Network Layer

CMPS 4750/6750: Computer Networks
Outline

- Overview of network layer
  - Forwarding (data plane)
  - Routing (control plane)
- The Internet Protocol (IP)
- Routing in the Internet: OSPF, BGP
Network Layer

- transport segment from sending to receiving host
- on sending side encapsulates segments into datagrams
- on receiving side, delivers segments to transport layer
- network layer protocols in *every* host & router
- router examines header fields in all IP datagrams passing through it
Two key network-layer functions

- **forwarding**: move packets from router’s input to appropriate router output

- **routing**: determine route taken by packets from source to destination
  - *routing algorithms*
Network layer: data plane, control plane

**Data plane**
- local, per-router function
  - forwarding
  - dropping
  - modify field
  - ...

**Control plane**
- network-wide logic
  - routing
  - access control
  - load balancing
  - ...

- two control-plane approaches:
  - *traditional routing algorithms*: implemented in routers
  - *software-defined networking (SDN)*: implemented in (remote) servers
Per-router control plane

- Individual routing algorithm components *in each and every router* interact in the control plane.
Logically centralized control plane

- A distinct (typically remote) controller interacts with local control agents (CAs)

values in arriving packet header
Network service model

Q: What service model for “channel” transporting datagrams from sender to receiver?

example services for individual datagrams:
- guaranteed delivery
- guaranteed delivery with less than 40 msec delay

example services for a flow of datagrams:
- in-order datagram delivery
- guaranteed minimum bandwidth to flow
- restrictions on changes in inter-packet spacing

The Internet’s network layer provides “best-effort” service
Outline

- Overview of network layer
- Forwarding (data plane)
- Routing (control plane)
- The Internet Protocol (IP)
- Routing in the Internet: OSPF, BGP
Router architecture overview

- **routing processor**
- **high-seed switching fabric**
- **router input ports**
- **router output ports**

**Forwarding data plane** (hardware) operates in nanosecond timeframe.

**Routing, management control plane** (software) operates in millisecond timeframe.
Input port functions

- **Physical layer**: bit-level reception
- **Data link layer**: e.g., Ethernet, see chapter 5

**Decentralized switching**:
- Using header field values, lookup output port using forwarding table in input port memory ("match plus action")
- Goal: complete input port processing at 'line speed'
- Queuing: if datagrams arrive faster than forwarding rate into switch fabric

- **Line termination**
- **Link layer protocol (receive)**
- **Lookup, forwarding queueing**
Input port functions

- **physical layer**: bit-level reception
- **data link layer**: e.g., Ethernet see chapter 5
- **decentralized switching**:
  - using header field values, lookup output port using forwarding table in input port memory ("match plus action")
  - destination-based forwarding: forward based only on destination IP address (traditional)
  - generalized forwarding: forward based on any set of header field values
## Destination-based forwarding

A table for forwarding addresses:

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000 through 11001000 00010111 00111111 11111111</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>
### Destination-based forwarding

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010000 00000000 through 11001000 00010111 00010111 11111111</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 00000000 through 11001000 00010111 00011000 11111111</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011001 00000000 through 11001000 00010111 00011111 11111111</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>
Longest prefix matching

<table>
<thead>
<tr>
<th>Destination Address Range</th>
<th>Link interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>11001000 00010111 00010*** *********</td>
<td>0</td>
</tr>
<tr>
<td>11001000 00010111 00011000 *********</td>
<td>1</td>
</tr>
<tr>
<td>11001000 00010111 00011*** *********</td>
<td>2</td>
</tr>
<tr>
<td>otherwise</td>
<td>3</td>
</tr>
</tbody>
</table>

Examples: DA: 11001000 00010111 00010110 10100001 which interface? 0
          DA: 11001000 00010111 00011000 10101010 which interface? 1

*longest prefix matching*

when looking for forwarding table entry for given destination address, use *longest* address prefix that matches destination address.
Switching fabrics

- transfer packets from input buffer to appropriate output buffer

- switching rate: rate at which packets can be transfer from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
Crossbar switches

- at any time, one input point can be connected to at most one output port, and vice versa
- a schedule in a crossbar switch corresponds to a matching in the corresponding bipartite graph
Input port queuing

- fabric slower than input ports combined -> queueing may occur at input queues
  - queueing delay and loss due to input buffer overflow!

- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
Output ports

- **buffering** required when datagrams arrive from fabric faster than the transmission rate.

- **scheduling** chooses among queued datagrams for transmission.

Datagram (packets) can be lost due to congestion, lack of buffers.

