Topology Management

Prof. C. Tschudin, M. Sifalakis, T. Meyer,
G. Bouabene, M. Monti

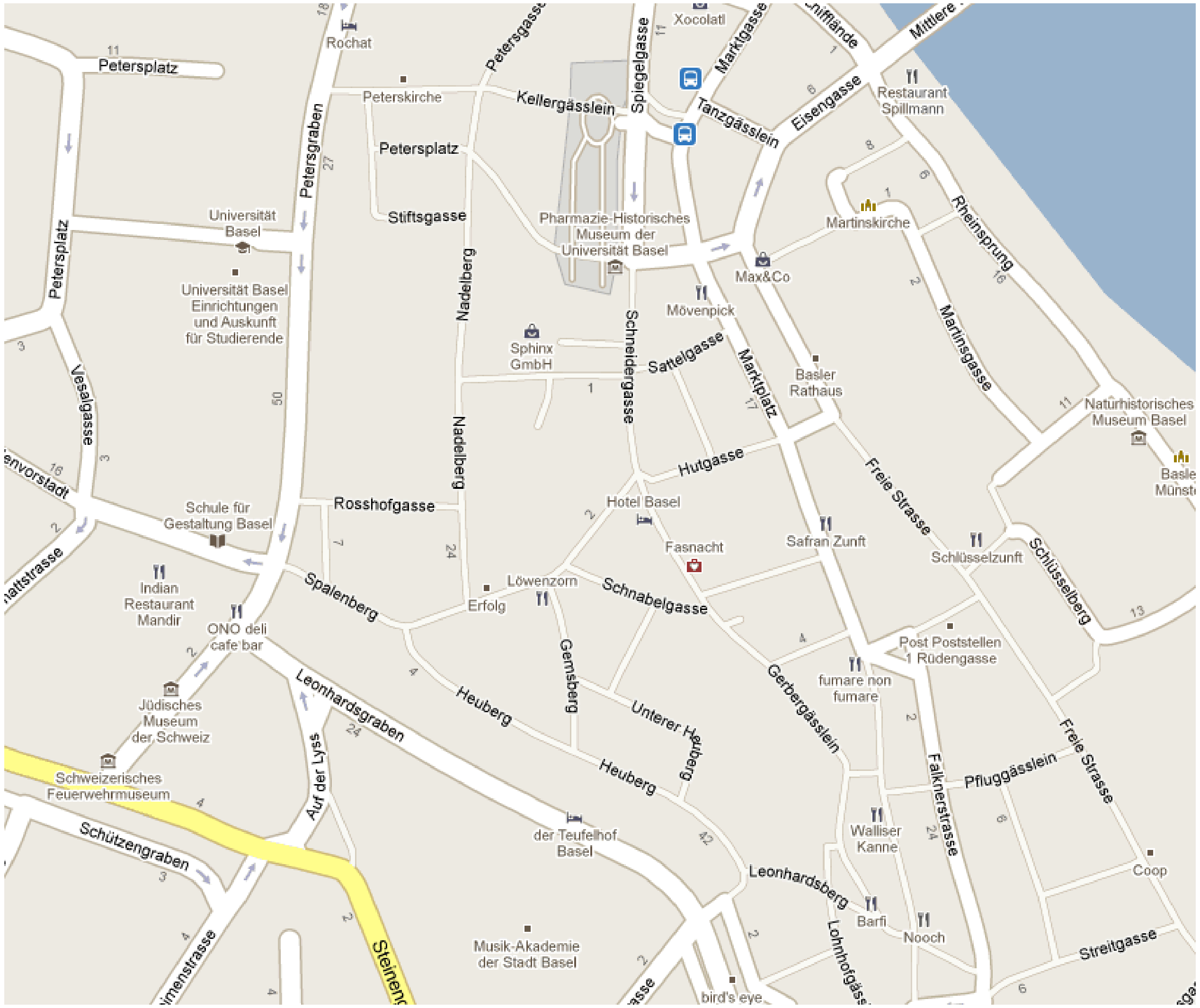
University of Basel

Cs321 - HS 2011

Overview

- Topology and Connectivity, A real life analogy
- Basics of Graph Theory
- Net topology - Design phase
  - Maximise Utility, Max-Flow Min-Cut theorem
  - Reliability, Edge and Node connectivity
- Net topology - Operational Phase
  - Common problems
  - Operational efficiency, Spanning Trees, ST protocol
  - The wireless setting, Common Dominant Sets, MPR protocol
- Topology support mechanisms
  - From cables to virtual circuits
  - Multi Protocol Label Switching (MPLS)
  - Metro/Carrier Ethernet

Topology and Connectivity: Real life example



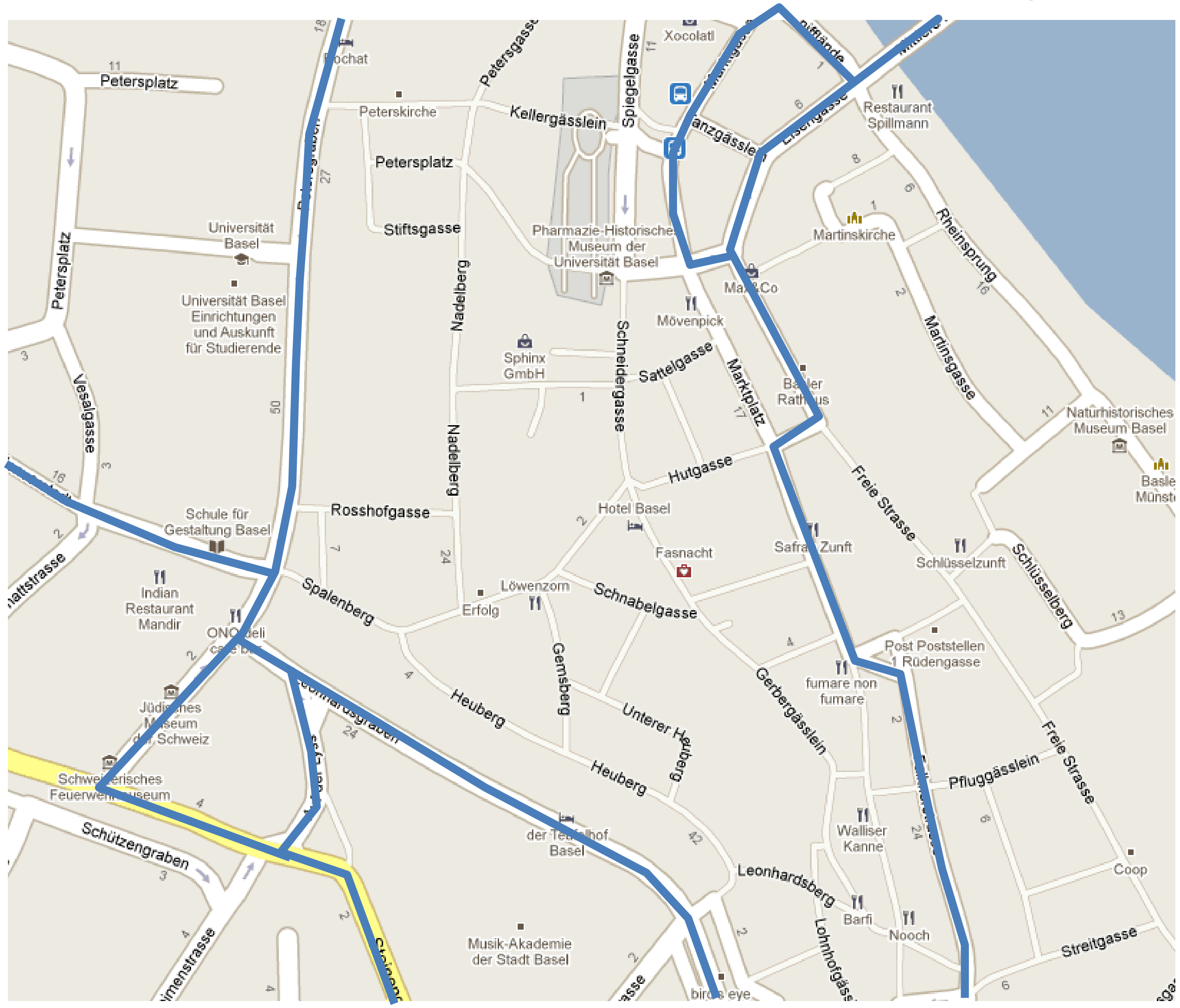
Part of the road network in Basel, interconnecting a number of points in the city.

One may be say that this is the "transportation network topology" in Basel

However, this map shows only the transportation potential (medium). Much like wires or air provide a medium for communication potential

… in fact, if you try to drive with a car across all the roads of this map you will either make a few people angry or end up with a police ticket!

Topology and Connectivity: Real life example



Where car traffic can go through

… this is actually a bit more useful to car drivers

Topology and Connectivity: Real life example

Where velo traffic can go through

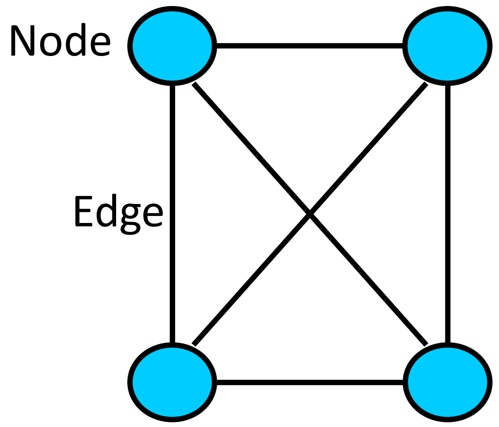
So when we talk about network topology, we don't refer just to the cabling!

Rather cables and connectivity is used to organise the flow of information. This organisation we refer to as network topology

Some use the term physical and logical topology to distinguish between the two, although what we mostly care about is the latter

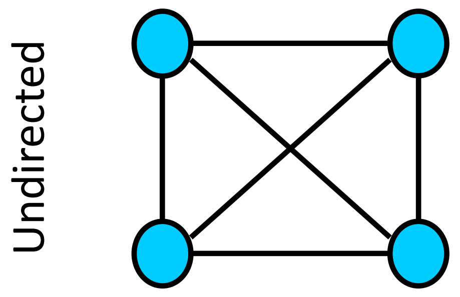
Some Graph Theory terminology

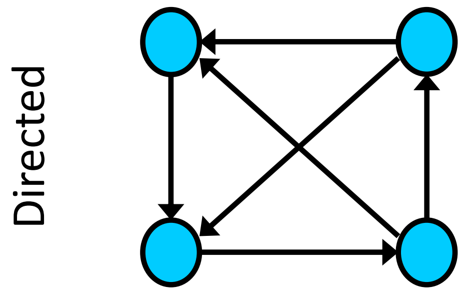
- Graph G = (V, E)
  - nodes V (aka vertices)
  - edges E (aka arcs, links)

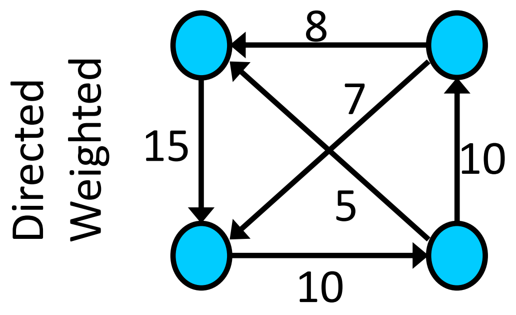


Some Graph Theory terminology

- Graph G = (V, E)
  - nodes V (aka vertices)
  - edges E (aka arcs, links)
    - undirected
    - directed
    - weighted (capacity, delay, cost, …)

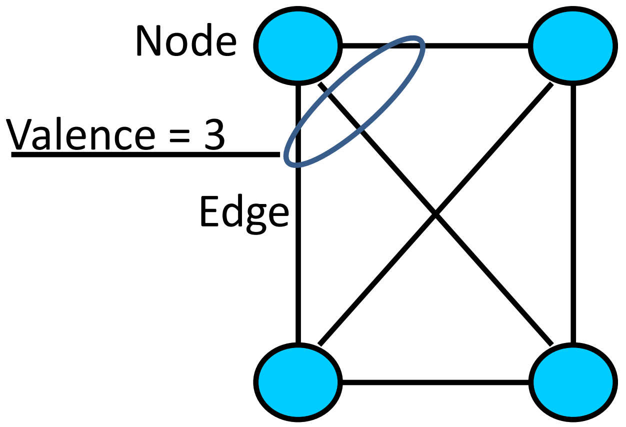






Some Graph Theory terminology

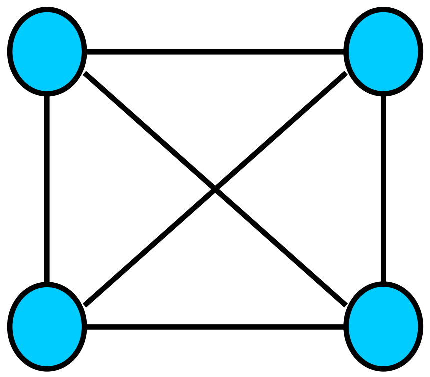
- Graph G = (V, E)
  - nodes V (aka vertices)
    - degree (aka valence)
  - edges E (aka arcs, links)
    - undirected
    - directed
    - weighted (capacity, delay, cost, …)
- D-regular graph: all nodes have degree D

