Network layer: Overview

- Network layer:
  - Functionality?
  - Devices on this layer? How to address them?
  - Protocol of this layer?
- Routing and forwarding (difference?): Routers!
- The IP protocol („glue of the Internet“: be careful with definition..., e.g., BGP)
Network Layer Functions

- Goal: transport packet from sending to receiving hosts
  - host-host semantic (what about transport layer?)
- Network layer protocols in every host, router

Three important functions:

- **Addressing**
- **Path determination**: route taken by packets from source to dest. *Routing algorithms*
- **What is forwarding?**
- **Switching/forwarding**: move packets from router’s input to appropriate router output
Routing vs Forwarding

Routing Information Base:
Lists routes (sometimes with costs) to particular network destinations (via next hop), and is updated by routing protocol.

<table>
<thead>
<tr>
<th>Network Destination</th>
<th>Netmask</th>
<th>Gateway</th>
<th>Interface</th>
<th>Metric</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0.0.0</td>
<td>0.0.0.0</td>
<td>192.168.0.1</td>
<td>192.168.0.100</td>
<td>10</td>
</tr>
<tr>
<td>127.0.0.0</td>
<td>255.0.0.0</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>1</td>
</tr>
<tr>
<td>192.168.0.0</td>
<td>255.255.255.0</td>
<td>192.168.0.100</td>
<td>192.168.0.100</td>
<td>10</td>
</tr>
<tr>
<td>192.168.0.100</td>
<td>255.255.255.255</td>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>10</td>
</tr>
<tr>
<td>192.168.0.255</td>
<td>255.255.255.255</td>
<td>192.168.0.100</td>
<td>192.168.0.100</td>
<td>10</td>
</tr>
</tbody>
</table>

Forwarding Information Base:
Smaller, optimized for fast lookup of destination addresses (find forwarding port), contains only „chosen routes“.
Graph abstraction for routing

Graph (known?): \( G = (N,E) \)

Set of nodes? Switch?
... no, only entities with IP addresses, like...?
\( N = \) set of routers = \{ \( u, v, w, x, y, z \) \}
\( E = \) set of links =\{ \( (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \) \}

Path: Sequence of edges (routers)

Remark: Graph abstr. is useful in other network contexts
Example: P2P (“overlay network”), where \( N \) is set of peers and \( E \) is set of TCP connections (logical links over multiple IP hops), railway networks, telephone networks, etc.
Graph abstraction: Costs

- $c(x,x') = \text{cost of link } (x,x')$
  - e.g., $c(w,z) = 5$
- What does cost denote?
- Cost can be always 1, or inversely related to bandwidth, or related to congestion, transport / peering cost, ...

Cost of path?
Cost $c(x_1,x_2,x_3,\ldots,x_p) = c(x_1,x_2) + c(x_2,x_3) + \ldots + c(x_{p-1},x_p)$

**Question:** What’s the least-cost path between $u$ and $z$?

**Routing algorithm:** Alg. that finds “good” path (typically: least cost path)
Routing algorithm classification

Global or decentralized information?

Global?
- All routers have complete topology, link cost info
- "Link state" algorithms
- Example?
  - Dijkstra (no negative links!), ...

Decentralized?
- Router knows physically-connected neighbors, link costs to neighbors
- Iterative process of computation, exchange of info with neighbors
- "Distance vector" algorithms
- Example?
  - Bellman-Ford (no negative cycles!), ...

Static or dynamic?

Static:
- Routes change slowly over time
Dynamic:
- Routes change more quickly
  - periodic update
  - proactively in response to link cost changes
A link-state routing algorithm

Dijkstra’s algorithm

- Net topology, link costs known to all nodes
  - Accomplished via “link state broadcast”
  - All nodes have same info
- Computes least cost paths from one node (‘source’) to all other nodes
  - Gives routing table for that node
- Iterative: after $k$ iterations, know least cost path to $k$ dest.’s

Notation:

- $c(i,j)$: Link cost from node $i$ to $j$. Cost infinite if not direct neighbors
- $D(v)$: Current value of cost of path from source to dest. $v$
- $p(v)$: Predecessor (why? routing!) node along path from source to $v$
- $N'$: Set of nodes whose least cost path definitively known
**Dijsktra’s algorithm**

1. **Initialization for A:**
   - $N' = \{A\}$
   - for all nodes $v$
     - if $v$ adjacent to $A$
       - then $D(v) = c(A,v)$
     - else $D(v) = \infty$

2. **Loop**
   - find $w$ not in $N'$ such that $D(w)$ is a minimum
   - add $w$ to $N'$
   - update $D(v)$ for all $v$ adjacent to $w$ and not in $N'$:
     - $D(v) = \min( D(v), D(w) + c(w,v) )$
   - /* new cost to $v$ is either old cost to $v$ or known shortest path cost to $w$ plus cost from $w$ to $v$ */

3. **Until all nodes in $N'$**

---

- best route cost so far...
- overall minimum: definitely known! (why? assumption?)
- update distances
**Visualization:**

Question: Does this work for **negative link costs** also? Without negative cycles? No... (e.g., triangle with costs -2,2,3)

In each step, minimal cost node \( v \) in **border** is added to \( N' \): definitely no shorter path! And then \( v \)'s neighbors' costs are updated (e.g., added to border).

Question: Does this work to find path with **highest** costs also? No...
Dijkstra’s algorithm: Example

<table>
<thead>
<tr>
<th>Step</th>
<th>start N’</th>
<th>D(B),p(B)</th>
<th>D(C),p(C)</th>
<th>D(D),p(D)</th>
<th>D(E),p(E)</th>
<th>D(F),p(F)</th>
</tr>
</thead>
<tbody>
<tr>
<td>→ 0</td>
<td>A</td>
<td>2,A</td>
<td>5,A</td>
<td>min! 1,A</td>
<td>infinity</td>
<td>infinity</td>
</tr>
<tr>
<td>→ 1</td>
<td>AD</td>
<td>2,A</td>
<td>4,D update!</td>
<td>2,D</td>
<td>infinity</td>
<td></td>
</tr>
<tr>
<td>→ 2</td>
<td>ADE</td>
<td>2,A</td>
<td>3,E</td>
<td></td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>→ 3</td>
<td>ADEB</td>
<td></td>
<td>3,E</td>
<td></td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>→ 4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td>4,E</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Note: Dijkstra computes more than needed: shortest paths to all closer nodes!

