on top of those. Hardware and software network resources and network functionalities combined into a single, software-based administrative **virtual network**.

SDN vulnerabilities

Control plane

- Single point of failure (controller)
- Open programmability may induce erroneous code
- Weak authentication among controller and control plane applications/data plane

Data plane

- Limited switch buffer memory
- Physical links shared by control and data traffic can expose sensitive information to attackers
- Cache pollution attacks

Cache pollution attack (DoS)

- *Idea:* craft an intelligent attack by sending flows that cause highly frequent cache misses to legitime users.

1. *Infer cache size C and replacement policy (FIFO/LRU)*
   - measuring delays
2. *Infer background traffic rate*
   - $\{f_1, \dots, f_F\}$ background flows, f_i independent Poisson process of rate λ_i
3. *Infer probing rates λ_j' of malicious flow*
   - find λ_a such that if $\lambda_j' \equiv \lambda_a$ hit probability is very low.

M. Yu, T. Xie, T. He, P. McDaniel and Q. K. Burke. "Flow table security in SDN: Adversarial reconnaissance and intelligent attacks," IEEE/ACM Transactions on Networking 29.6 (2021): 2793-2806

Network Tomography

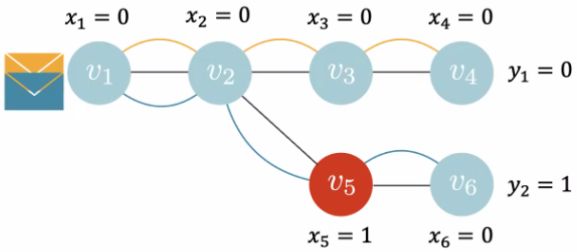
- *Idea:* monitor a network via end-to-end path probes.
  - Boolean Network tomograhy: nodes, links and paths have a binary state: working or failed.
  - Additive Network tomography: nodes and links have continuous values and are summed up in the path.

- Routing matrix R: given set of paths $P = \{p_1, \dots, p_m\}$, $R_{i,j} = 1$ if node v_j / link l_j is traversed by path p_i. $R_{i,j} = 0$ otherwise.

- *Why Network Tomography for SDN monitoring?*
  Do not overload the controller with continuous requests

Boolean Network Tomography

- Nodes can be working or failed. If node v_i works, $x_i = 0$, if v_i fails, $x_i = 1$
- A path p_j is working if it only traverses working nodes, $y_j = 0$.
  A path fails if it traverses at least one failed node, $y_j = 1$
- System of Boolean equations

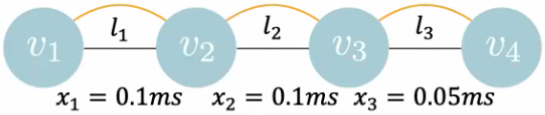


$$y_j = \bigvee_{v_i \in p_j} x_i$$

$O(2^n)$ solutions

Additive Network Tomography (delay)

- Each link l_i has a delay x_i. A path p_j has a delay equal to the sum of the delays of the links it traverses y_j.
- System of linear equations



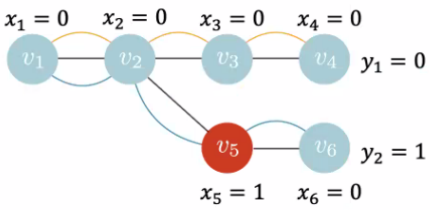
$$x_1 + x_2 + x_3 = y_1$$

Possibly ∞ solutions

- Can be transformed into a problem of BNT.
  Delayed links $x_i > t$, t threshold for anomalous delay.