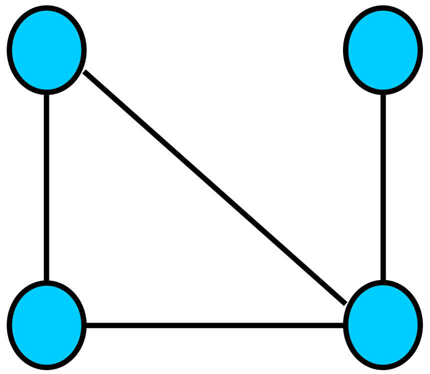


3-regular graph

Some Graph Theory terminology

- Graph G = (V, E)
  - nodes V (aka vertices)
    - degree (aka valence)
  - edges E (aka arcs, links)
    - undirected
    - directed
    - weighted (capacity, delay, cost, …)
- D-regular graph: all nodes have degree D
- Graphs can be fully …or partially connected

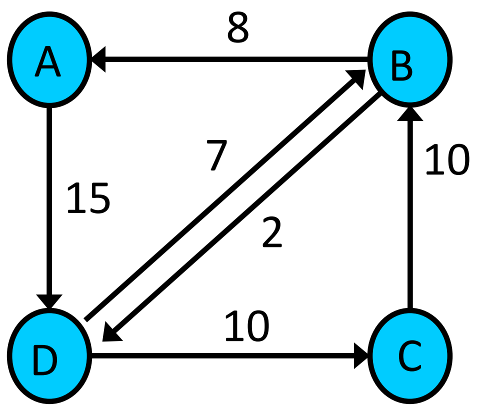

Fully Connected


Partly Connected

Graphs and computers

- Representing Graphs in Matrix form

		to			
		A	B	C	D
from	A	-	-	-	15
	B	8	-	-	2
	C	-	10	-	-
	D	-	7	10	-



From graphs to network topology

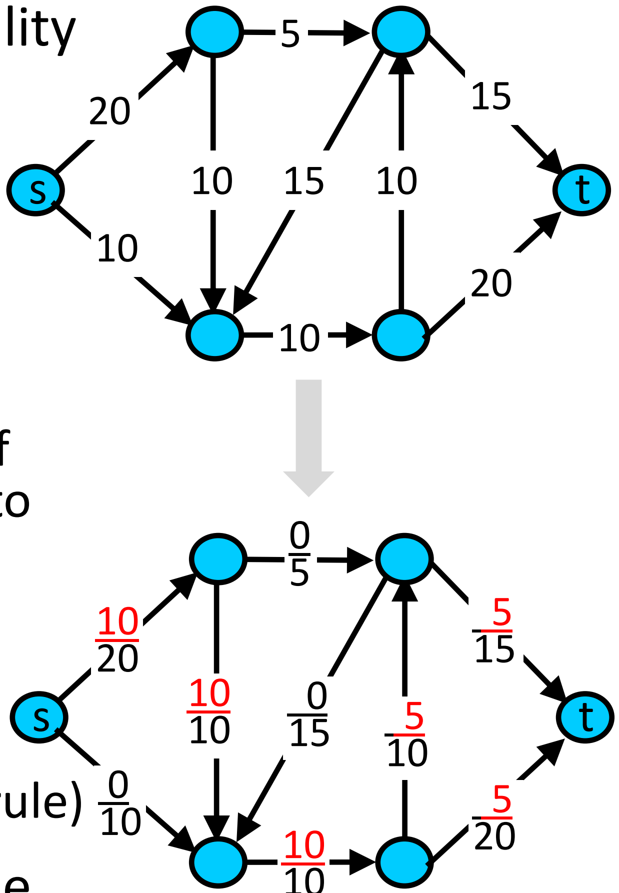
- We have a set of nodes (and connectivity potential) and we want to decide what is the best way of organising them in a network topology
- Two main sets of objectives
  - Design phase objectives/challenges
  - Operational phase objectives/challenges
- Let’s start with the former

Objective 1: Maximum utility network

- Problem: We want to know the maximum utility of the network when any two pair of nodes communicate
- (*s*-*t*) flow
  - Information flowing from *s* towards *t*
  - If $c(u \rightarrow v)$ is the nominal capacity of a communication link, then s-t flow is feasible if there exists a function *f* that assigns weights to edges such that

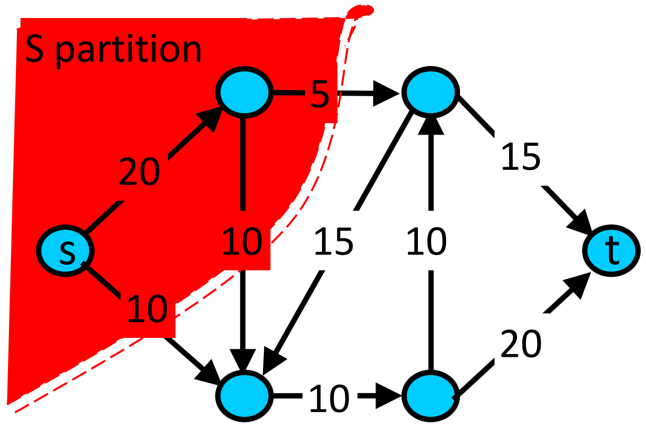
    $0 \leq f(u \rightarrow v) \leq c(u \rightarrow v)$, and
  - For each graph node other than *s*,*t*:

    $\sum f(u \rightarrow v) = \sum f(w \rightarrow u)$ (flow conservation rule)
- What is the maximum feasible (*s*-*t*) flow in the network (for every pair *s, t* in V) ?



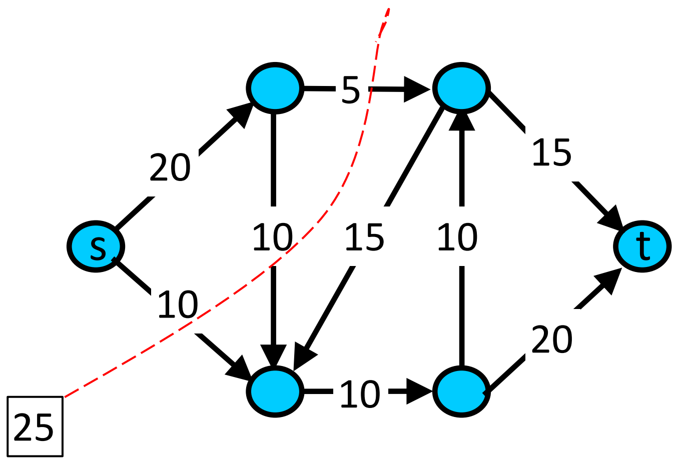
Some more Graph Theory terminology

- Graph cut (S,T)
  - A node partitioning such that S∪T = V, S∩T = ∅, *s* is in S and *t* is in T
  - A set of edges whose removal disconnects node s from node t



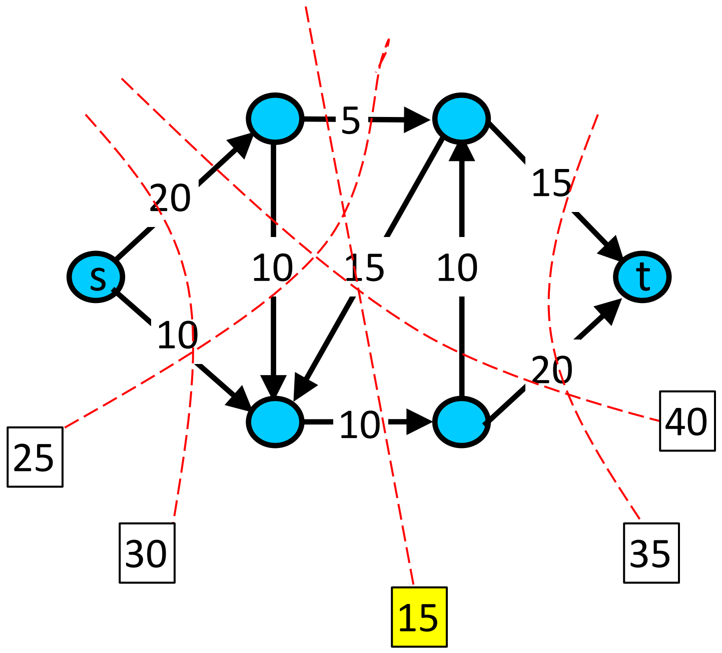
Some more Graph Theory terminology

- Graph cut (S,T)
  - A node partitioning such that S∪T = V, S∩T = ∅, *s* is in S and *t* is in T
  - A set of edges whose removal disconnects node s from node t
- Cut capacity
  - Sum of weights of the directed edges from partition S to partition T



Some more Graph Theory terminology

- Graph cut (S,T)
  - A node partitioning such that S∪T = V, S∩T = ∅, *s* is in S and *t* is in T
  - A set of edges whose removal disconnects node s from node t
- Cut capacity
  - Sum of weights of the directed edges from partition S to partition T
- Minimum cut
  - The graph cut with the minimum capacity



Max-Flow Min-Cut Theorem

The value of the maximum feasible flow is equal to the capacity of the minimum cut

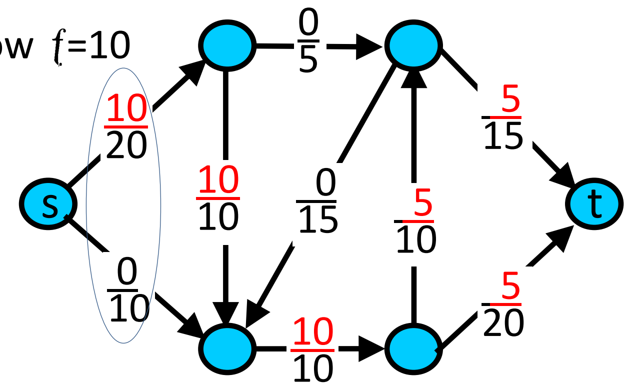
- The minimum cut determines the bottleneck in the flow of information between *s* and *t*
- Essentially an upper bound for the maximum feasible flow
- How does the maximum feasible flow map to our network ?

Ford-Fulkerson algorithm

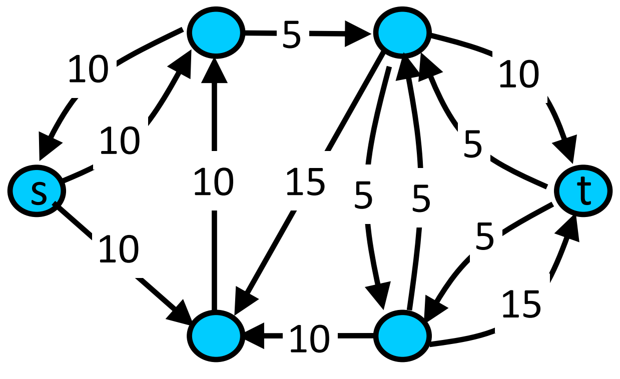
1. Let f be an initial feasible (s-t) flow.
2. We define as residual capacity a function that assigns new weights in the graph edges

$$c_f = \begin{cases} c(u \rightarrow v) - f(u \rightarrow v) & \text{if } u \rightarrow v \text{ in E} \\ f(u \rightarrow v) & \text{if } v \rightarrow u \text{ in E} \\ 0 & \text{otherwise} \end{cases}$$

Initial flow f=10



Original graph G



Residual graph G_f

Example (and more mathematical explanations) can be found in the notes of Jeff Erickson
http://www.cs.uiuc.edu/~jeffe/teaching/algorithms/notes/21-maxflow.pdf

Ford-Fulkerson algorithm

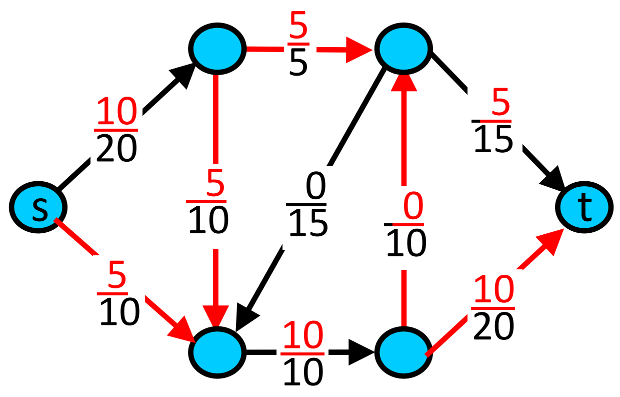
3. Check if there is a path from *s* to *t* in G_f : *augmenting path A*