Routes: via opt predecessors!

Result: predecessor structure = shortest path spanning tree!
(Forwarding = in which subtree is destination!)
Dijkstra’s algorithm: Example (2)

Resulting shortest-path tree from A:

Resulting forwarding table from routing algo in A:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>(A,B)</td>
</tr>
<tr>
<td>D</td>
<td>(A,D)</td>
</tr>
<tr>
<td>E</td>
<td>(A,D)</td>
</tr>
<tr>
<td>C</td>
<td>(A,D)</td>
</tr>
<tr>
<td>F</td>
<td>(A,D)</td>
</tr>
</tbody>
</table>

Time complexity of algorithm?
Dijkstra’s algorithm: Discussion

Algorithm complexity: \( n \) nodes

- Each iteration: need to check all nodes, \( w \), not in \( N \)
  - find minimum, decrease key via new node, ...
- Depending on data structure and \# edges: \( O(n^2) \), \( O(n \log n) \), ...

Imagine: link cost = amount of carried traffic. Problem?

Oscillations possible (weight = traffic load, asymmetric):

Assume loads to A:
\begin{align*}
(D,A) &= 1 \\
(B,A) &= 1 \\
(C,A) &= e
\end{align*}

\[ \begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C \\
\downarrow \\
B \\
\downarrow \\
1 \\
\end{array} \quad \begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C \\
\downarrow \\
B \\
\downarrow \\
1+e \\
\end{array} \quad \begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C \\
\downarrow \\
B \\
\downarrow \\
0 \\
\end{array} \quad \begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C \\
\downarrow \\
B \\
\downarrow \\
0 \\
\end{array} \quad \begin{array}{c}
A \\
\downarrow \\
D \\
\downarrow \\
C \\
\downarrow \\
B \\
\downarrow \\
0 \\
\end{array} \]

Initially

... recomputing routing

Counter-clockwise!
Dijkstra’s algorithm: Discussion

Solution for oscillation problem?
- Routers should not change at the same time...
- ... but one after the other.
- There are results showing that routers can synchronize in network... (see later!)

![Network Diagram]

Initially:

- A → D: 1
- A → B: e
- A → C: 1+e
- B → A: 0
- B → D: 0
- B → C: e
- C → A: 0
- C → B: 0
- C → D: 1
- D → A: 0
- D → B: 0
- D → C: 1

... recompute routing:

- A → D: 2+e
- A → B: 0
- A → C: 0
- B → A: 0
- B → D: 1+e
- B → C: 0
- C → A: 1+e
- C → B: 0
- C → D: 1+e
- D → A: 0
- D → B: 0
- D → C: 1+e

... recompute:

- A → D: 2+e
- A → B: 0
- A → C: 0
- B → A: 0
- B → D: 1+e
- B → C: 0
- C → A: 1+e
- C → B: 0
- C → D: 1+e
- D → A: 0
- D → B: 0
- D → C: 1+e

... recompute:
**Distance vector algorithm**

**Bellman-Ford Equation (dynamic programming)**

Define

\[ d_x(y) := \text{cost of least-cost path from } x \text{ to } y \]

Then

\[ d_x(y) = \min \{ c_v(x, v) + d_v(y) \} \]

where \( \min \) is taken over all neighbors \( v \) of \( x \)
Bellman-Ford: Example

Clearly, for neighbors of u, distance to z?
\[ d_v(z) = 5, \quad d_x(z) = 3, \quad d_w(z) = 3 \]

Bellman-Ford equation says:
\[ d_u(z) = \min \{ c(u,v) + d_v(z), \]
\[ c(u,x) + d_x(z), \]
\[ c(u,w) + d_w(z) \} \]
\[ = \min \{ 2 + 5, \]
\[ 1 + 3, \]
\[ 5 + 3 \} = 4 \]

Node that yields minimum (here: x) is next hop in shortest path => forwarding table

How to implement these principles in a distributed manner?
Bellman-Ford: Visualization

Unlike Dijkstra which greedily selects min next node, we greedily relax all edges... (go n times over all m edges: O(nm))

Induction (over length):
Opt path of length 1 => easy
Opt path of length k => k+1
Distance vector algorithm (2)

- $D_x(y) = \text{estimate of least cost } d_x(y) \text{ from } x \text{ to } y$
- Distance vector defined as (distance from $x$ to each node in network $N$): $D_x = [D_x(y) : y \in N ]$
- Of course, node $x$ knows cost to each neighbor $v$: $c(x,v)$
- Idea: node $x$ also maintains its neighbors’ distance vectors (distributed implementation of Bell.-Ford!)
  - For each neighbor $v$, $x$ maintains $D_v = [D_v(y) : y \in N ]$
  - How to combine with $c(x,v)$ to compute $D_x$?
Distance vector algorithm (3)

Basic idea:

- Each node **periodically** sends its own distance vector estimate to neighbors
- When a node $x$ receives new DV estimate from neighbor $v$, it updates its own DV using B-F equation (path over new best neighbor?):

$$D_x(y) \leftarrow \min_v \{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$

- Under “natural” conditions the estimates of $D_x(y)$ converge to the actual least cost $d_x(y)$
Distance vector algorithm (4)

Iterative, asynchronous:

- Each local iteration caused by:
  - Local link cost change
  - DV update message from neighbor

Distributed:

- Each node notifies neighbors only when its Distance Vector changes
  - Neighbors then notify their neighbors if necessary

Each node:

- wait for (change in local link cost of msg from neighbor)
- recompute estimates
- if Distance Vector to any dest has changed, notify neighbors
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \]
\[ = \min\{2+0, 7+1\} = 2 \]

\[ D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \]
\[ = \min\{2+1, 7+0\} = 3 \]
Summary: DV Algo

At each node, $x$:

1. Initialization:
2. for all destinations $y$ in network $N$:
   3. $D_x(y) = \infty$ if $y$ is not a neighbor
   4. $D_x(y) = c(x,y)$ if $y$ is a neighbor
5. for each neighbor $w$
6. send distance vector $D_x = [D_x(y): y \text{ in } N]$ to $w$
Summary DV Algo (2)

9  loop
10  wait (until I see a link cost change to neighbor \(w\)
11          or until I receive update from neighbor \(w\))
12
13  for each \(y\) in \(N\):
14      \(D_x(y) = \min_v \{ c(x, v) + D_v(y) \} \)
15
16  if \(D_x(y)\) changed for any destination \(y\)
17      send DV \(D_x = [D_x(y)]: y \text{ in } N\) to all neighbors
18
19  forever

Does it work for negative weights?
And negative cycles?
    Yes. No...
Distance vector (DV): Link cost changes

So what if link cost changes?
- Node detects local link cost change
- Updates routing info, recalculate distance vector
- If DV changes, notify neighbors

“Good news travels fast (three steps)”

At time $t_0$, $y$ detects the link-cost change, updates its DV (“cost 1 to $x$”), and informs its neighbors.

