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

No application layer, but IP always there...
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

- $c(x,x') = \text{cost of link (}x,x'\text{)}$
  - 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 $\sigma(x_1, x_2, x_3, ..., x_p) = c(x_1, x_2) + c(x_2, x_3) + ... + c(x_{p-1}, x_p)$

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 table!) node along path from source to $v$
- $N'$: Set of nodes whose least cost path definitively known
Dijkstra’s algorithm

1. **Initialization for A:**
2. \( N' = \{A\} \)
3. for all nodes \( v \)
4. \( \text{if } v \text{ adjacent to } A \)
5. \( \text{then } D(v) = c(A,v) \)
6. \( \text{else } D(v) = \infty \)
7. 
8. **Loop**
9. find \( w \) not in \( N' \) such that \( D(w) \) is a minimum
10. add \( w \) to \( N' \)
11. update \( D(v) \) for all \( v \) adjacent to \( w \) and not in \( N' \):
12. \( D(v) = \min(D(v), D(w) + c(w,v)) \)
13. /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v */
14. 
15. **until all nodes in \( N' \)**
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</td>
<td>update!</td>
<td>2,D</td>
<td>infinity</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2,A</td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td>3,E</td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4,E</td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Routes: via opt predecessors!

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

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?
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:
(D,A) = 1
(B,A) = 1
(C,A) = $e$

![Diagram](initial routing)

... recompute routing

![Diagram](recompute)

... recompute

![Diagram](recompute)
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!)
- .. and thus change asynchronously (e.g., at random)
**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: $O(nm)$)

Neighbors known...

Distance vectors from neighbors known...

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

\[ D_x(y) = \text{estimate of least cost } d_x(y) \text{ from } x \text{ to } y \]

\[ \text{Distance vector defined as (distance from } x \text{ to each node in network): } D_x = [D_x(y) : y \in N] \]

\[ \text{Of course, node } x \text{ knows cost to each neighbor } v: c(x,v) \]

\[ \text{Idea: node } x \text{ also maintains its neighbors’ distance vectors (distributed implementation of Bell.-Ford!)} \]

- 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
\[ D_x(y) = \min \{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min \{2+0, 7+1\} = 2 \]

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

**node x table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>from z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
</tr>
</tbody>
</table>

**node y table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**node z table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>from y</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>from z</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Distance vector algorithm

At each node, $x$:

1. Initialization:
2. for all destinations $y$ in $N$:
   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 $w$
Distance vector algorithm (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, recalculates 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 \textit{not} send any message to \( z \).
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>x</th>
<th>y</th>
<th>z</th>
<th>cost to</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### node z table

<table>
<thead>
<tr>
<th>from</th>
<th>x</th>
<th>y</th>
<th>z</th>
<th>cost to</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>4</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>5</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Stefan Schmid - 24
Distance vector: Link cost changes (2.)

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

Diagram:
- Node X connected to node Y with a cost of 4
- Node Y connected to node Z with a cost of 1
- Bad news!!
- Costs: X to Y: 4, Y to Z: 1, X to Z: 50
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...

Why?
z still stores
old value, so:
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. "net"

2. "net"

3. "net unreachable"

4. "Router 3 has path!"

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. (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: but 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

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

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
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!)
Interconnected ASes

- When only one gateway router in ASs…?
  - Intra-routers simply forward to it (scales!)
  - Otherwise flood (how?) inside AS which prefixes available via which gateway, job of Inter-AS protocol
  - 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)
  - 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
OSPF “advanced” features

- **Security**: All OSPF messages are authenticated (to prevent malicious intrusion)
- **Multiple same-cost paths** allowed
- For each link, **multiple cost metrics** for different TOS (eg, 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
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.
Our example...:

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

- Both routers of Y advertize 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!) 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?
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
- Note that BGP sessions do not correspond to physical links.
- Symmetric routing?
  - 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 is a **path vector protocol**

Why? *Scales better, and keeps policy / cost secret!*

Link state for all ~500k prefixes: exchange would take 20 minutes...

- **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

Why? *Loops, policy, etc.*
NEXT-HOP

AS Y may hear two routes to AS X (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!

NEXT-HOP interfaces
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 Import Policies (ingress filter)

Best Route Selection

BGP Route Table

Apply Export Policies (egress filter)

Install Best Routes

IP Forwarding Table
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 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), see next slide
BGP route selection

- Router learn > 1 route to some prefix
- Router must select best route.

Elimination rules:
1. Local preference value attribute: policy decision
2. Shortest AS-PATH
3. Best MED (multi-exit-discriminator): for multiple parallel connections to the same AS (eBGP attribute)
4. Closest NEXT-HOP router: hot potato routing
5. Additional criteria
6. IP address of peer
Different types of ASs

- Providers: Offer connectivity to direct customer 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....
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!