TWO SCALABILITY ISSUES

- **Network Addresses** (scalable but not enough in terms of size)
  - A network with 2 nodes uses an entire Class Network, wasting addresses.
  - A network with slightly > 255 hosts wastes over 64,000 addresses (need B class).

- **Routing** (in Internet)
  - Bellman-Ford (loops & slow convergence)
  - Link State (too much state info)
  - No common metrics (e.g., weights)

---

**Network Addresses**

- **Class B Address**
  - Network #
  - Host #

- **Subnet Mask**
  - Example: 255.255.255.0

- **Subnet Address**
  - Subnet #
  - Host ID

- Make part of host go to network # & call it subnet.
- Configure hosts with a subnet mask.
- Host IP AND Subnet Mask = Subnet #
- All hosts on same subnet give same subnet #
MODIFICATIONS

- Now same network actually means same subnet
- H1 & H2 on same network =>
  \[ IP_1 \land \text{subnet mask} = IP_2 \land \text{subnet mask} \]

- ROUTER TABLE

<table>
<thead>
<tr>
<th>Subnet #</th>
<th>Subnet Mask</th>
<th>Next-Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.96.34.0</td>
<td>255.255.255.128</td>
<td>Interface 0</td>
</tr>
</tbody>
</table>

- Upon receiving a packet, AND its IP with every subnet mask & check if it gives the corresponding subnet #.

ROUTING

- Each network is regarded as autonomous system as each network has an AS # (16 bit #)
- AS is a network under same administrative control
RELATIONSHIPS

- ASes usually define relationships among them
  - Customer - Provider
  - Peers

- Routing is done depending on policies
  - An AS might not want to route through another AS
  - An AS might want to route through a peer (instead of going to provider) to save $$$ (peers usually don't exchange $$$)

EXAMPLE:

- →: Provider → Customer
- ••: Peer to peer

Outside world

Shortcut
BGP
(BORDER GATEWAY PROTOCOL)

- EVERY AS HAS A BGP SPEAKER & BGP GATEWAY

- BGP SPEAKERS EXCHANGE ROUTE INFORMATION
- GATEWAY: POINT OF ENTRY TO AS
- BGP & GATEWAY NEED NOT BE THE SAME, BUT THEY COULD BE
EXCHANGING ROUTES

* EVERY AS DECIDES TO ADVERTISE PATHS IT KNOWS TO OTHER AS (COMPLETE PATH)

```
  A
 / \
B   C
  \
X   Y
```

* FOR INSTANCE, C COULD ADVERTISE TO B & A THAT IT CAN REACH Y & X

* EVERY AS ALSO DECIDES TO ADVERTISE INFORMATION IT KNOWS FROM OTHER ASes.

LOOPS

* C ADVERTISES Y TO A: (C, Y)
* A ADVERTISES Y TO B: (A, C, Y)
* B ADVERTISES Y TO C: (B, A, C, Y)

* WHAT IF C DECIDES TO SEND TO Y THROUGH B?
  \[ \Rightarrow \text{LOOP} \]

* EVERY AS CHECKS IF IT IS PART OF THE PATH BEFORE ACCEPTING IT \[ \Rightarrow \text{DISCARD PATHS CONTAINING ID OF THE AS ITSELF} \]
WHAT TO IMPORT, WHAT TO EXPORT?

- Usually we would like "valley-free" routes
  - number links (+1, 0, -1) for provider, peer, customer
  - in any path we want sequence of +1, followed by at most one 0, followed by sequence of -1.

EXAMPLE

---: not allowed
-: allowed

IMPORTING RULES

- Import everything but give preference to
  - customer first
  - then peer
  - then provider

- Then among different paths, choose shortest.

B will eventually learn

1. $X$
2. $(C, X)$
3. $(A, C, X)$

priority
EXPORTING RULES

• Export customers routes to everyone
  (Need everyone to be able to reach your customers)

• Export routes to your own addresses to everyone
  (Need to be connected)

• Don’t export routes advertised by your provider
  (May advertise to customers) (Valley-free routes)

• Don’t export routes advertised by peer
  (May advertise to customers) (Valley-free routes)
MINIMUM SPANNING TREE

- Assume undirected graph
- Each edge has weight

\[ \sum_{(u,v) \in T} w(u,v) \text{ minimized} \]

EXAMPLE

\[ w(T) = 3 + 8 + 2 + 15 + 9 + 5 + 4 = 46 \]
**How do we find it?**

- Let $T$ be the MST & let $A \subseteq T$ be a subtree of $T$.
- Let $(u,v)$ be minimum weight edge that connects $A$ to $V-A$.

Then $(u,v) \in T$.

**Proof**

Assume $T$ contains $A$ but not $(u,v)$.

Add $(u,v)$, remove $e$, obtain $T'$ with $w(T') \leq w(T)$. 
PRIM'S ALGORITHM

1. START WITH ANY NODE
2. BUILD THE TREE BY ALWAYS CHOOSING THE EDGE WITH SMALLEST WEIGHT THAT GOES OUTSIDE
This is similar to Dijkstra's Algorithm

INIT
key(s) ← 0
key(u) ← ∞ u ≠ s

ALL
S ← Ø
while (S ≠ V)
do choose u ≠ S with smallest key
S ← S ∪ {u}
for every (u, v) ∈ E & v ≠ S
if w(u, v) < key (v)
then key(v) ← w(u, v)
p(v) ← u

KRUSKAL'S ALGORITHM

1. REPEATEDLY CHOOSE THE SMALLEST WEIGHT EDGE THAT DOES NOT CREATE A CYCLE & ADD IT
2. ALSO CAN BE PROVED TO WORK
MINIMUM SPANNING TREE & SHORTEST PATH TREE

- THEY ARE NOT THE SAME.
- COUNTER EXAMPLE

- SHORTEST PATH TREE FOR A AS SOURCE:

- MINIMUM SPANNING TREE STARTING WITH A:

No shortest path tree is a minimum spanning tree.