At time $t_1$, $z$ receives the update from $y$ and updates its table (“cost 2 to $x$”). It computes a new least cost to $x$ and sends its DV to neighbors.

At time $t_2$, $y$ receives $z$’s update (“cost 2 to $x$”) and updates its distance table. $y$’s least costs do not change and hence $y$ does not send any message to $z$. 

Stefan Schmid - 23
y learns that z reaches x in two, does not make any difference anymore => **termination** after three steps!

### node y table

<table>
<thead>
<tr>
<th>from</th>
<th>cost to</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

### node z table

<table>
<thead>
<tr>
<th>from</th>
<th>cost to</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>y</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>y</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Distance vector: Link cost changes (2.)

Link cost changes:
- Good news travels fast
- Bad news travels slow

Diagram:
- Node X
- Node Y
- Node Z
- Link cost 4 between X and Y
- Link cost 1 between Y and Z
- Link cost 50 between X and Z
- Bad news!!
Why?

z still stores old value, so:

\[ D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60 + 0 , 1 + 5\} = 6 \]

\[ D_y(x) = \min\{c(y,x) + D_x(x), c(y,z) + D_z(x)\} = \min\{60 + 0 , 1 + 7\} = 8 \]

Increases two by two, always via z node, after which z increases too, does not realize is same path…
Distance vector: Link cost changes (3.)

Link cost changes:
- Good news travels fast
- Bad news travels slow - “count to infinity” problem!
  - 44 iterations before algorithm stabilizes: see text
- What happens here? Real infinity...?

Solution? Poisoned reverse?
- Idea: prevent mutual „better routes“!
- If Z’s shortest path to X is via Y:
  - Z claims to Y that its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
  - As long as Z reaches X cheaper like this, we’re fine?
- Will this completely solve count to infinity problem?
  - Longer cycles (not just between neighbors)?
Poisoned reverse fails?

1. 

2. 

3. 

4. 

Solution?
E.g., Router 1 should hold down this destination for a certain while which gives other routers time to learn about failure before depending on wrong paths. During hold down time, Router 1 does not accept any updates to the net prefix (later good news Fast). (But how long?
RIP updates every 30sec, can get lost.... Announce negative reachability?)
Comparison of LS and DV algorithms

Message complexity

- 
- 

Speed of Convergence

- 
- 

Robustness: What happens if router malfunctions?

LS:

- 

DV:

- 

Comparison of LS and DV algorithms

Message complexity
- LS: each node needs to know whole graph => \(N \times |E|\) 😞
- LS problem: how to learn topology, afterwards „we’re fine“
- DV: only between neighbors, only if shortest routes change 😊

Speed of Convergence
- LS: exchange graphs... 😊
- DV: slow for bad news, count to infinity 😞

Robustness: What happens if router malfunctions?
**LS:**
- In LS, typically all nodes have same view on topology, this avoids certain DV problems!
- Router could send wrong information about connections to its neighbors, delete other packets, etc.
- quite robust: nodes make their computations themselves 😊

**DV:**
- Problems “less visible”, e.g., router could announce super routes to everyone! Happened to a small ISP => got too much traffic, Internet down in 1997! 😞
- Wrong info can propagate in whole network... 😞
Internet routing

So far – idealization
- All routers identical
- Network “flat”
... not true in practice

Administrative autonomy
- Internet = network of (autonomous) networks
- Each network admin may want to control routing in its own network
- Aggregate routers into regions, “autonomous systems” (AS)
- One ISP can organize its network in multiple ASs
- Routers in same AS run same routing protocol
  - “Intra-AS” routing protocol
  - Routers in different AS can run different intra-AS routing protocol

Scale: With 200 million destinations:
- Can’t store all dest’s in routing tables!
- Routing table exchange would swamp links!
Interconnected ASes

- Forwarding table is configured by both intra- and inter-AS routing algorithm
  - Intra-AS sets entries for **internal dests**
  - Inter-AS & Intra-AS sets entries for **external dests** (via **AS border routers**!)

Diagram:
- AS1
  - 1a, 1b, 1c, 1d
- AS2
  - 2a, 2b, 2c
- AS3
  - 3a, 3b, 3c

Intra-AS Routing algorithm
Inter-AS Routing algorithm
Forwarding table
Interconnected ASes

- Intra-AS Routing algorithm
- Inter-AS Routing algorithm
- Forwarding table

What if only one gateway router in AS...?

- Intra-routers simply forward to it (scales!)
- Otherwise flood (how?) inside AS which prefixes available via which gateway, job of Inter-AS protocol (full or reflector)
- Intra-AS protocol tells us how to get to this gateway
Interplay between routing and forwarding

Routing defines forwarding table: which prefixes go to which port?

<table>
<thead>
<tr>
<th>header value</th>
<th>output link</th>
</tr>
</thead>
<tbody>
<tr>
<td>0100</td>
<td>3</td>
</tr>
<tr>
<td>0101</td>
<td>2</td>
</tr>
<tr>
<td>0111</td>
<td>2</td>
</tr>
<tr>
<td>1001</td>
<td>1</td>
</tr>
</tbody>
</table>

value in arriving packet’s header
Why different Intra- and Inter-AS routing?