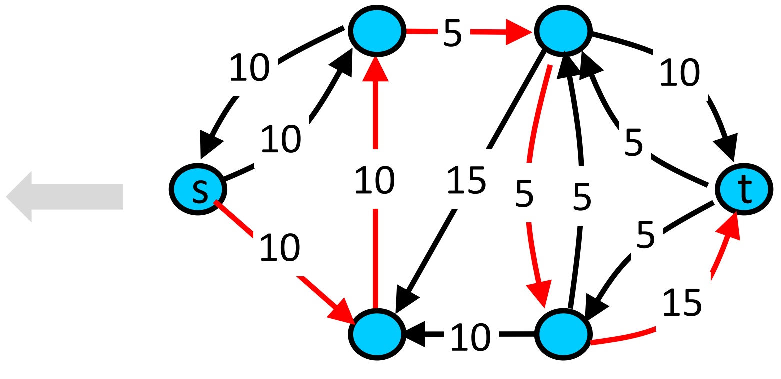
   If >1, select the one with largest bottleneck value (Edmonts-Karp '72) or the shortest (Dinic '70)

4. Let $F = \min \{c_f(v_i \rightarrow v_{i+1})\}$

5. Travel in G, along *A* of G_f , and update the flow weights as follow, then back to step 2

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + F & \text{if } u \rightarrow v \text{ in A} \\ f(u \rightarrow v) - F & \text{if } v \rightarrow u \text{ in A} \\ 0 & \text{otherwise} \end{cases}$$


Original graph G


Residual graph G_f

Ford-Fulkerson algorithm

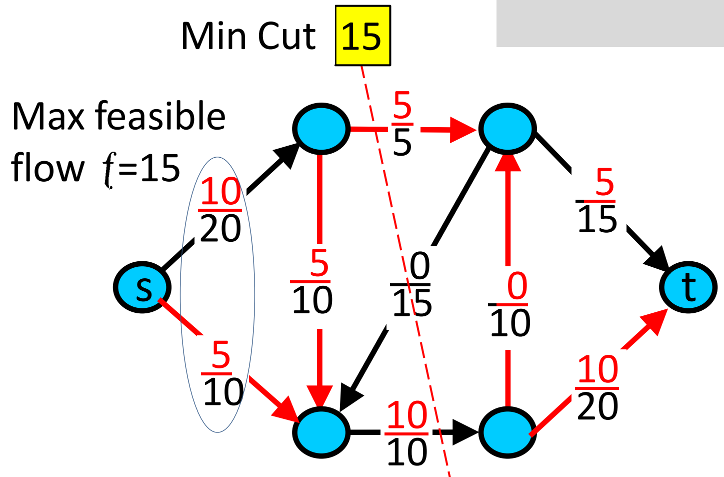
3. Check if there is a path from *s* to *t* in G_f : *augmenting path A*

   If >1, select the one with largest bottleneck value (Edmonts-Karp '72) or the shortest (Dinic '70)

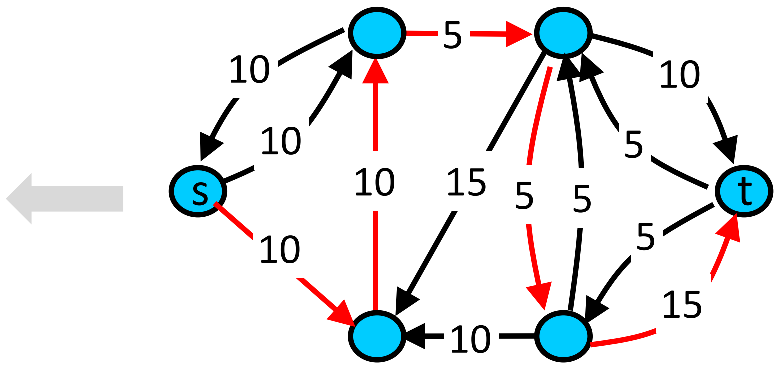
4. Let $F = \min\{c_f(v_i \rightarrow v_{i+1})\}$

5. Travel in G, along *A* of G_f , and update the flow weights as follow, then back to step 2

$$f'(u \rightarrow v) = \begin{cases} f(u \rightarrow v) + F & \text{if } u \rightarrow v \text{ in A} \\ f(u \rightarrow v) - F & \text{if } v \rightarrow u \text{ in A} \\ 0 & \text{otherwise} \end{cases}$$



Original graph G

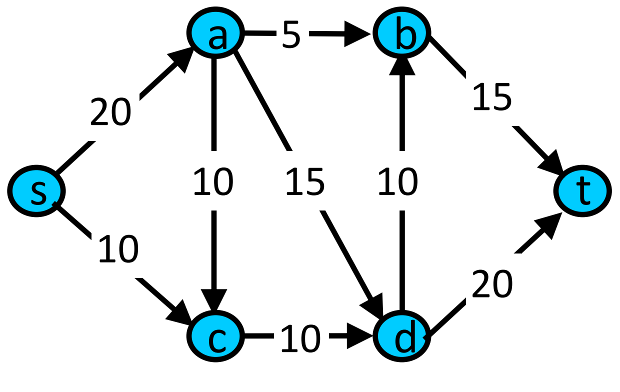


Residual graph G_f

Objective 2: Reliability

- Problem: We want the network to sustain the loss of links or nodes without becoming disconnected
  - If it sounds somewhat relevant to Minimum Cuts, that's because it is ! ... more soon
- *s*-*t* edge/node connectivity
  - Minimum number of edges/nodes that must be removed from a graph to disconnect node s from node t
  - Note this does not necessarily imply partitioning of the graph
  - Intuitively node loss is a stronger condition since it will always result in edge loss (the reverse is not necessarily true)
- What is the *s*-*t* minimum node/edge connectivity of the network (for every *s*, *t* in V) ?

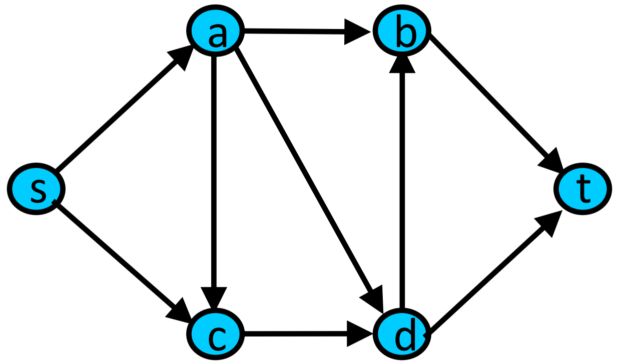
Back to Graph Theory terminology



- Graph path
  - Sequence of adjacent nodes. E.g. {*sabt*}
  - ... or the set of links connecting them
- Path length
  - Sum of weights along the path E.g. {*sabt*}=*40*
- Edge disjoint paths
  - Paths between a pair of nodes that have no edges in common. E.g. {*sadbt*}, {*scdt*}
- Node disjoint paths
  - Paths between a pair of nodes that share no edges and no nodes. E.g. {*sabt*}, {*scdt*}
- Node connectivity C_n (resp. edge connectivity C_e) of Graph
  - The minimum node connectivity (resp. edge connectivity) among all pairs of nodes
- *K* node-connected (resp. *K* edge-connected) Graph
  - When all pairs of nodes are *k* node-connected (resp. *k* edge-connected)

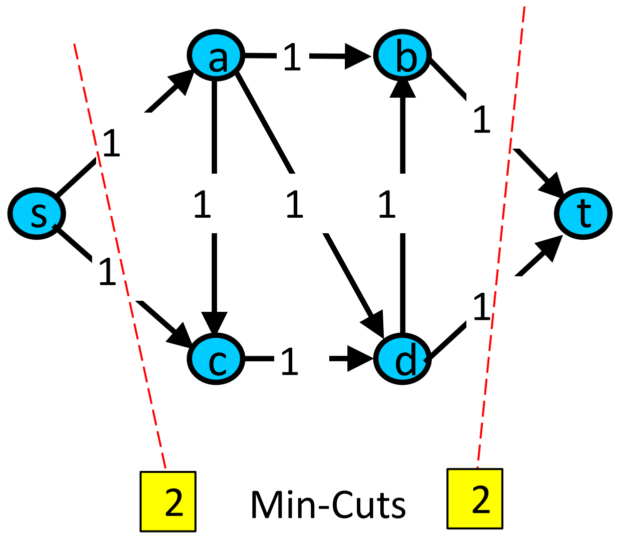
Reliability and Min-Cuts

- Observation
  - If there are *k* edge-disjoint *s*-*t* paths, then removing at least *k* edges (1 from each path) will disconnect *s* from *t*



Edge-connectivity and Min-Cuts

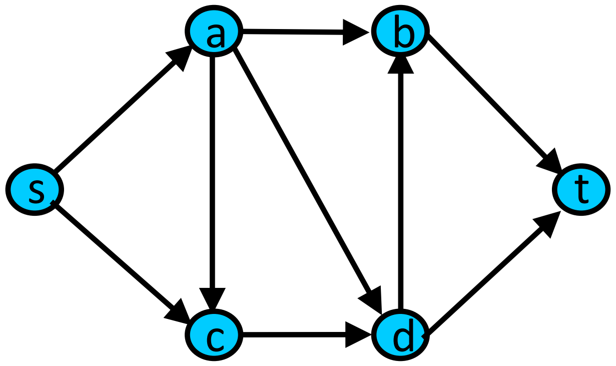
- Observation
  - If there are *k* edge-disjoint *s*-*t* paths, then removing at least *k* edges (1 from each path) will disconnect *s* from *t*
- Max-Flow Min-Cut theorem with unit weights
  - Set all the graph weights to 1, then the Min-Cut capacity enumerates the number of edge disjoint paths for the pair *s*-*t* (!)
  - Solve the problem by applying Max-Flow Min-Cut for every *s*-*t* in V ;-)



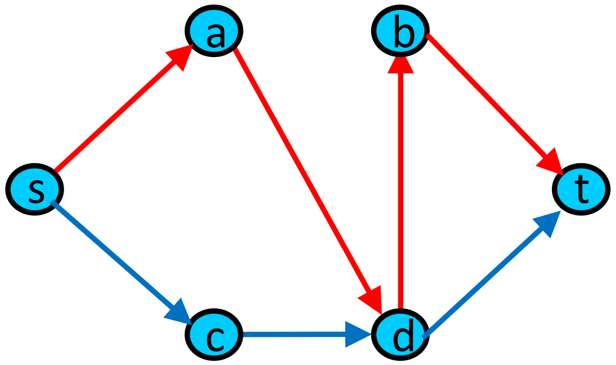
Edge-connectivity or Node-connectivity ?

- However, node connectivity is a stricter requirement than edge connectivity
  - removing a node deletes ≥1 edges from the graph, the reverse is not necessarily true
  - $C_n \leq C_e \leq$ min node degree in graph (Whitney '32)
- So node connectivity sets a lower acceptable bound for edge connectivity. Problem reduces to how to determine the minimum node-connectivity of a Graph
- Solution:
  - Convert the problem of finding node-disjoint paths to one that involves edge-disjoint paths... then
  - Apply again the Max-Flow Min-Cut method with unit weights

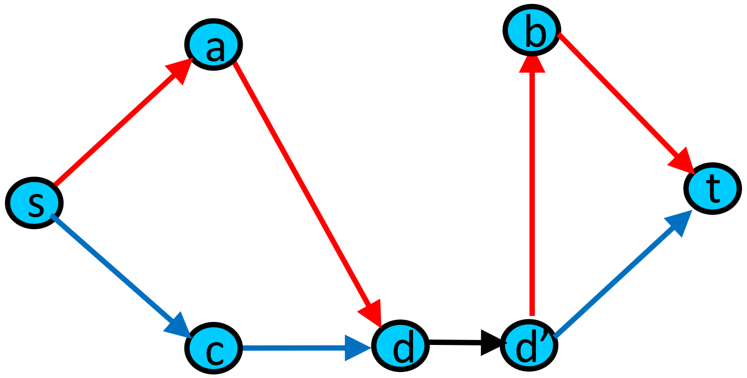
From node-disjoint to edge-disjoint



Edge-disjoint: {*sadbt*}-{*scdt*}, {*sabt*}-{*scdt*}
Node-disjoint: {*sabt*}-{*scdt*}