Identifiability

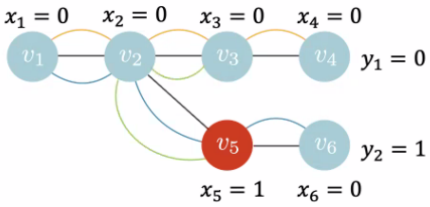
- *Boolean Network Tomography*
  A node v is 1-identifiabile if its *binary encoding* $b(v)$ is unique.



$$\begin{matrix} b(v_1) & b(v_2) & b(v_3) & b(v_4) & b(v_5) & b(v_6) \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \end{bmatrix}$$

Identifiability

- *Boolean Network Tomography*
  A node v is 1-identifiabile if its *binary encoding* $b(v)$ is unique.

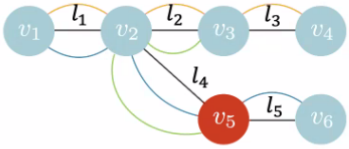


$$\begin{matrix} b(v_1) & b(v_2) & b(v_3) & b(v_4) & b(v_5) & b(v_6) \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 0 & 1 & 0 \end{bmatrix}$$

Identifiability

- *Delay Network Tomography*

  A routing matrix. A link l_i is identifiabile if $\forall x \in Ker(A), x_i = 0$.
  $Ker(A) \coloneqq \{x: Ax = 0\}$



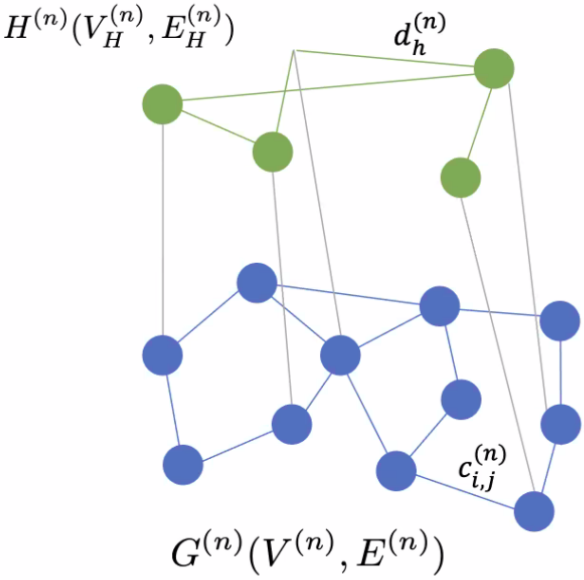
$$\begin{matrix} b(l_1) & b(l_2) & b(l_3) & b(l_4) & b(l_5) \end{matrix}$$
$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 0 \end{bmatrix}$$

$$\begin{cases} x_1 + x_2 + x_3 = 0 \\ x_1 + x_4 + x_5 = 0 \\ x_2 + x_4 = 0 \end{cases}$$

$$Ker(A) = span(\{-s - t, -s, 2s + t, s, t\})$$

41

Flow routing



42

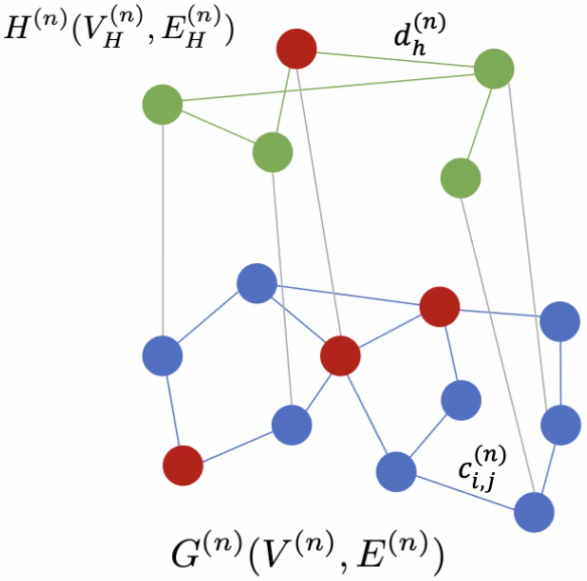
Some the node fail:

Flow routing

Node i is repaired at step n

Link (i,j) is repaired at step n

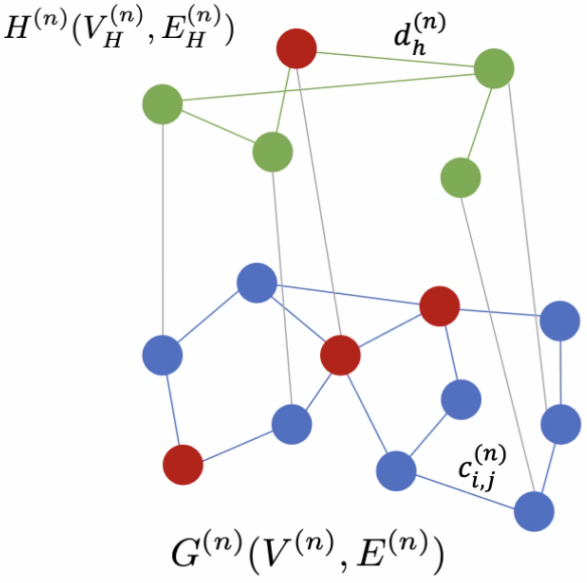
$$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V,\ (i,j) \in E$$
$$f_{ij}^h(n) \geq 0, \quad h \in E_H$$



Flow routing

$$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V,\ (i,j) \in E$$
$$f_{ij}^h(n) \geq 0, \quad h \in E_H$$
$$\alpha_h(n) \in [0,1], \quad h \in E_H$$

Percentage of flow for demand h at step n



Flow routing

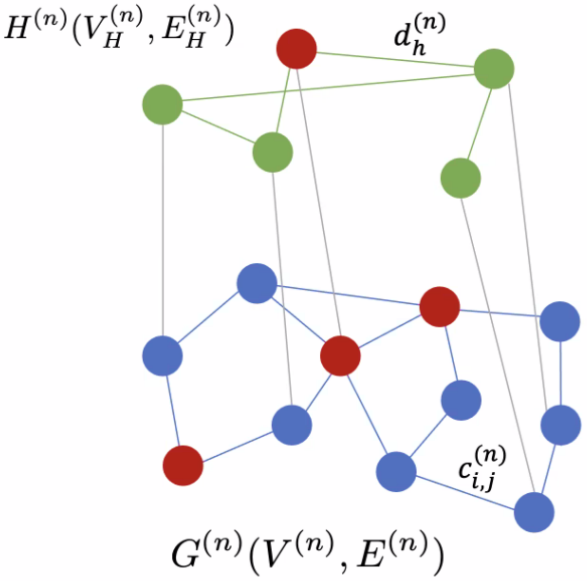
$\text{Max} \sum_{n=1}^{N} \sum_{h \in E_H} d_h \cdot \alpha_h(n)$

We want to maximize the cumulative flow over *N* steps

$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V,\ (i,j) \in E$

$f_{ij}^h(n) \geq 0, \quad h \in E_H$

$\alpha_h(n) \in [0,1], \quad h \in E_H$



Flow routing

$\text{Max} \sum_{n=1}^{N} \sum_{h \in E_H} d_h \cdot \alpha_h(n)$

$c_{ij} \cdot \sum_{k=0}^{n} x_{ij}(k) \geq \sum_{h=1}^{|E_H|} (f_{ij}^h(n) + f_{ji}^h(n)), \quad \forall (i,j)$