Policy:
- Inter-AS ("the glue of the Internet", e.g., BGP): Admin wants control over how its traffic routed, who routes through its net. ("policies")
- Intra-AS: Single admin, so no policy decisions needed
  - Cost metric?
  - E.g., link utilization (not delay, would be instable)

Scale:
- Hierarchical routing saves table size, reduced update traffic

Performance:
- Intra-AS: Can focus on performance
- Inter-AS: Policy may dominate over performance

We need BOTH!
Inter-AS Tasks

- Suppose router in AS1 receives datagram for dest outside of AS1
  - Router should forward packet towards an **AS-border router**, but which one?

AS1 needs ...

1. ... to learn which dests are reachable through AS2 and which through AS3
2. ... to propagate this reachability info to all routers in AS1

Job of inter-AS routing!
Intra-AS routing

- Also known as Interior Gateway Protocols (IGP)
- Most common Intra-AS routing protocols:
  - **RIP**: Routing Information Protocol
    - Distance vector protocol (based on Bellman-Ford)
    - Routers *periodically* exchange reachability info with their neighbors
    - Distance metric?
    - Hop count...
    - Advantage: Simple, minimal communication overhead
    - Disadvantage: Long convergence times, loop detection (typical for B-F)
Intra-AS routing protocols

- **OSPF**: Open Shortest Path First
  - Link state protocol (based on Dijkstra)
  - Routers periodically flood immediate reachability information to all other routers
  - Distance metric: administrative weight to reach prefixes / subnets (weight should be stable but can depend on load also; e.g., all 1 would mean...?)
  - (... hop count...)
  - Advantage: fast convergence
  - Disadvantage: complexity and communication overhead

- **ISIS**: Intermediate-System-to-Intermediate-System (ISO 10589) (link state)

- **IGRP**: Interior Gateway Routing Protocol (Cisco proprietary) (distance vector)

- **EIGRP**: Enhanced Interior Gateway Routing Protocol (Cisco proprietary) (enhanced distance vector)
OSPF (Open Shortest Path First)

- “Open”: Specification publicly available
- Uses the **link state algorithm**, so in terms of implementation as expected:
  - State: per router info about itself and attached networks (link weights...)
  - OSPF advertisements: propagates state
  - Link state **database**: state of all routers
  - Topology **map**: derived from link state database
OSPFv2: Components

- Bootstrap: who is my neighbor?
  - Hello Protocol

- How do I distribute info?
  - Advertisements disseminated to entire Autonomous System (via reliable flooding, to all paths)
  - OSPF messages directly over IP (rather than TCP or UDP)

- Route computation
  - From link state database with Dijkstra’s algorithm
  - Supports equal-cost path routing (load-balancing over equal cost paths)
OSPF “advanced” features

- **Security**: All OSPF messages are authenticated (to prevent malicious intrusion)
- **Multiple same-cost paths** allowed (e.g., per flow basis load balancing)
- For each link, **multiple cost metrics** for different TOS (aka “external cost metrics”, eg, reliability; satellite link cost set “low” for best effort, high for real time)
- **Integrated uni- and multicast** support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- **Hierarchical** OSPF in large domains
Hierarchical OSPF: Structure inside AS!

- ABRs summarize distances and announce to other ABRs.
- Routers have full knowledge of area only.

To other AS...
Hierarchical OSPF (2.)

- **Two-level hierarchy**: Local area and backbone.
  - Link-state advertisements *only in respective areas*.
  - Nodes in each area have detailed area topology; only know direction (shortest path) to networks in other areas.

- **Area Border routers** “summarize” distances to networks in the area and advertise them to other Area Border routers.

- **Backbone routers**: Run an OSPF routing algorithm **limited to the backbone**.

- **Boundary routers**: Connect to other ASs.
Example: Inter-Provider Routing?

- Where would Provider X send data (from A to B)?

- Both routers of Y advertise B...
- ... provider X will most likely send it to lower node, just to get rid of data asap ("hot potato routing"), should not stay in own network (costly!)
Internet Inter-AS routing: BGP

- The de facto standard: Border Gateway Protocol (BGP)
- BGP provides each AS a means to:
  1. Obtain subnet reachability (aggregated and based on prefixes!) information from neighboring ASs
  2. Propagate reachability information to all routers in the AS
  3. Determine “good” routes to subnets based on...?
  4. ... reachability information and routing policy.
- Allows a subnet to advertise its existence to rest of the Internet: "I am here"
- Issues:
  - Which routing algorithm?
  - How are routes advertised?
  - How to implement routing policies?
  - Convergence time? (can be dozens of seconds)
BGP Basics

- Pairs of routers (BGP peers) exchange routing info over semi-permanent TCP connections: **BGP sessions**
  - eBGP for inter-AS links, iBGP for intra-AS links (full mesh / reflector)
- Note that BGP sessions do not correspond to physical links.
- Are BGP routes symmetric?
- No! Opposite direction may have different path!
- When AS2 advertises a prefix to AS1, AS2 is *promising* it will forward any datagrams destined to that prefix towards the prefix.
  - AS2 can aggregate prefixes in its advertisement

![BGP Diagram](image)
BGP is a **path vector protocol**

- **Distance vector algorithm with extra information**
  - Two important attributes:
    - **AS-PATH**: contains all ASs along the way, updated via eBGP: AS 67 AS 17 (but not all link states!) => why needed?
    - ... for policies...
    - **NEXT-HOP**: Indicates the specific internal-AS router to next-hop AS (router interface where AS-PATH starts, there can be many for same AS-PATH: NEXT-HOP of remote router interface is submitted to Intra-AS module to find route to internal gateway!)
  - Path can be used to make routing decisions, e.g., to avoid loops
  - Pure distance vector does not enable policies
  - Link state does not scale and exposes policies

- **When advertising a prefix, advert includes BGP attributes**
  - Prefix + other attributes = “route”

- **When gateway router receives route advertisement, uses ingress filters to accept/decline**
  - Can make decision based on ASes on path, e.g., to avoid loops
NEXT-HOP

AS X may hear two routes to AS Y (via two peering connections), have same AS-PATH (same sequence of ASs) but different NEXT-HOP. iBGP can determine cost of both routes with NEXT-HOP, and make, e.g., hot-potato routing!
BGP messages

Peers exchange BGP messages using TCP

- **OPEN:**
  - Opens TCP conn. to peer
  - Authenticates sender

- **UPDATE:**
  - Advertises new routes (or withdraws old)

- **KEEPALIVE:**
  - Keeps conn alive in absence of UPDATES, ACKs OPEN request

- **NOTIFICATION:**
  - Reports errors in previous msg; closes a connection

Process:

- Initialization: open $\Rightarrow$ updates **for all routes**
- Ongoing: updates for changed routes
BGP route processing

Receive BGP Updates
Apply Policy = filter routes (e.g., loops) & tweak attributes

Based on Attribute Values

Best and Alternate Routes

Apply policies to Best Routes!