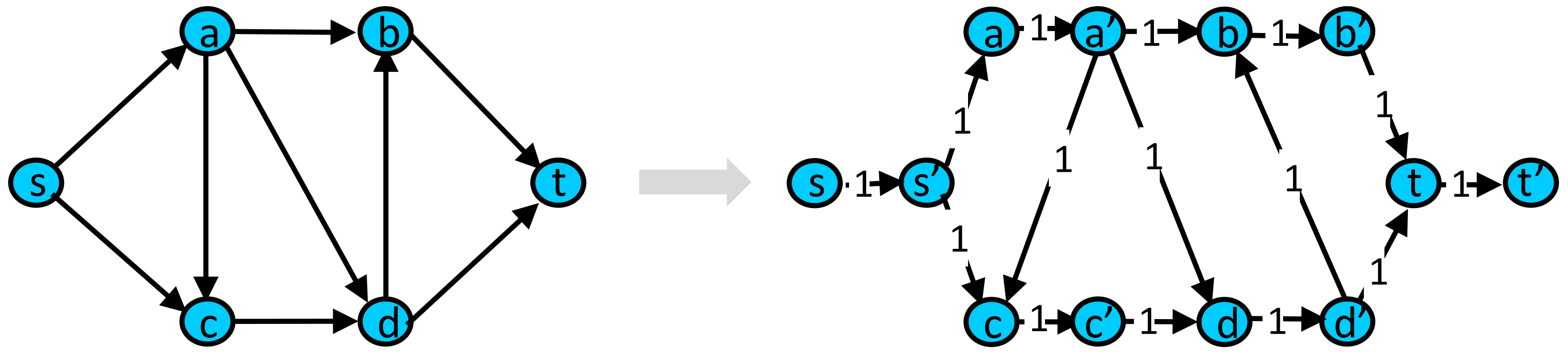
Problem: need to get rid of {*sadbt*}-{*scdt*}
Observation: {*sadbt*}, {*scdt*} share node *d*



Solution: for each shared node add a shared edge. Now {*sadbt*}-{*scdt*} are not edge-disjoint

Node-connectivity with Max-Flow Min-Cut

1. For every node X in the graph add a second node X’ and connect them with an edge X→X’
2. Move all outgoing edges from node X to X’, leaving the incoming edges intact
3. Set a weight of 1 on all edges of the graph
4. Apply Max-Flow Min-Cut to find the number of node-disjoint paths

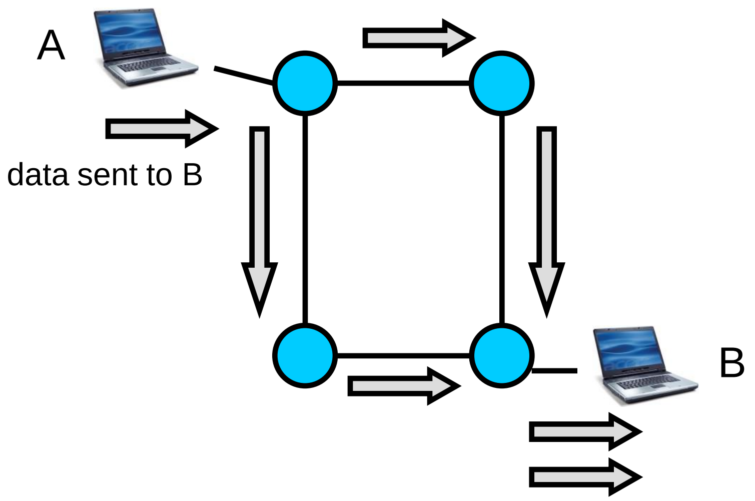


From design to operation

- Designing a network
  - Problem is complex with often in-tractable solutions
  - Often most effective approach is to generate a potential topology (based on experimental intuition), and test if it satisfies the constraints. If not, try to improve it or generate another candidate. Repeat until criteria met.
- OK, so now we have a topology. What about topology management and operational challenges

Commonly experienced problems in closed networks topologies #1

- Duplicated traffic

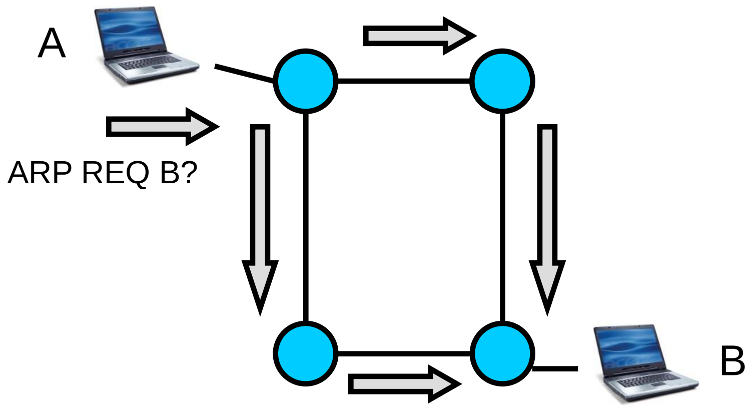


Duplicate frame is received!

After all a reliable topology design protects against packet loss!

Commonly experienced problems in closed networks topologies #2

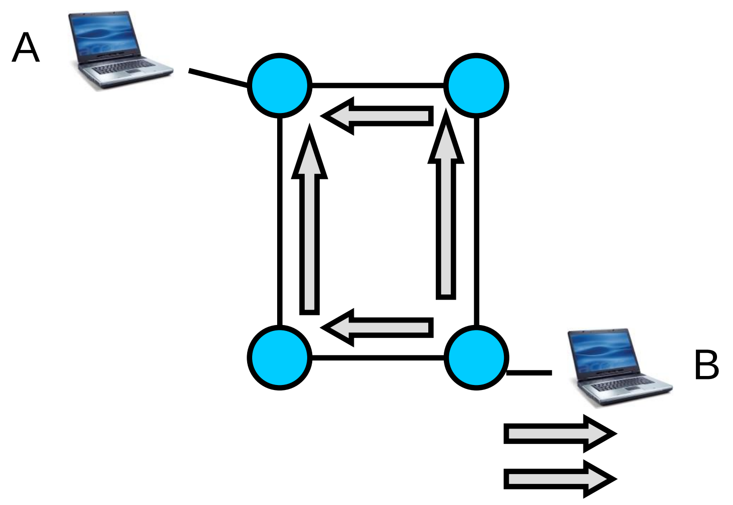
- Broadcast storms



Commonly experienced problems in closed networks topologies #2

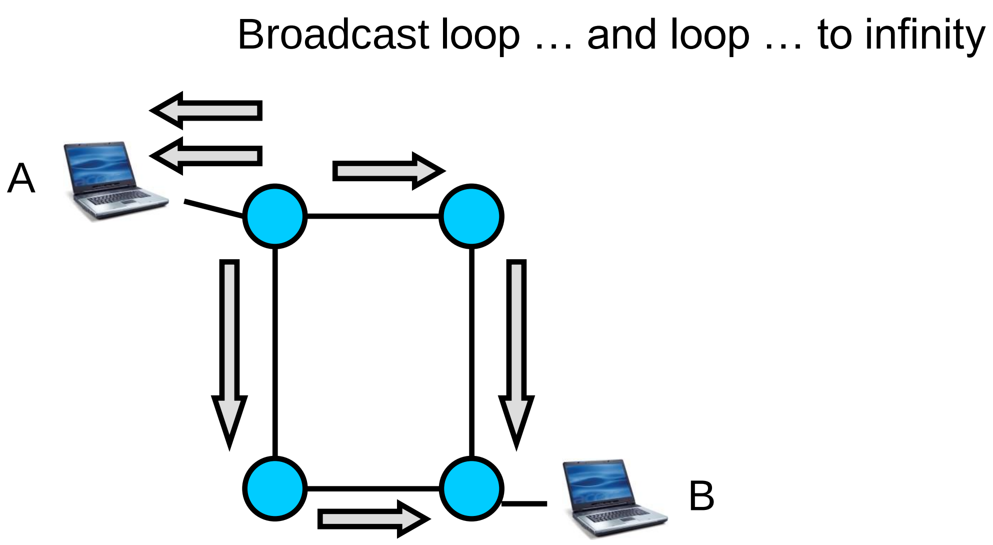
- Broadcast storms

Broadcast loop



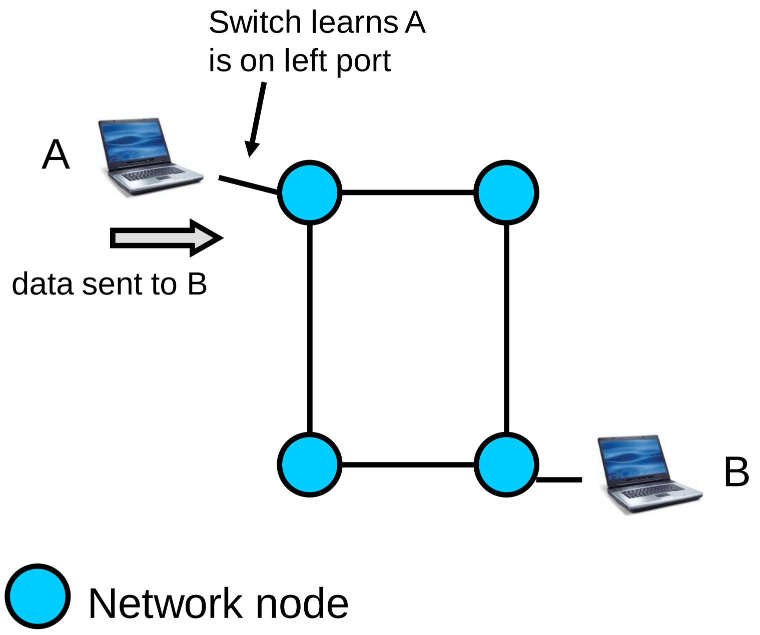
Commonly experienced problems in closed networks topologies #2

- Broadcast storms



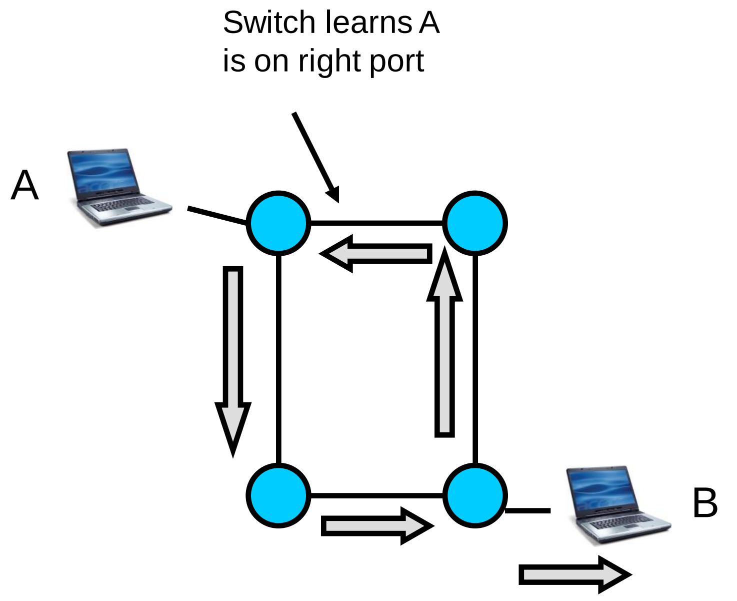
Commonly experienced problems in closed networks topologies #3

- Inconsistent network state (broadcast nodes)



Commonly experienced problems in closed networks topologies #3

- Inconsistent network state (broadcast nodes)



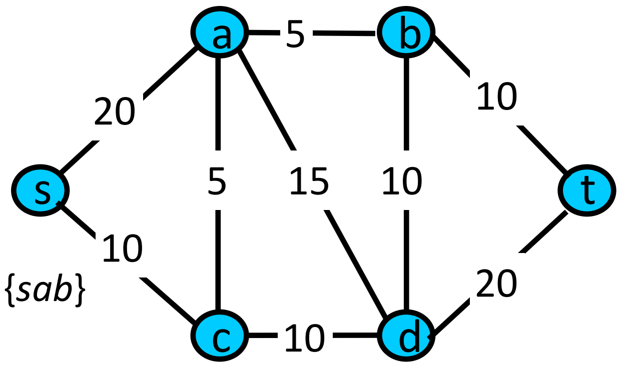
Design objectives versus operational objectives

- And there are more, similar, but nastier problems
- Seems that our design objectives contradict operational simplicity
- This is only partly true!
  - We can keep our reliable topology
  - But distinguish between main operational parts and back up parts

Objective 3: Operational efficiency

- So we need to identify open sub-graphs within the connectivity graph
- Problem: Which open sub-graphs are best candidates for the operational topology
  - Maximise throughput
  - Minimise delays
  - Avoid loops