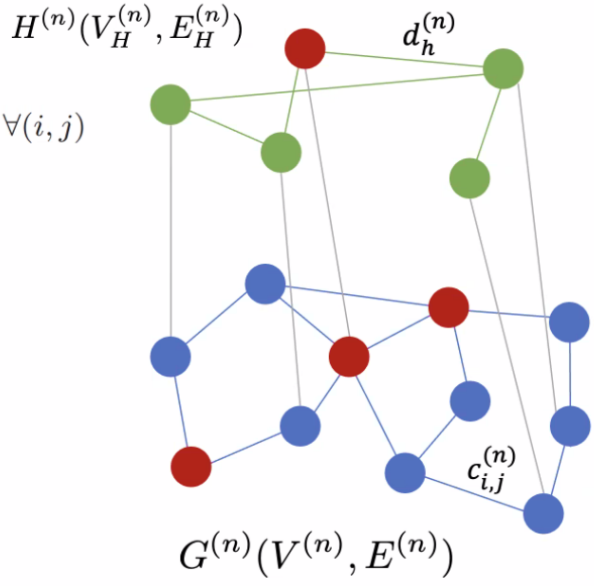
$\sum_{k=0}^{n} y_i(k) \geq x_{ij}(n), \quad \forall (i,j)$

If a link is working/has been repaired,
also its end points must be working or repaired

$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V,\ (i,j) \in E$

$f_{ij}^h(n) \geq 0, \quad h \in E_H$

$\alpha_h(n) \in [0,1], \quad h \in E_H$



Flow routing

$$\text{Max} \sum_{n=1}^{N} \sum_{h \in E_H} d_h \cdot \alpha_h(n)$$

$$c_{ij} \cdot \sum_{k=0}^{n} x_{ij}(k) \geq \sum_{h=1}^{|E_H|} (f_{ij}^h(n) + f_{ji}^h(n)), \quad \forall (i,j)$$

$$\sum_{k=0}^{n} y_i(k) \geq x_{ij}(n), \quad \forall (i,j)$$

$$\sum_{j \in V} f_{ij}^h(n) = \sum_{k \in V} f_{ki}^h(n) + b_i^h \cdot \alpha_h(n), \quad \forall (i,h)$$

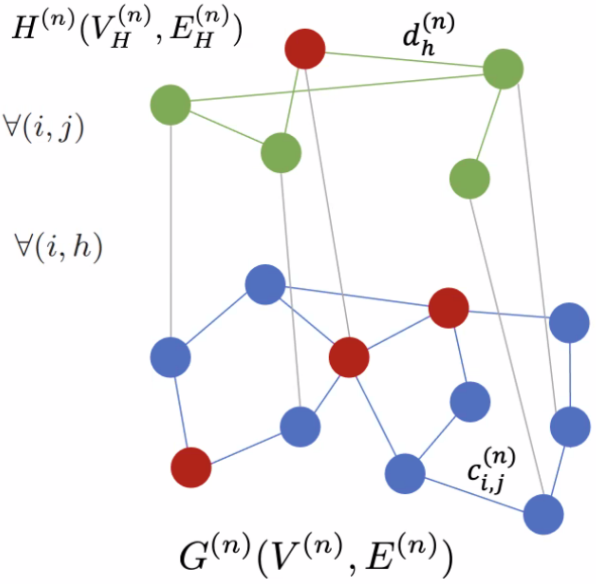
$$b_i^h = \begin{cases} d_h & \text{If } i \text{ is the source of the flow} \\ -d_h & \text{If } i \text{ is the destination of the flow} \\ 0 & \text{otherwise} \end{cases}$$

Flow balance constraint

$$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V, \ (i,j) \in E$$

$$f_{ij}^h(n) \geq 0, \quad h \in E_H$$

$$\alpha_h(n) \in [0,1], \quad h \in E_H$$



Flow routing

$$\text{Max} \sum_{n=1}^{N} \sum_{h \in E_H} d_h \cdot \alpha_h(n)$$

$$c_{ij} \cdot \sum_{k=0}^{n} x_{ij}(k) \geq \sum_{h=1}^{|E_H|} (f_{ij}^h(n) + f_{ji}^h(n)), \quad \forall (i,j)$$

$$\sum_{k=0}^{n} y_i(k) \geq x_{ij}(n), \quad \forall (i,j)$$

$$\sum_{j \in V} f_{ij}^h(n) = \sum_{k \in V} f_{ki}^h(n) + b_i^h \cdot \alpha_h(n), \quad \forall (i,h)$$