Transmit BGP Updates

Apply Export Policies (egress filter)

Install Best Routes

IP Forwarding Table

Note:
- BGP never accepts a route at AS X that contains X in AS_PATH.
- AS only announces its best route to given destination, so router hears at most N best routes for any destination, where N is number of neighbors.
Routing policy

- Reflects goals of network provider
  - Which routes to *accept* from other ASes
  - How to manipulate the accepted routes
  - *How to propagate* routes through network
  - How to manipulate routes before they leave the AS
  - Which routes to send to another AS
Routing policy: Examples

- **Honor business relationships**
  (E.g., customers get full-table; peers only customer prefixes)
  (E.g., prefer customer routes over peer routes over upstream routes)

- **Allow customers a choice of route**
  (E.g., on customer request do not export prefix to AS x, etc.)

- **Enable customer traffic engineering**
  (E.g., prepend AS x to all peers or to specified AS)

- **Enable DDoS defense for customers**
  (E.g., blacklist filters, blackholing by rewriting the next hop to reduce connectivity)

- …
Example 1: BGP routing policy

- A, B, C are provider networks
- X, W, Y are customer (of provider networks)
- X is dual-homed: attached to two networks: goal?
  - X does not want to route from B via X to C
  - .. so X will not advertise to B a route to C
Example 2: BGP routing policy

- A advertises to B the path AW
- B advertises to X the path BAW
- Should B advertise to C the path BAW?
  - No way! B gets no “revenue” for routing CBAW since neither W nor C are B’s customers
  - B wants to force C to route to w via A
  - B wants to route *only* to/from its customers!
BGP route selection

- Router learn > 1 route to some (IP) prefix
- Router must select best route.
- Elimination rules (ordered priorities!): if NEXT-HOP reachable...

1. **Local preference** value attribute: policy decision (not standardized: when new route learned, apply some local policy rules to it)
2. Shortest **AS-PATH**: distance vector part with cost = # AS hops! (it is allowed that a AS appears multiple times consecutively to make it appear longer)
3. **Best MED** (multi-exit-discriminator): for multiple parallel connections to the same AS (**eBGP attribute**: AS tells you on which path it prefers in-bound traffic)
4. Closest **NEXT-HOP** router (gateway router connecting to given next AS): **hot potato routing**
5. Additional criteria
6. **IP address** of peer ("tie breaker")
Local preference attribute

- BGP router may know multiple routes!
  - elimination rules to find best of them
- Admin can specify local preferences => path with highest local preference wins
- Allows providers to prefer routes
  - Local preference, shortest AS-PATH, closest NEXTHOP router (wrt intra AS cost)
Different types of ASs
Different types of ASs

We also use the notion of money transfers between ASes to define valid and invalid AS paths. A valid path between source and destination ASes is one in which for every ISP providing transit (a transit provider), there is a payee. The payee of the transit provider must be its immediate neighbor in the path. An invalid path is one in which there is at least one transit provider not paid by a neighbor in the path.

In Figure 2 the top two examples are valid paths, while the bottom two are invalid. In Example 1 the transit providers are A, B, and C. ISPs B and C pay to A. D pays to B, and F pays to C. In Example 2 the transit providers are B and C, and they are paid by D and F respectively. In contrast, in Example 3 the transit provider is B, but not only does no one pay B, but B itself pays both A and Z. Example 4 also illustrates a situation where nobody pays transit provider B.

We conclude that a valid path must have the following valid path pattern: zero or more c2p links, followed by zero or one p2p link, followed by zero or more p2c links. In addition, s2s links can appear in any number anywhere in the path.

Figure 2. The top two paths 1 and 2 are valid, while the bottom two 3 and 4 are invalid.
Different types of ASs

- Providers: Offer connectivity to direct customers and offer transit to other ISPs
- Customers: Buy connectivity from providers
- Peers: Exchange customers traffic at no cost
- Siblings: others

<table>
<thead>
<tr>
<th></th>
<th>Own Routes</th>
<th>Customer’s Routes</th>
<th>Sibling’s Route</th>
<th>Provider’s Route</th>
<th>Peer’s Route</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exporting to a Provider</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td></td>
</tr>
<tr>
<td>Exporting to a Customer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
<tr>
<td>Exporting to a Peer</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
<td>×</td>
</tr>
</tbody>
</table>
Question

- How to test whether connectivity is preserved when becoming Tier-1 provider?

- E.g., try to announce prefix over all links but not uplinks to Tier-1 provider (which will disappear)...
  - Although Tier-1 providers fully connected, still different policies apply for different Tier-1 providers, just being part does not guarantee connectivity!
  - And depending on previous peering agreements, Tier-1 provider may not always get paid by Tier-2 provider....
Some Wikipedia Stories…

- In 2008, **Pakistan** tried to block YouTube by injecting a wrong path to iBGP. Unfortunately, this path was announced to eBGP also. => YouTube was blocked in whole Asia.

- In 2009, a **Czech** ISP accidentally announced very long AS paths that could not be handled by some routers, which led to problems.

- During the **Egyptian** revolution 2011/12, 3500 routes from Egyptian providers were withdrawn simultaneously, which led to the disconnection of the whole country (apparently, for the first time in the history of the Internet)
Summary: why different Intra- and Inter-AS routing?

Policy:
- Inter-AS: Admin wants control over how its traffic routed, who routes through its net.
- Intra-AS: Single admin, so no policy decisions needed

Scale:
- Hierarchical routing saves table size, reduced update traffic

Performance:
- Intra-AS: Can focus on performance
- Inter-AS: Policy may dominate over performance

We need BOTH!
IP addressing

Important network layer concept...