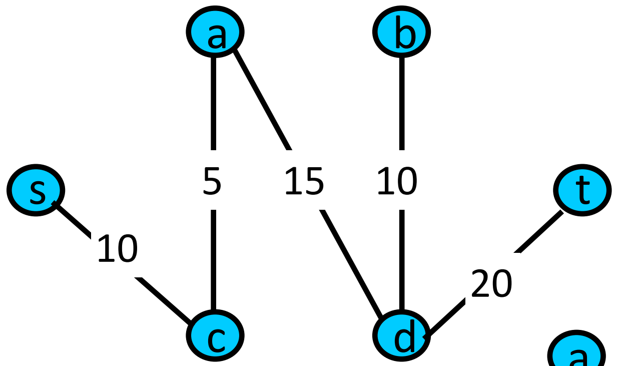
Need more Graph Theory terminology

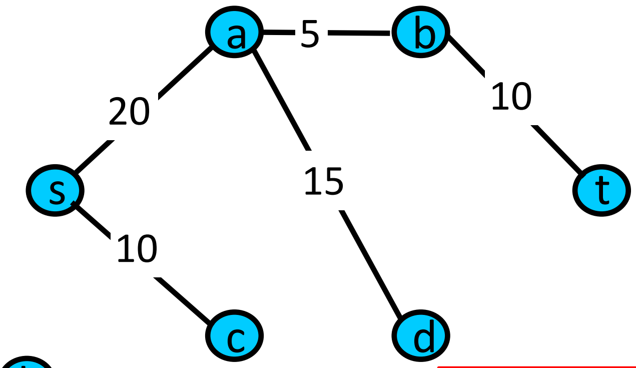


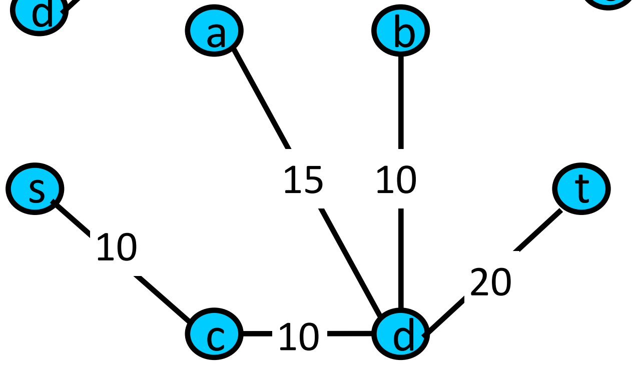
- Hamiltonian paths
  - include all the nodes in the graph. E.g: {*abtdcs*}, {*scadbt*}
- Geodesic path
  - Shortest path between two nodes. E.g: Geo(s-b)={*scab*} not {*sab*}
- Network diameter
  - Length of longest geodesic path in the graph. E.g: Geo(s-t)={scabt }=30
- Cycle
  - A path that visits the same node more than once. E.g: {*cdabdt*}
- Circuit
  - A cycle that starts and ends on the same node. E.g: {*sabtdcs*}
- Tree
  - A connected graph without circuits or cycles
  - one and only one path joins any pair of nodes
  - every edge is a cut
  - N nodes, N-1 edges

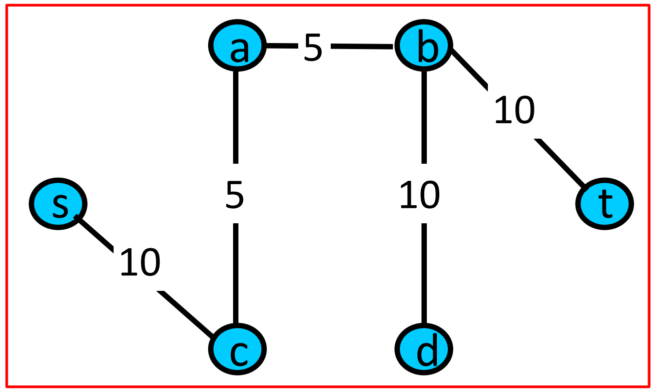
Need more Graph Theory terminology

- Spanning tree
  - A sub-graph that is a tree and spans all vertices of the original graph
- Minimum spanning tree
  - Among all spanning trees of a <u>weighted</u> graph, the one (possibly more) with the smallest sum of weights









Computing minimum spanning trees

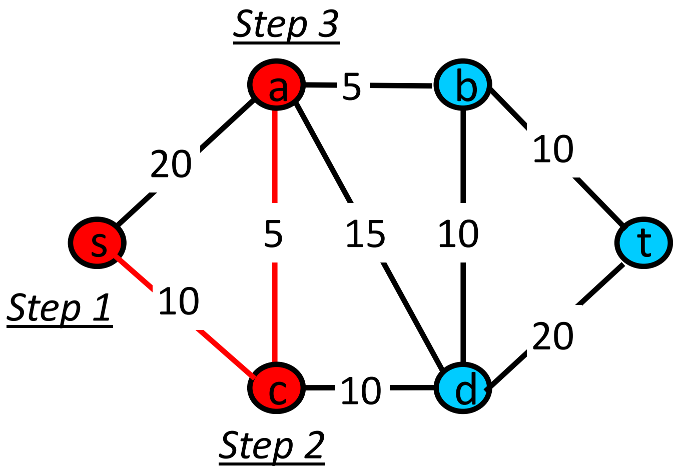
- So … if the weights of a graph represent the inverse of the capacities, finding a minimum spanning tree seems to solve the problem!
- 2 simple algorithms in broad use today
  - Prim ’57 (Also Jarník ’30, Dijkstra ’59)

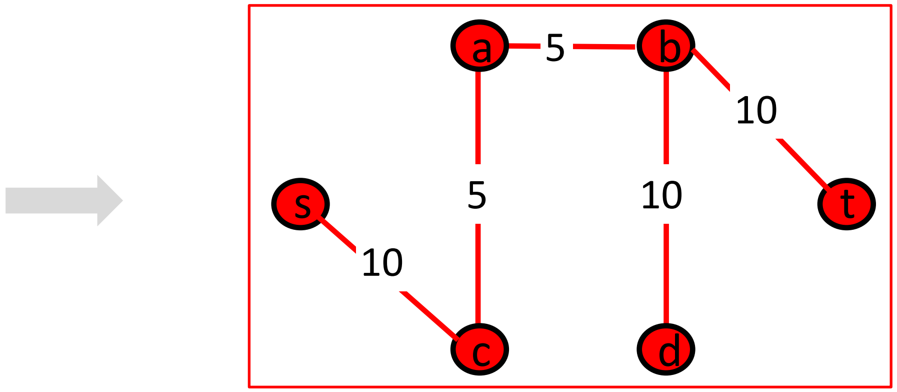
    O(n^2) or O(m log n) complexity depending on implementation (m edges, n nodes in graph)
  - Kruskal ’56

    O(m log m) = O(m log n) complexity, (m edges, n nodes in graph)
- And a de-facto standard protocol that all (?) off-the-shelf switching products today implement
  - Perlman ’85, distributed version of spanning tree algorithm

PJD's algorithm

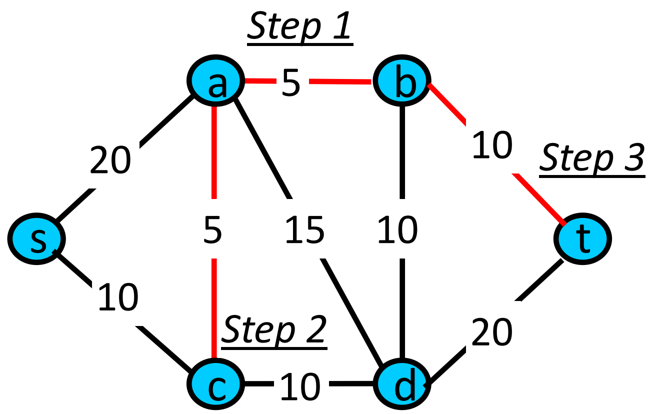
1. Pick any vertex as a starting node (call it S) and color it (e.g. red)
2. Find the nearest neighbor of S (call it P_1). Color P_1 and edge S-P_1
3. Find the nearest uncolored neighbor to the colored sub-graph (call it P_2). Color P_2 and the edge connecting the node to the rest of the colored sub-graph
4. Repeat Step 3 until all nodes are colored. The colored sub-graph is a minimum spanning tree

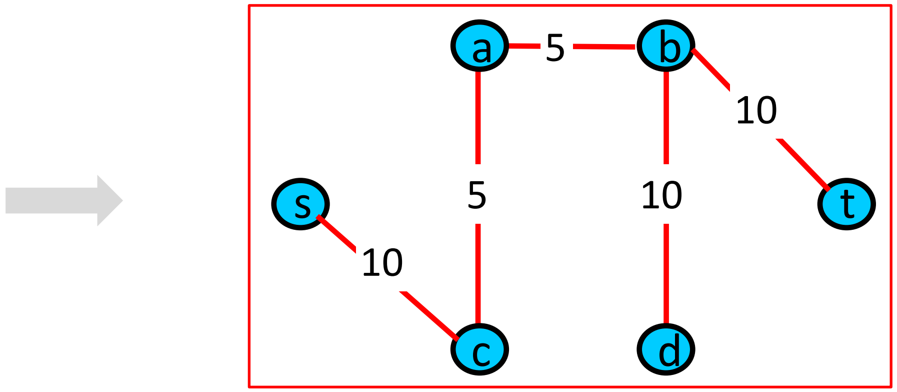




Kruskal's Algorithm

1. Find the cheapest edge in the graph (if more than one, pick one at random) and color it (e.g. red)
2. Find the cheapest uncolored edge in the graph that doesn't close a colored circuit and color it
3. Repeat Step 2 until all vertices are connected (or until you have *m* colored edges – for the *m* nodes). The colored edges form a minimum spanning tree





Perlman’s Spanning Tree Protocol – informational only –

<u>Phase A: Leader/ROOT NODE election</u>

1. On start-up all nodes start exchanging, with their neighbors, configuration BPDU messages that contain the Bridge ID (BID) of every node
2. Every other node on the network relays BPDU messages from other nodes based on the following conditions
   i. *Forward a BPDU if and only if BID < my_BID*
   ii. *Stop sending my own BPDUs if I see BPDUs with BID < my_BID*
3. Eventually all nodes in the network learn the node with the <u>lowest BID</u>. This node is elected as the ROOT NODE (RN) for the spanning tree

<u>Phase B: ROOT PORT selection</u>

1. The RN now is the only node transmitting BPDU messages out of its ports every <u>*hello_time*</u> period. Each message has a counter set to 0
2. Each node that receives a BPDU message, increases the counter by the respective *link cost* of the port the message was received; caches that value; and forwards the message.

   In this way all nodes learn the *path cost* towards the RN from the ports they receive BPDU messages. The port connecting to the RN with the <u>*lowest path-cost*</u> becomes the ROOT PORT (RP) of the node
3. If more than 1 ports are eligible for RP (equal cost paths) the port that belongs to the node with the lowest BID becomes RP

Perlman’s Spanning Tree Protocol – informational only –

<u>Phase C: DESIGNATED PORT selection</u>

1. The nodes on every network segment collectively vote for the node that has the <u>*lowest path*-cost</u> from that network segment to the RN.

   The port connecting the elected node to the network segment becomes the DESIGNATED PORT (DP) for the segment

1. If more than 1 nodes have eligible DP, the port of the node with the lowest BID is preferred

   All other nodes set their ports to BLOCKED PORT (BP) mode, which means listen-only mode (back-up use)

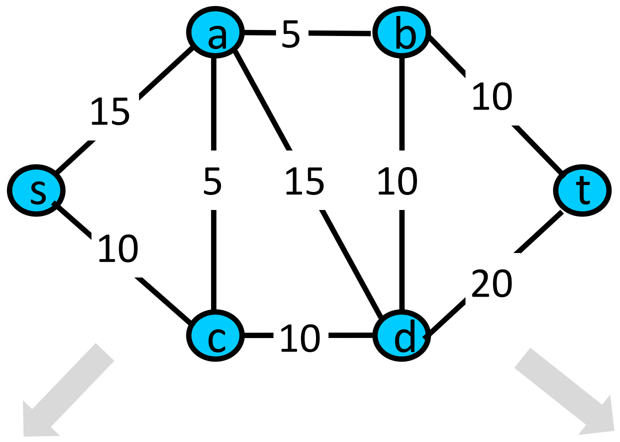
At this point any port is marked either RP, DP, or BP …

<u>Phase D: Topology Maintenance</u>

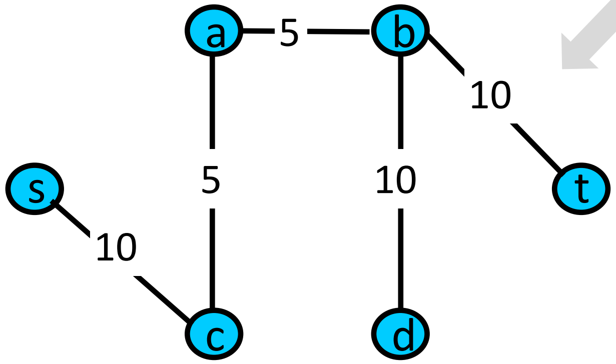
1. RN keeps sending keep-alive BPDU messages to help tracking changes in the topology
2. If a topology change is detected by a node (not receiving keep-alive BPDU messages at its RP), it starts sending TCN messages to its neighbors
3. Every node receiving a TCN acknowledges it and relays it towards the RN
4. In turn the RN will set a *topology_change* flag in its normal BPDU messages to inform all other nodes in the network to update their state