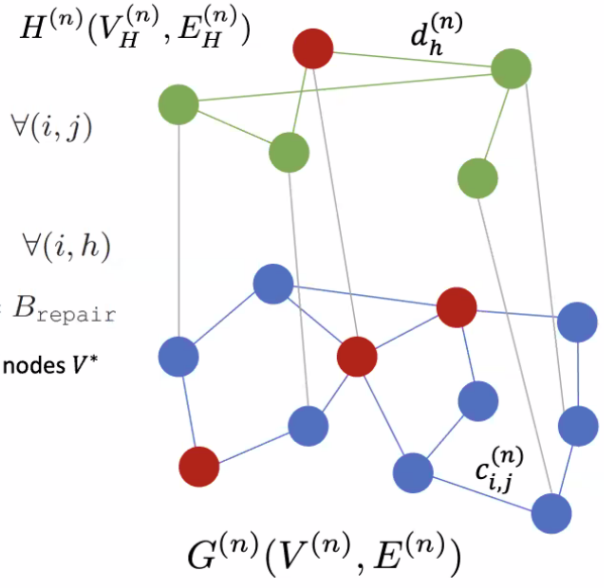
$$\sum_{(i,j) \in E^*} x_{ij}(n) \cdot k_{ij}^e + \sum_{i \in V^*} y_i(n) \cdot k_i^v <= B_{\text{repair}}$$

The cost κ_{ij}^e of repaired links E^* and the cost κ_i^v of repaired nodes V^* have to be lower than a given repair budget

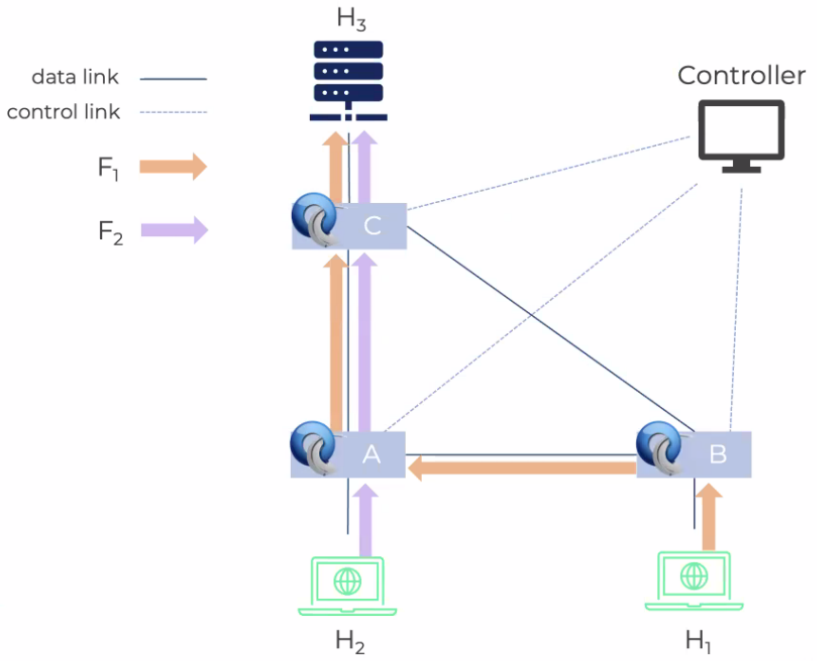
$$y_i(n), x_{ij}(n) \in \{0,1\}, \quad \forall i \in V, \ (i,j) \in E$$