- Which components have IP addresses?
- Why needed?
IP addressing

- **IP address:** 32-bit identifier for host, router *interface*

- **Interface:** Connection between host, router and physical link
  - Routers typically have multiple interfaces
  - Host may have multiple interfaces
  - IP addresses associated with interface, not host, router

4 x integer representation (more convenient than bits)
IP addressing (2)

- **IP address**: Structure
  - **Network part** (high order bits)
  - **Host part** (low order bits)

- **What’s a network?**
  (from IP address perspective)
  - Device interfaces with same network part of IP address ("prefix")
  - Can physically reach each other **without intervening router**

Network consisting of 3 IP networks
(for IP addresses starting with 223, first 24 bits are network address)
IP addressing (3)

How to find the networks?

- Detach each interface from router, host
- Create “islands” of isolated networks
- So how many networks in this example?

Interconnected system consisting of six networks
IP networks: Subnets

- Sub divide address space
  - network part
  - host address

- Address format: a.b.c.d/x, where x is # bits in subnet portion of address

```
11001000  00010111  00010000  00000000
```

200.23.16.0/24
Fixed subnetting (**classful**)  

- **class**  
  - **A**  
    - 0 network  
    - host  
    - 1.0.0.0 to 127.255.255.255  
    - 128.0.0.0 to 191.255.255.255  
  - **B**  
    - 10 network  
    - host  
    - 192.0.0.0 to 239.255.255.255  
    - 240.0.0.0 to 247.255.255.255  
  - **C**  
    - 110 network  
    - host  
  - **D**  
    - 1110 multicast address  
    - 32 bits  

- **4 Classes: A, B, C, D**  
  - Identified by unique prefix: 0, 10, 110, 1110  
  - Uniquely defines network and host part
Address management

- Problem: We are running out of networks

- Solution (a):
  - **Subnetting**: E.g., Class B Host field (16 bits) is subdivided into <subnet:host> fields

- Solution (b):
  - **CIDR** *(Classless Inter Domain Routing)*

  Allow for arbitrary network/host splits! (prefixes)
CIDR

CIDR: Classless InterDomain Routing

- Motivation
  - Class A is too large, Class C is too small
  - Everyone had a Class B address!!!

- Solution:
  - Sites are given contiguous blocks of class-C addresses (256 addresses each) and a mask or parts of former class A/B networks.
CIDR (2.)

CIDR: Classless InterDomain Routing

- Subnet portion of address **of arbitrary length**
- Address format: `a.b.c.d/x`, where `x` is # bits in subnet portion of address

```
/11001000  00010111 00010000 00000000
\  200.23.16.0/23
```
Q: How does host get IP address?

- Hard-coded by system admin in a file
  - Wintel: Control Panel → Network → Configuration → TCP/IP → Properties
  - UNIX: /etc/rc.config

- DHCP: Dynamic Host Configuration Protocol: dynamically get address from as server
  - “Plug-and-play”

- IP / Subnets allocated by provider (RIPE/ARIN/…)

Stefan Schmid - 72
Hierarchical address structure

- Recall: CIDR

128.119.48.12/18 = 10000000 01110111 00110000 00001100

- High order bits form the **prefix**
- Once inside the network, can further **subnet**: divide remaining bits
- Subnet example:

![Diagram showing hierarchical address structure]

- **Forwarding decision**: Longest prefix match

Note: picture shows prefix **masks**, not interface addrs!
Forwarding vs. routing

- **Forwarding**: the process of moving packets from input to output
  - The forwarding table (**Forwarding Information Base FIB**)  
  - Information in the packet

- **Routing**: process by which the forwarding table is built and maintained
  - One or more routing protocols  
  - Procedures (algorithms) to convert routing info (**Routing Information Base RIB**) to forwarding table.
Forwarding with CIDR

- Packet should be sent toward the interface with the **longest matching prefix**

Where does it go here?
Trie Representation

- Trie = binary tree of prefixes
  - Patricia tree: short cuts (edges = strings)
  - Start from leaf = most specific

```
B  1000 0110
    /        
A  1000 110  1000 1100 1101
    /                /         
C  1000 1101 1000 1101 0110
```

Where does it go here?

```
Advertised address
A  1000 110 1000 1101 00
B  1000 0110
C  1000 1101 1000 1101 001
```
Lookup: Longest prefix match

- Forwarding table:
  \(<Network>/\langle mask\rangle \ <next-hop>\)

- IP Packets: destination IP address
  - Find next-hop via longest prefix match

- Example:

<table>
<thead>
<tr>
<th>Forwarding table (w/ port/hop)</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>134.96.252.0/24</td>
<td>A 134.96.252.200</td>
</tr>
<tr>
<td>134.96.0.0/16</td>
<td>C 134.96.254.2</td>
</tr>
<tr>
<td>134.96.240.0/20</td>
<td>B 134.96.239.200</td>
</tr>
<tr>
<td>134.96.252.192/28</td>
<td>B 134.97.239.200</td>
</tr>
<tr>
<td>134.96.252.128/28</td>
<td>A 134.96.252.191</td>
</tr>
</tbody>
</table>

Use binary representation!
### Lookup: Longest prefix match

#### Example:

<table>
<thead>
<tr>
<th>Forwarding table (w/ port/hop)</th>
<th>Packets</th>
</tr>
</thead>
<tbody>
<tr>
<td>134.96.252.0/24 A 134.96.252.200</td>
<td></td>
</tr>
<tr>
<td>10000110.01100000.11111100.0 10000110.01100000.11111100.11001000</td>
<td></td>
</tr>
<tr>
<td>134.96.0.0/16 C 134.96.254.2</td>
<td></td>
</tr>
<tr>
<td>10000110.01100000.0.0 10000110.01100000.11111110.00000010</td>
<td></td>
</tr>
<tr>
<td>134.96.240.0/20 B 134.96.239.200</td>
<td></td>
</tr>
<tr>
<td>10000110.01100000.11110000.0 10000110.01100000.11101111.11001000</td>
<td></td>
</tr>
<tr>
<td>134.96.252.192/28 B 134.97.239.200</td>
<td></td>
</tr>
<tr>
<td>10000110.01100000.11111100.1100000 10000110.01100000.11101111.11001000</td>
<td></td>
</tr>
<tr>
<td>134.96.252.128/28 A 134.96.252.191</td>
<td></td>
</tr>
<tr>
<td>10000110.01100000.11111100.1000000 10000110.01100000.11111110.1100.1011111</td>
<td></td>
</tr>
</tbody>
</table>
**IP addressing: The last word ...**