Spanning trees in wireless setting

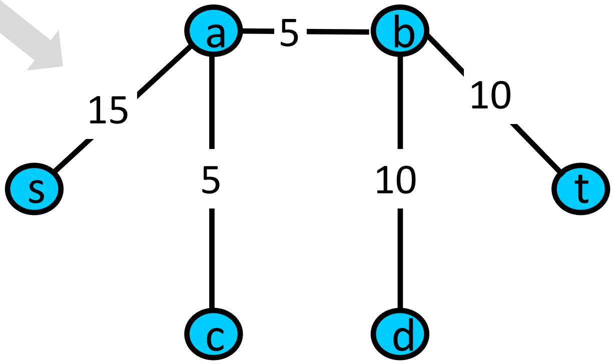
- So… finding a minimum spanning tree seems to solve the problems!
- Best solution ? Let's see, consider a wireless setting
  - Connectedness > throughput
  - Reduce Flooding



<u>Minimum Spanning Tree</u>
Total weight: 30
Diameter: 30
Max hop count: 4
Avg hop count: 2.1
Avg path cost: 15

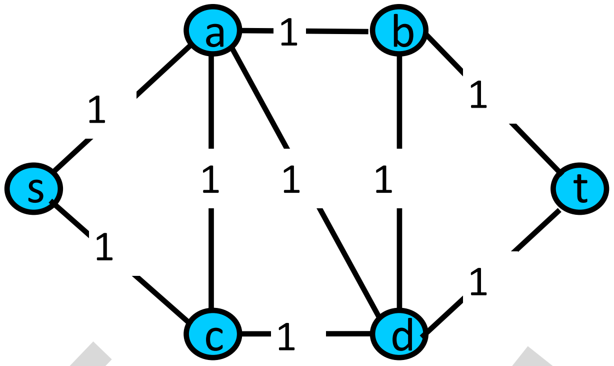


<u>Another Spanning Tree</u>
Total weight: 45
Diameter: 30
Max hop count: 3
Avg hop count: 1.9
Avg path cost: 14.6

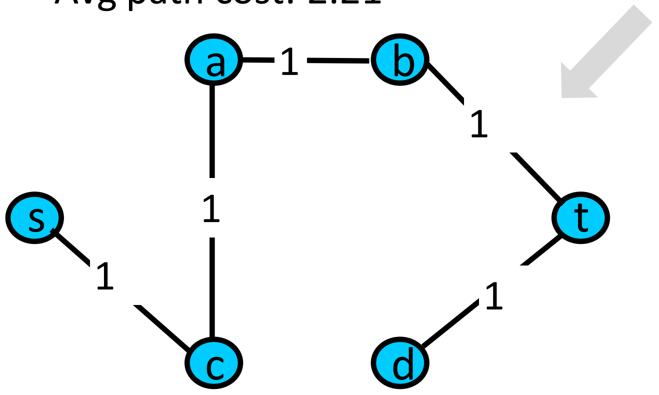


Spanning trees in wireless setting

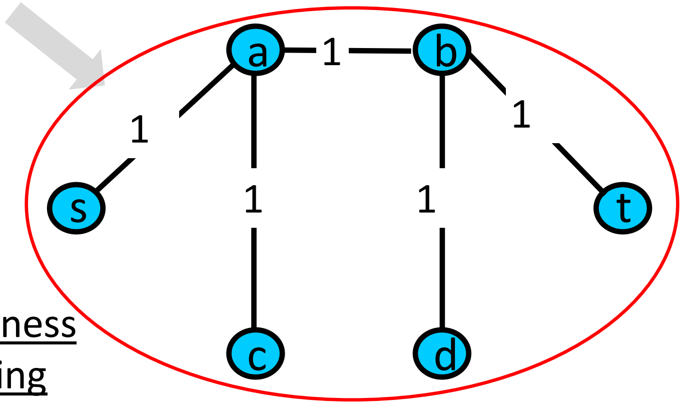
- So… finding a minimum spanning tree seems to solve the problems!
- Best solution ? Let's see, consider a wireless setting
  - Connectedness > throughput
  - Reduce Flooding



<ins>Minimum Spanning Tree</ins>
Total weight: 5
Diameter: 5
Max hop count: 5
Avg hop count: 2.21
Avg path cost: 2.21



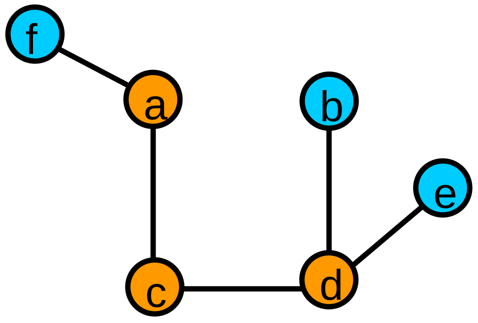
<ins>Minimum Spanning Tree</ins>
Total weight: 5
Diameter: 3
Max hop count: 3
Avg hop count: 1.66
Avg path cost: 1.66

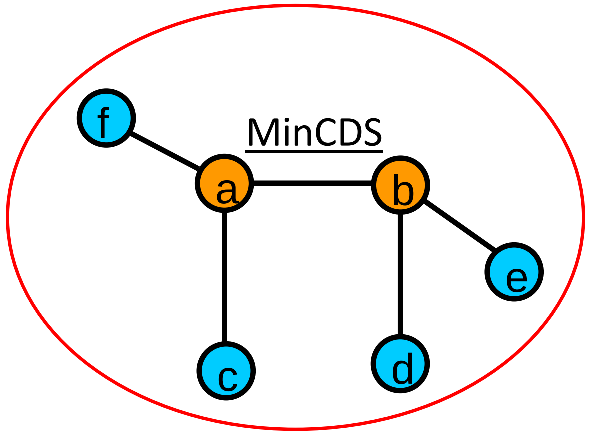


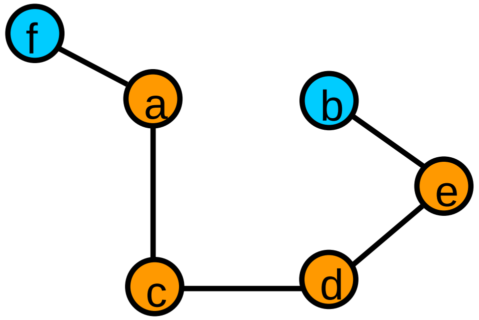
<ins>- Better connectedness</ins>
<ins>- Suited for clustering</ins>

Last visit to Graph theory terminology

- Maximum Leaf Spanning Tree (MaxLST)
  - one that has the largest possible number of leaf nodes (blue nodes)
- Connected Dominating Set (CDS)
  - the nodes of a spanning tree that are not leaf nodes (red nodes)
- Minimum CDS (MinCDS)
  - A CDS with the smallest possible *cardinality*
  - The CDS of a MaxLST (!)
  - Finding one is NP-hard







How to construct a CDS: Multi Point Relay protocol

- Initially every node learn their 2-hop neighborhood, through broadcasting. E.g. for *A*

i. *A* broadcasts: *HELLO_A{}*

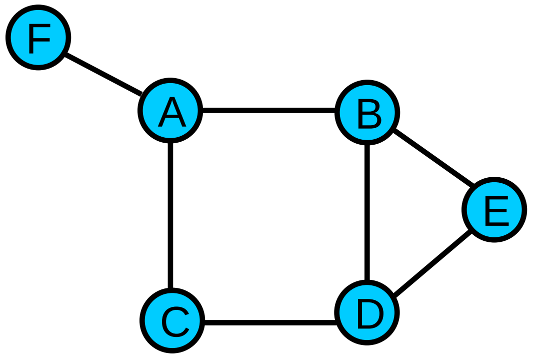
ii. *A* receives: *HELLO_B{A}*, HELLO_F{A}, HELLO_C{A}

iii. *A* broadcasts again: *HELLO_A{B,C,F}*

iv. *F* learns about: *B,C*

B learns about: *F,C*

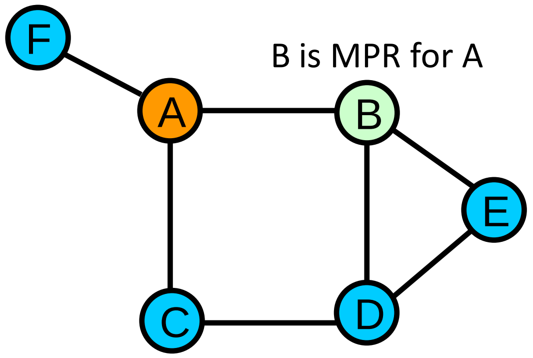
C learns about: *B, F*



- Every node creates MPR list with the minimum set of nodes that allow it to see his 2-hop neighborhood, and advertise it. E.g: for *A* this is *B*

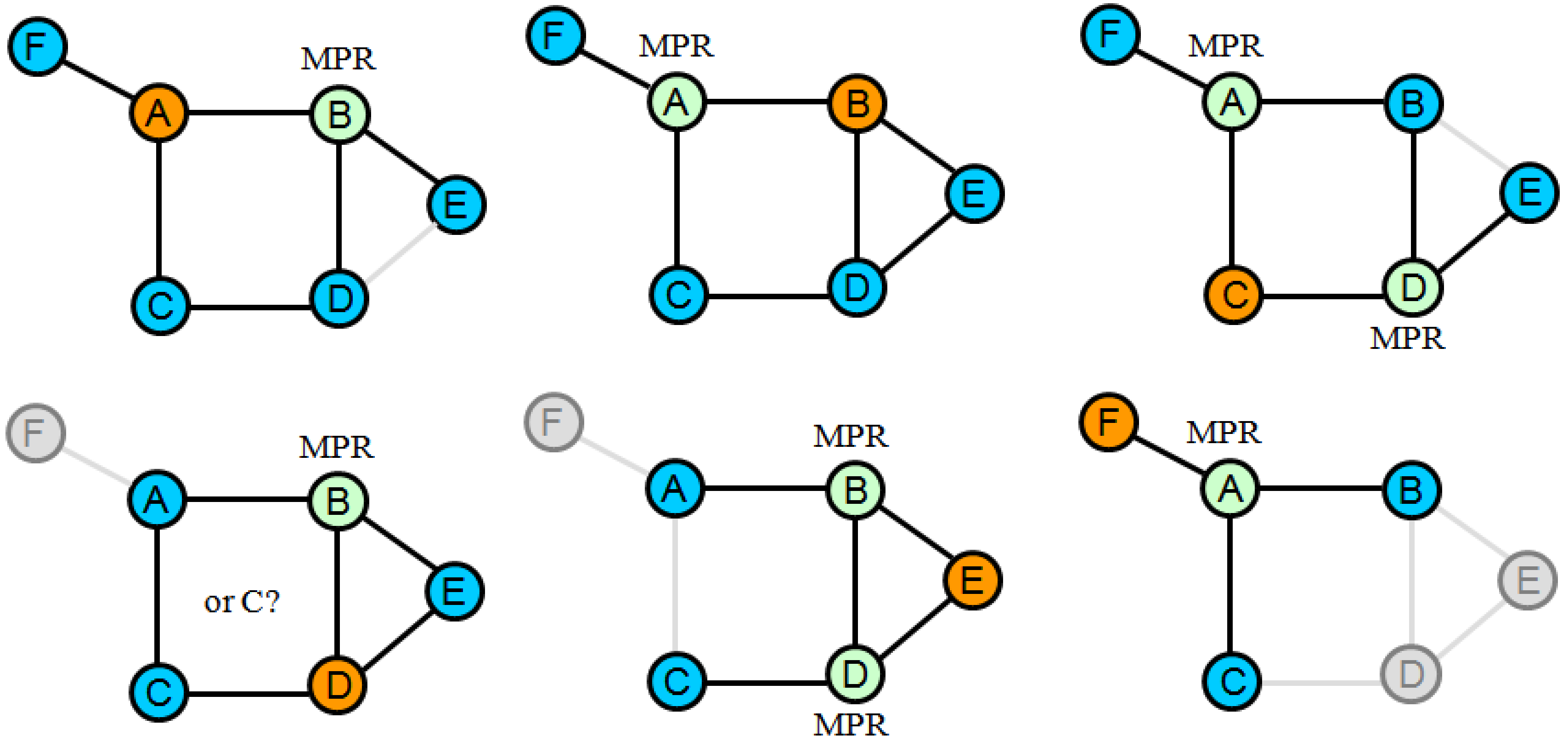
v. *A* broadcasts again: *HELLO_A{B_{MPR},C,F}*

vi. *B* learns and accepts to be an MPR for *A*



How to construct a CDS: Multi Point Relay protocol

- After all nodes build their MPR list we have a CDS {*A,B,D*} but not the MinCDS {*A,B*}