$$f_{ij}^h(n) \geq 0, \quad h \in E_H$$

$$\alpha_h(n) \in [0,1], \quad h \in E_H$$



Initial scenario

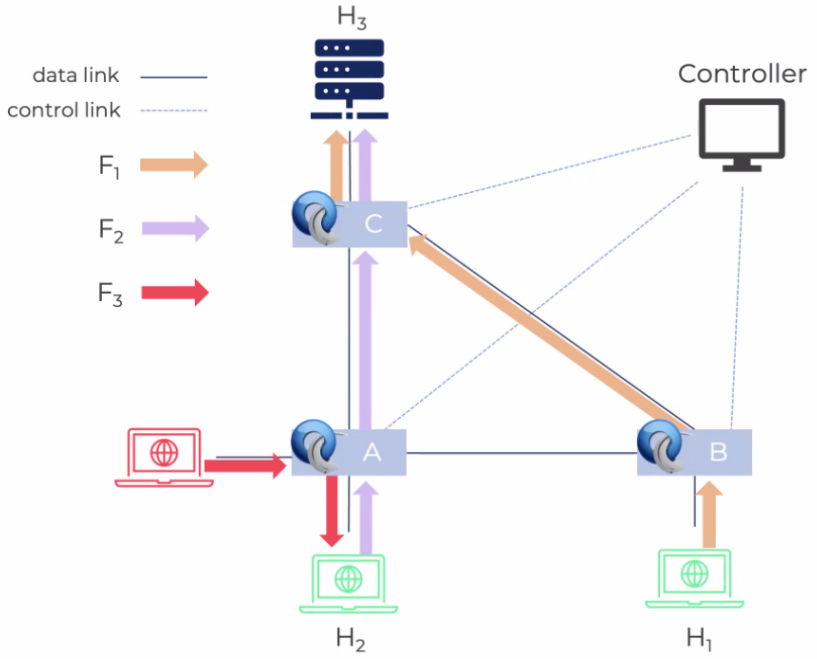


...back to the experiments

First scenario

Attacker injects malicious flow to pollute switch A's table

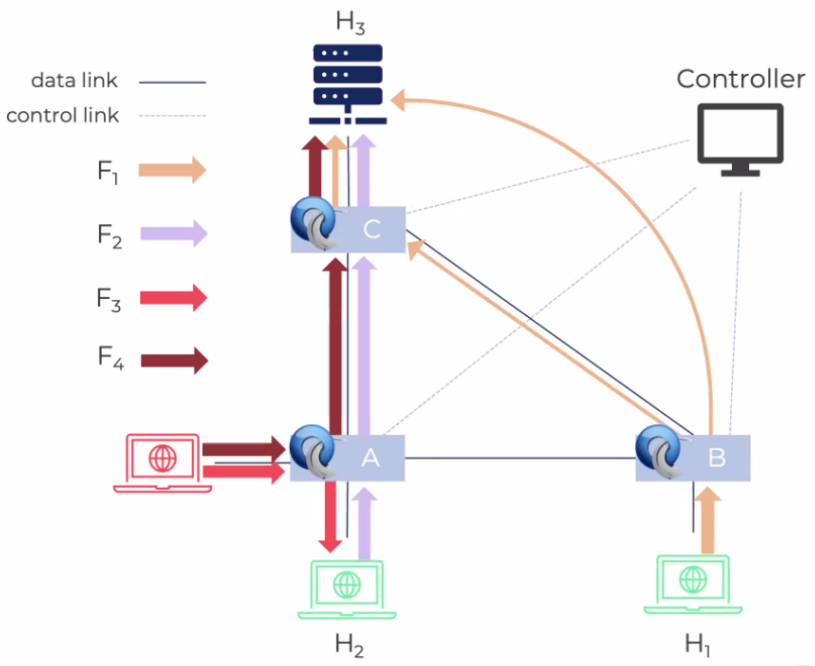
F_1 is rerouted to switch C.



Second scenario

Attacker generates a new flow F_4 to pollute switches A and C's tables

Part of F_1 is rerouted to switch C.



Third scenario

Malicious flow F_4 increases and pollutes switches A and C's tables

Flow F_1 is rerouted to H_3.