**Q:** How does an ISP get block of addresses?  
**A:** ICANN: Internet Corporation for Assigned Names and Numbers (private or public organization?)  
- allocates addresses  
- manages **DNS**  
- assigns domain names, resolves disputes

**Q:** What do I do if I don’t have a public address?  
**A:** Private IP addresses (RFC 1918)  
- 10/8  
- 172.16/12  
- 192.168/16

- Private use only – **not routable** in the Internet
NAT: Network address translation

Motivation: Local network uses just one IP address as far as outside world is concerned:
  - Just one IP address for all devices
  - Why?
    - No needed range of addresses from ISP?
    - Security?
  - But how to demultiplex again?
**NAT: Network address translation (2.)**

All datagrams leaving local network have **same** single source NAT IP address: 138.76.29.7, but different source port numbers (for demultiplexing)

Datagrams with source or destination in this network have 10.0.0/24 address for source, destination (as usual)
NAT: Network address translation (3.)

Motivation: Local network uses just one IP address as far as outside world is concerned:

- Range of addresses not needed from ISP: **just one IP** address for all devices
- Can **change addresses** of devices in local network without notifying outside world
- Can **change ISP** without changing addresses of devices in local network
- Devices inside local net not explicitly addressable, visible by outside world (a **security plus**).
NAT: Network address translation (4.)

Implementation: NAT router must:

- **Outgoing datagrams:** Replace (source IP address, port #) of every outgoing datagram to (NAT IP address, new port #). . . remote clients/servers will respond using (NAT IP address, new port #) as destination addr.

- **Remember (in NAT translation table)** every (source IP address, port #) to (NAT IP address, new port #) translation pair.

- **Incoming datagrams:** Replace (NAT IP address, new port #) in dest fields of every incoming datagram with corresponding (source IP address, port #) stored in NAT table.
NAT: Network address translation (5.)

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

2: NAT router changes datagram source addr from 10.0.0.1, 3345 to 138.76.29.7, 5001, updates table

<table>
<thead>
<tr>
<th>NAT translation table</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>WAN side addr</td>
<td>LAN side addr</td>
</tr>
<tr>
<td>138.76.29.7, 5001</td>
<td>10.0.0.1, 3345</td>
</tr>
<tr>
<td>......</td>
<td>......</td>
</tr>
</tbody>
</table>

3: Reply arrives dest. address: 138.76.29.7, 5001

4: NAT router changes datagram dest addr from 138.76.29.7, 5001 to 10.0.0.1, 3345
NAT: Network address translation (6.)

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address! (not enough for a country...)

- NAT is controversial:
  - Routers should only process up to layer 3 (not ports)
  - Violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
  - Address shortage should instead be solved by IPv6
Data Link Layer (Layer 2)
Data Link Layer


- Principles behind data link layer services
  - Error detection, correction (!)
  - Sharing a broadcast channel: Multiple access
  - Link layer addressing
  - Reliable data transfer, flow control: Done!
- Example link layer technology: Ethernet, 802.11, ...
- Devices: switches/bridges, hubs? (no...), ...
Link layer services

Framing and link access

- Encapsulate datagram: Frame adds header, trailer
- Channel access – if shared medium
- Frame headers use ‘physical addresses’ = “MAC” to identify source and destination
  - Different from IP address!

Reliable delivery (between adjacent nodes)

- Seldom used on low bit error links (fiber optic, co-axial cable and some twisted pairs)
- Sometimes used on high error rate links (e.g., wireless links)
Link layer services (2.)

Flow Control
- **Pacing** between sending and receiving nodes

Error Detection
- Errors are caused by signal attenuation and noise
- Receiver detects presence of errors signals sender for retrans. ("NAK") or drops frame

Error Correction
- Receiver identifies and **corrects** bit error(s) **without resorting to retransmission**

Half-duplex and full-duplex
- With **half duplex**, nodes at both ends of link can transmit, but not at same time
Multiple access links / protocols

Two types of “links”:

- **Point-to-point**
  - PPP for dial-up access
  - Point-to-point link between Ethernet *switch* and host

- **Broadcast** (shared wire or medium)
  - Traditional Ethernet
  - Upstream HFC
  - 802.11 wireless LAN
MAC protocols: Concepts

- **Channel Partitioning**
  - Divide channel into smaller “pieces” (time slots, frequency, ...)
  - Allocate piece to node for exclusive use

- **Random Access**
  - Allow collisions
  - “Recover” from collisions

- **“Taking turns”**
  - Tightly coordinate shared access to avoid collisions

**Goal:** Efficient, fair, simple, decentralized
Addresses: IP vs MAC

IP address (32-bit):
- Network-layer address (for routing!)
- Used to get datagram to destination network (recall IP network definition)

MAC (or LAN or physical or Ethernet) address:
- Data link-layer address
- Used to get datagram from one interface to another physically-connected interface (same network)
- 48 bit MAC address (for most LANs) burned in the adapter ROM
Addresses (2.)

Each adapter on LAN has unique LAN address

Broadcast address = FF-FF-FF-FF-FF-FF-FF

LAN (wired or wireless)

1A-2F-BB-76-09-AD

71-65-F7-2B-08-53

58-23-D7-FA-20-B0

0C-C4-11-6F-E3-98

= adapter
Addresses (3.)

- MAC address allocation administered by IEEE
- Manufacturer buys portion of MAC address space (to assure uniqueness)

Analogy:
- MAC address: Like Social Security Number (never changes)
- IP address: Like postal address (depends on location)

- MAC flat address $\Rightarrow$ portability
  - Can move LAN card from one LAN to another
- IP hierarchical address NOT portable
  - Depends on network to which one attaches
Example: Encapsulation

Starting at A, given IP datagram addressed to B:

- Look up net. address of B, find B on same net. as A (IP subnet)
- Link layer send datagram to B inside link-layer frame
ARP: Address Resolution Protocol

Question: how to determine MAC address of B knowing B’s IP address?