CDS Algorithm in MPR

- Qayvum et al. '02: *O(log n)* complexity, for *n* number of nodes in *G*

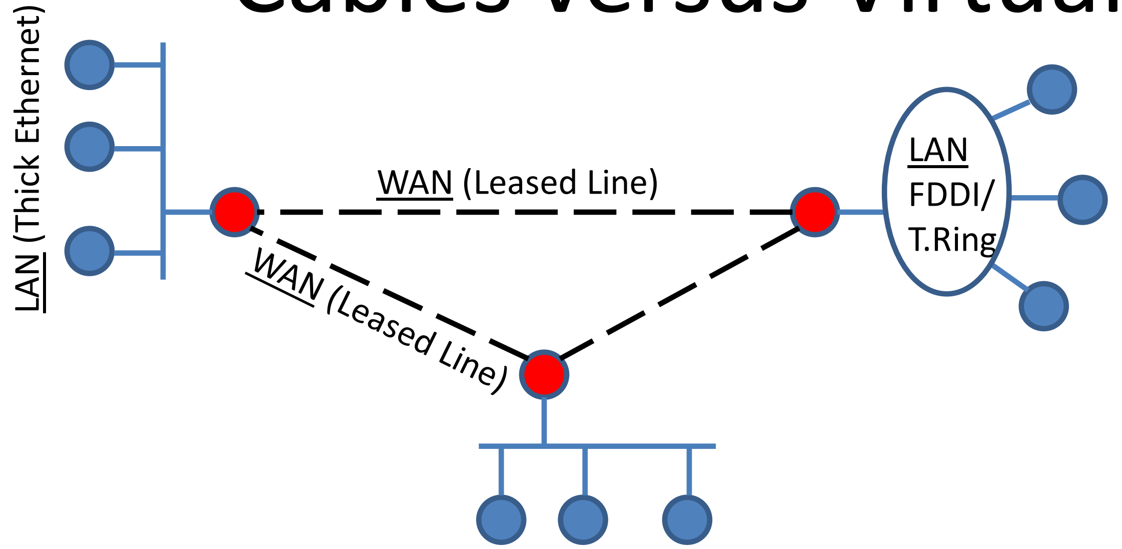
  1. Start with empty multipoint relay set *MPR(x)*
  2. Select and add in MPR(x) those 1-hop neighbour nodes, *which are the only* neighbour of some node in 2-hop neighbourhood
  3. While there still exist some node in 2-hop neighbourhood, not covered in *MPR(x) :*
     i. For each node in 1-hop neighbourhood, *not* in *MPR(x),* compute the *number of* nodes that it covers among the uncovered nodes in the 2-hop neighbourhood
     ii. From them add *in MPR(x) those for* which this number is maximum

- Heuristic for optimizing the MPR set → MinCDS
  - Node accepts to be in MPR list of another node if it, or the inviting node, has the smallest ID in its 1-hop neighborhood

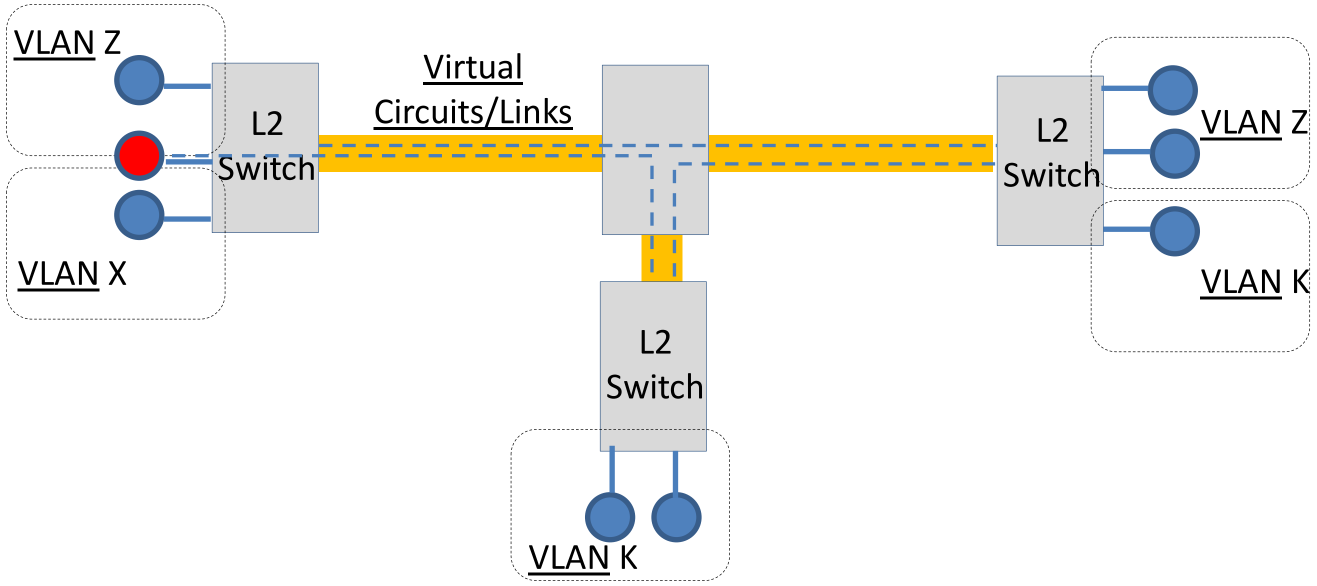
And now something different …

- Now that we've introduced some of the difficult parts (theory), let's look at a couple of basic mechanisms used in creating and maintaining connectivity for a network topology
  - Multi Protocol Label Switching (MPLS)
  - 802.1Q tunneling

Maintaining connectivity
Cables versus Virtual links



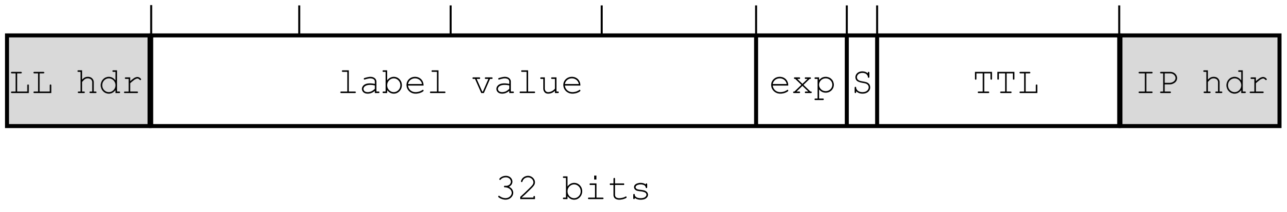
Physical
Interconnectivity
~ 15 years ago



Physical/Virtual
Interconnectivity
Today

Virtual links with L3 Routers Multi Protocol Label Switching

- A label enables the notion of a *Forwarding Equivalence Class* (FEC)
  - A label is a short (4 bytes) locally-significant identifier used to identify a Forwarding Equivalence Class (FEC)

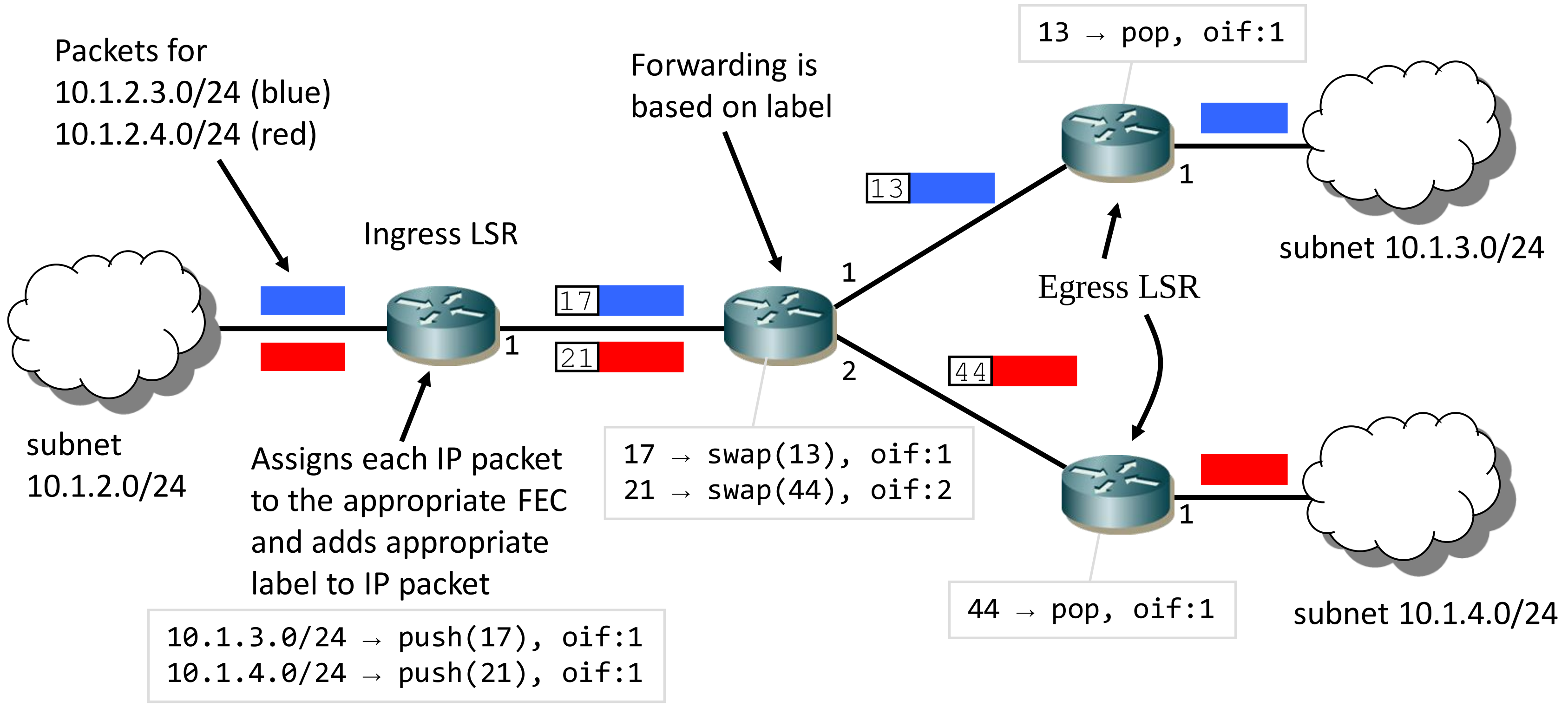


- label: 20 bits, flat identifier
- exp: 3 bits, used as Class of Service (CoS)
- S bit: "bottom of stack“ indicator (when labels are stacked)
- Time To Live: 8 bits

  - A FEC identifies a group of (IP) packets which are handled in the same manner inside an MPLS network
  - Forwarding is based on "exact match“ of label: much faster than IP address “longest-prefix match”

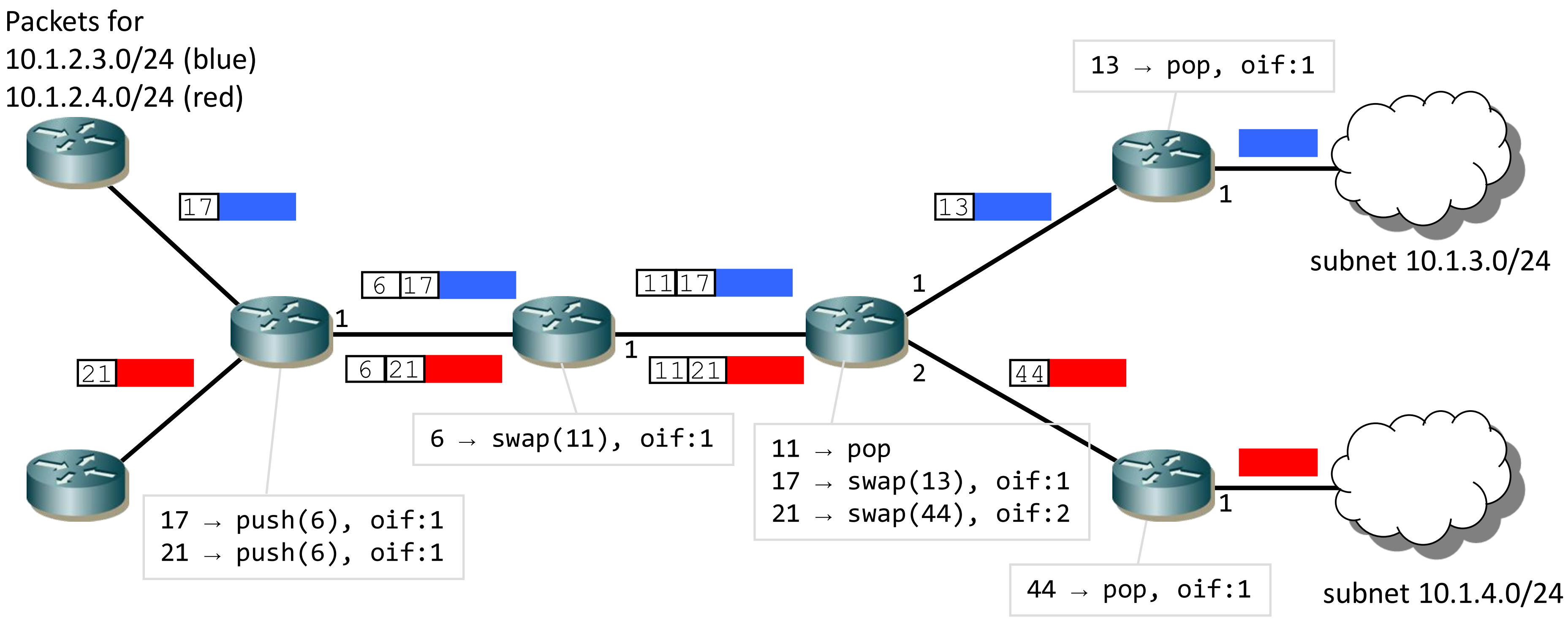
Multi Protocol Label Switching

- MPLS Forwarding
  - Labels identify *Label-Switched Paths* (LSP) in the MPLS network
  - IP flow-to-FEC/LSP mapping at *Label Switched Routers* (LSR) near edges of the MPLS network



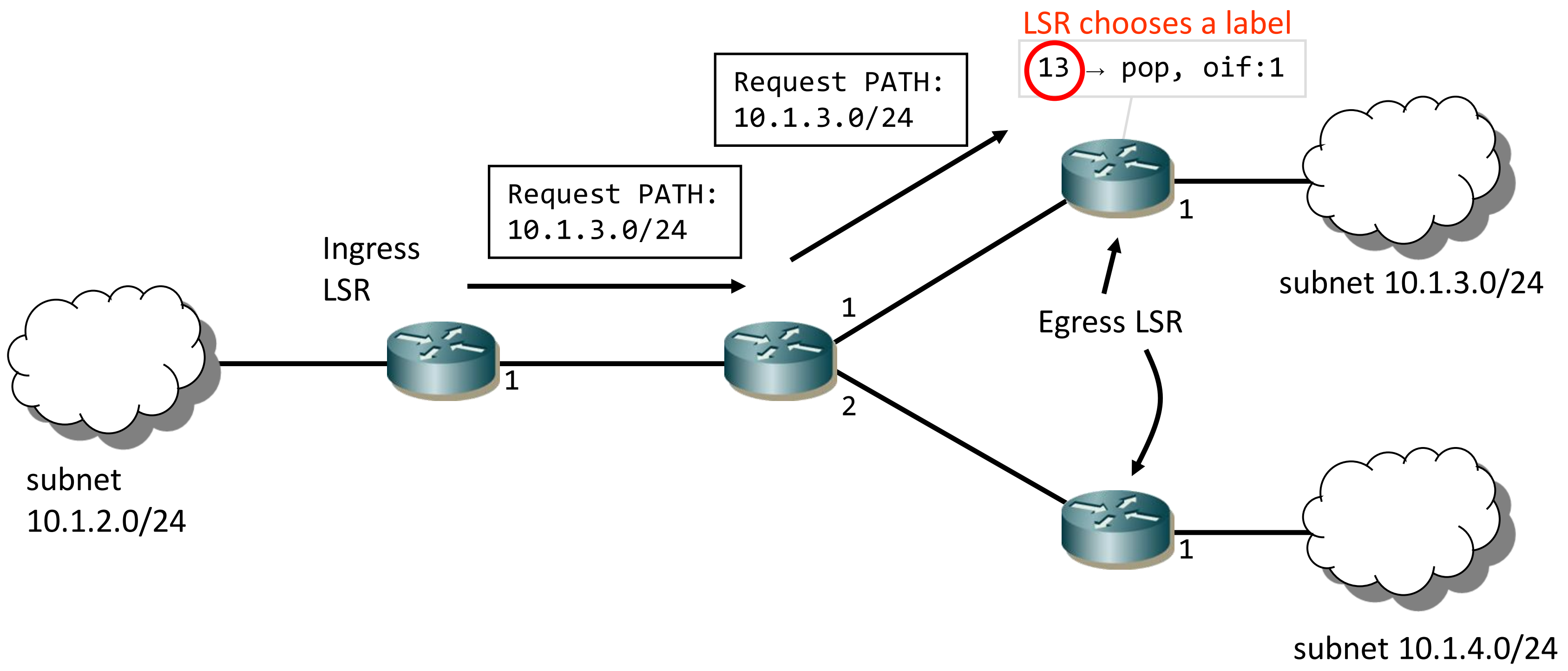
Multi Protocol Label Switching

- Label stacking
  - FECs can be encasulated inside other FECs: we end up with stacks of labels. This is useful to create "trunks" and reduce state in the core MPLS network



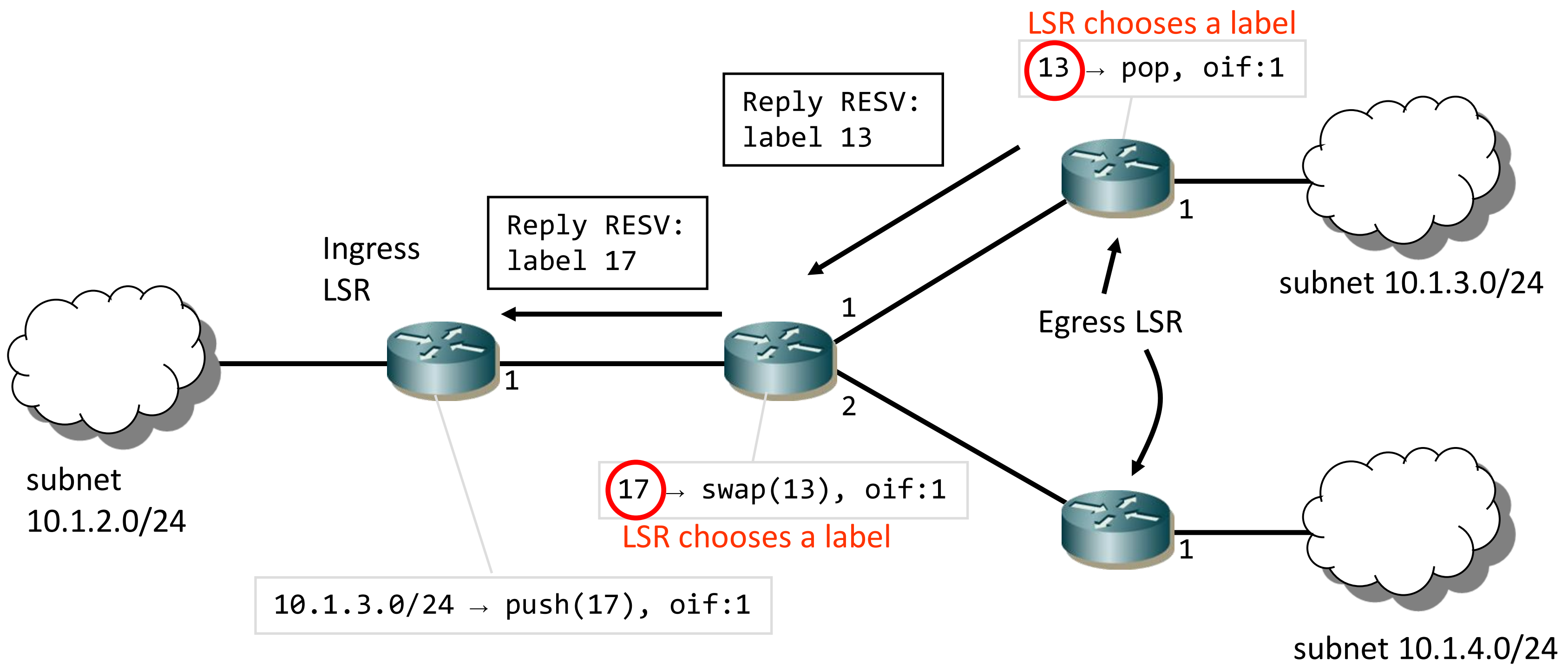
Multi Protocol Label Switching

- Distribution of labels: "against the flow of packets"
  - At each hop, label chosen by downstream LSR and sent to upstream LSR
  - 2 protocols: LDP - Label Distribution Protocol, RSVP-TE - Resource reSerVation Protocol for Traffic Engineering



Multi Protocol Label Switching

- Distribution of labels: "against the flow of packets"
  - At each hop, label chosen by downstream LSR and sent to upstream LSR
  - 2 protocols: LDP - Label Distribution Protocol, RSVP-TE - Resource reSerVation Protocol for Traffic Engineering



The same idea applied in L2 switches

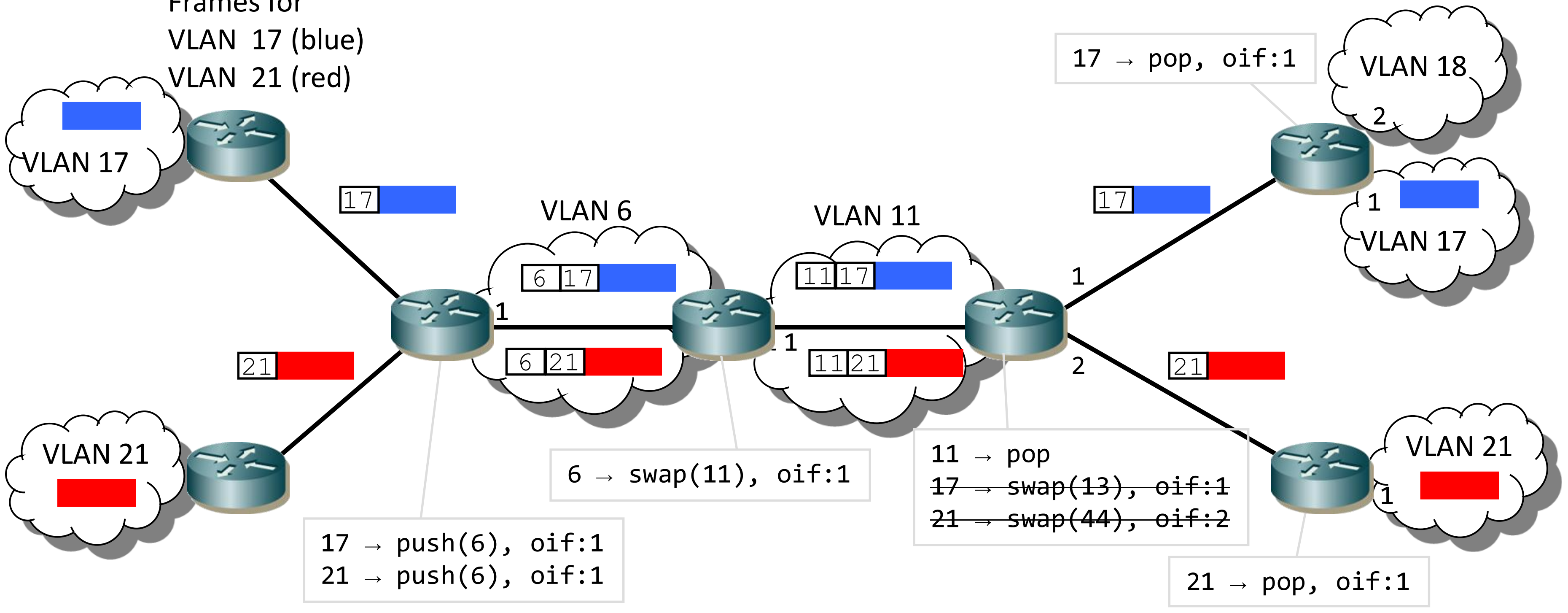
- What if we are to do exactly the same thing but instead of labeling IP packets, we would tag(!) ethernet frames …
  - Packet and frame are the same thing (datagrams), just different headers…
  - Label and tag are the same thing too, no ?
  - Of course we would need to replace routers with ethernet switches (however they would both do “*exact-match*” based classification on labels ..or tags!)

Let’s see …

Metro/Carrier Ethernet

- IEEE 802.1Q tunneling, or “tag stacking”, or “QinQ”!
  - Clouds everywhere, called VLANs …. that define the scoping of tags
  - Same VLANs usually at both ends, albeit they don’t have to (VLAN translation)

Frames for
VLAN 17 (blue)
VLAN 21 (red)



17 → push(6), oif:1
21 → push(6), oif:1

6 → swap(11), oif:1

11 → pop
~~17 → swap(13), oif:1~~
~~21 → swap(44), oif:2~~

17 → pop, oif:1

21 → pop, oif:1

A bit of (simplified) history

- When first moved from leased lines to VCs, it was Telcos (IEEE) offering X.25, then Frame Relay, then ATM, as the new network solution that would eventually obsolete IP.
- The way to sabotage IP was nothing less than splitting 1 foot across *2.sthg* shoes (and you had to cut the toes!)
  - Force larger variable size IP packets in smaller fixed size X.25/F.R./ATM cells
- The IETF had enough in the end, and so they developed their own solution. Instead of trimming IP packets at the size of cells they used part of the VC idea (only the headers) as FECs, leaving the size open.
  - Better throughput, less processing, easier buffer mgmt
- So in the end IEEE learned the lesson and went off to re-apply it at the edge switches, this time on ethernet frames, which in a way has been their territory traditionally (Link layer).

- By the way … do you see the similarity to LISP (last lecture on association)?

Questions ?