- Each IP node (Host, Router) on LAN has ARP table
- ARP Table: IP/MAC address mappings for some LAN nodes
  - < IP address; MAC address; TTL>
  - TTL (Time To Live): time after which address mapping will be forgotten (typically 20 min) => soft state
ARP Protocol: Same LAN (Network)

- A wants to send datagram to B, and B’s MAC address not in A’s ARP table.
- A broadcasts ARP query packet, containing B's IP address
  - Dest MAC address = FF-FF-FF-FF-FF-FF
  - All machines on LAN receive ARP query
- B receives ARP packet, replies to A with its (B's) MAC address
  - Frame sent to A’s MAC address (unicast)
- A caches (saves) IP-to-MAC address pair in its ARP table until information becomes old (times out)
  - Soft state: information that times out (goes away) unless refreshed
- ARP is “plug-and-play”:
  - Nodes create their ARP tables without intervention from net administrator
Ethernet

“Dominant” LAN technology:

- Cheap!
- First widely used LAN technology
- Simpler, cheaper than token LANs and ATM
- Kept up with speed race: 10 Mbps – 10 Gbps
- Shared medium

Metcalfe’s Etheret sketch
Unreliable, connectionless service

- **Connectionless:**
  No handshaking between sending and receiving adapter.

- **Unreliable:**
  Receiving adapter does not send ACKs or NACKs to sending adapter
  - Stream of datagrams passed to network layer can have **gaps**
  - Gaps will be filled if app is using TCP
  - Otherwise, app will see the gaps
Ethernet uses CSMA/CD

- Carrier-Sense-Multiple-Access & Collision Detection
  - Harder to achieve in wireless setting
- No slots, async.
- Adapter does not transmit if it senses that some other adapter is transmitting, that is: carrier sense
- Transmitting adapter aborts when it senses that another adapter is transmitting, that is: collision detection
- Before attempting a retransmission, adapter waits a random time, that is: random access
Interconnecting LANs

Q: Why not just one big LAN?
- Does not scale...
- All stations must share bandwidth
- Limited cable length
- Large "collision domain" (can collide with many stations)
- Limited number of stations
- Max extent for Ethernet
Interconnecting with hubs

- **Physical Layer** devices: Essentially repeaters operating at bit levels: Repeat received bits on one interface to all other interfaces
- Hubs can be arranged in a **hierarchy** (or multi-tier design), with **backbone** hub at its top
Hubs (2.)

- Each connected LAN referred to as LAN segment
- Hubs do not isolate collision domains: Node may collide with any node residing at any segment in LAN
- Hub Advantages
  - Simple, inexpensive device (well, switches today too...)
  - Multi-tier provides graceful degradation: portions of the LAN continue to operate if one hub malfunctions
  - Extends maximum distance between node pairs (100m per Hub)
Bridges (switches)

- Link Layer devices
  - Stores and forwards Ethernet frames
  - Examines frame header and selectively forwards frame based on MAC dst address
  - When frame is to be forwarded on LAN segment, uses CSMA/CD to access segment

⇒ Bridge isolates collision domains: It buffers frames
Bridges/switch: Advantages

- Scalability
- Higher total max throughput
- No limit on number of nodes
- No limit on geographical coverage
- Can connect different Ethernet types (store and forward)
- Transparent: Hosts do not need to change LAN adapters
- Plug-and-play, self-learning
  - Switches do not need to be configured
Bridges/switch: Forwarding

- Forwarding:
  - To which LAN segment should a frame be forwarded?
  - Looks like a routing problem
A bridge/switch has a bridge/switch table

Entry in table

- (MAC Address, Interface, Time Stamp)
- Stale entries in table dropped (TTL can be 60 min)

Bridge *learns* which hosts can be reached through which interfaces ("MAC learning switch")

- When frame received, switch “learns” location of sender: Incoming LAN segment
- Records sender/location pair in bridge table
Bridges/switch: Filtering/forwarding

When switch receives a frame:

Index switch table using MAC dest address

if entry found for destination

then{

if dest on segment from which frame arrived

then drop the frame (no “loop”!)

else forward the frame on interface indicated

}

else flood

forward on all but the interface on which the frame arrived
Switch: Traffic isolation

- Switch installation breaks subnet into LAN segments
- Switch **filters** packets:
  - Same-LAN-segment frames not usually forwarded onto other LAN segments
  - Segments become separate collision domains
Bridges spanning tree (STP Protocol)

- Avoid cycles
  - Frames may multiply and forwarded forever
- Organize bridges into spanning tree
  - Disable a subset of interfaces
Redundant networks

- Limitations of tree network on layer 2?
- Network with multiple paths (not just tree)
  - Alternate path for each source, destination pair

- Advantage
  - Increased reliability
  - Single network failure OK
  - More opportunities for load distribution

- Disadvantage
  - Added complexity
Bridges vs. Routers

- Both store-and-forward devices
  - Routers: Network layer devices (examine network layer headers)
  - Bridges/switches: Link layer devices

- Use tables
  - Routers: Routing tables via routing algorithms
  - Bridges: Filtering tables via filtering, learning, spanning tree algorithm
Bridges + and -

+ Simple operation
  Low processing bandwidth
- Restricted topologies:
  Spanning tree to avoid cycles
- Single broadcast domain
  No protection from broadcast storms
  (broadcasts will be forwarded by bridge)
Routers + and -

+ Arbitrary topologies
  Limited cycling (TTL and good routing protocols)
+ Firewalls protection
  Against broadcast storms
- Complex operation
  Require IP address configuration (not plug and play)
  Require higher processing bandwidth
Routers vs. Bridges

- **Bridges**
  - Good in small networks (few hundred hosts)

- **Routers**
  - Good in large networks (thousands of hosts)

- **“Layer 3 switch”**
  - Typically high performance switches enhanced by IP functionality
  - Bridge + router (but usually limited routing table!)
  - E.g., acts as a switch inside domain and as a router between domains
  - Other functionality: route caching (same MAC for subsequent packets of flow), etc.
<table>
<thead>
<tr>
<th></th>
<th>hubs</th>
<th>routers</th>
<th>switches</th>
</tr>
</thead>
<tbody>
<tr>
<td>traffic isolation</td>
<td>no</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>plug &amp; play</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
<tr>
<td>optimal routing</td>
<td>no</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>cut through</td>
<td>yes</td>
<td>no</td>
<td>yes</td>
</tr>
</tbody>
</table>

Cut through = forward before having fully read packet (unlike store-and-forward)