Output port queueing

- buffering when arrival rate via switch exceeds output line speed
- *queueing (delay) and loss due to output port buffer overflow!*

at $t$, packets more from input to output

one packet time later (assume switch operates at three times the line speed)
Scheduling mechanisms

- **scheduling**: choose next packet to send on link

- **FCFS (first-come-first-served) scheduling**: send in order of arrival to queue
  - Also known as *first-in-first-out, FIFO*
  - real-world example?
  - **discard policy**: if packet arrives to full queue: who to discard?
    - *tail drop*: drop arriving packet
    - *priority*: drop/remove on priority basis
    - *random*: drop/remove randomly
Scheduling policies: priority

- **priority scheduling**: send highest priority queued packet
- multiple *classes*, with different priorities
  - class may depend on marking or other header info, e.g. IP source/dest, port numbers, etc.
  - real world example?
Scheduling policies: still more

Round Robin (RR) scheduling:

- multiple classes
- cyclically scan class queues, sending one complete packet from each class (if available)
Scheduling policies: still more

*Weighted Fair Queuing (WFQ)*:

- generalized Round Robin
- each class gets weighted amount of service in each cycle
Outline

- Overview of network layer
- Forwarding (data plane)
- Routing (control plane)
- The Internet Protocol (IP)
- Routing in the Internet: OSPF, BGP
Network-layer functions

Recall: two network-layer functions:

- **forwarding**: move packets from router’s input to appropriate router output

- **routing**: determine route taken by packets from source to destination

Two approaches to structuring network control plane:

- per-router control (traditional)
- logically centralized control (software defined networking)
Routing protocols

*Goal:* determine “good” paths (equivalently, routes), from sending hosts to receiving host, through network of routers

- path: sequence of routers packets will traverse in going from given initial source host to given final destination host
- “good”: least “cost”, “fastest”, “least congested”
- routing: a “top-10” networking challenge!
Graph abstraction of the network

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

- **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) \}

**Aside:** Graph abstraction is useful in other network contexts, e.g., P2P, where \( N \) is set of peers and \( E \) is set of TCP connections
Graph abstraction: costs

$c(x,x') = \text{cost of link } (x,x')$

e.g., $c(w,z) = 5$

cost could always be 1, or inversely related to bandwidth, or related to congestion or delay

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

**key question:** what is the least-cost path between u and z?

**routing algorithm:** algorithm that finds that least cost path
Routing algorithm classification

Q: global or decentralized information?

**global:**
- all routers have complete topology, link cost info
- “link state” algorithms

**decentralized:**
- router knows physically-connected neighbors, link costs to neighbors
- iterative process of computation, exchange of info with neighbors
- “distance vector” algorithms

Q: static or dynamic?

**static:**
- routes change slowly over time

**dynamic:**
- routes change more quickly
  - periodic update
  - in response to link cost changes
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 *forwarding table* for that node
- iterative: after k iterations, know least cost path to k dest.’s

*notation:*

- $c(x,y)$: link cost from node $x$ to $y$; $= \infty$ if not direct neighbors
- $D(v)$: current value of cost of path from source to dest. $v$
- $p(v)$: predecessor node along path from source to $v$
- $N'$: set of nodes whose least cost path definitively known
Dijsktra’s algorithm

1 *Initialization:*
2 N' = \{u\}
3 for all nodes v
4 if v adjacent to u
5 then D(v) = c(u,v)
6 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 for all v adjacent to w and not in N' :
12 D(v) = \min(D(v), D(w) + c(w,v))
13 until all nodes in N'

new cost to v is either old cost to v or known shortest path cost to w plus cost from w to v
Dijkstra’s algorithm: example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>D(v)</th>
<th>D(w)</th>
<th>D(x)</th>
<th>D(y)</th>
<th>D(z)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>p(v)</td>
<td>p(w)</td>
<td>p(x)</td>
<td>p(y)</td>
<td>p(z)</td>
</tr>
<tr>
<td>0</td>
<td>u</td>
<td>7,u</td>
<td>3,u</td>
<td>5,u</td>
<td>∞</td>
<td>∞</td>
</tr>
<tr>
<td>1</td>
<td>uw</td>
<td>6,w</td>
<td>5,u</td>
<td>11,w</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uwx</td>
<td>6,w</td>
<td></td>
<td>11,w</td>
<td>14,x</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uwxv</td>
<td></td>
<td></td>
<td>10,v</td>
<td>14,x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uwxvvy</td>
<td></td>
<td></td>
<td></td>
<td>12,y</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uwxvzy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**notes:**
- construct **shortest path tree** by tracing predecessor nodes
- ties can exist (can be broken arbitrarily)

resulting forwarding table in u:

<table>
<thead>
<tr>
<th>destination</th>
<th>link</th>
</tr>
</thead>
<tbody>
<tr>
<td>v</td>
<td>(u,w)</td>
</tr>
<tr>
<td>w</td>
<td>(u,w)</td>
</tr>
<tr>
<td>x</td>
<td>(u,x)</td>
</tr>
<tr>
<td>y</td>
<td>(u,w)</td>
</tr>
<tr>
<td>z</td>
<td>(u,w)</td>
</tr>
</tbody>
</table>

![Graph diagram](image-url)
Complexity of Dijkstra’s algorithm

For a given network $G(N, E)$

- each iteration: need to check all nodes not in $N'$ and edges adjacent to $w$
- $|N|(|N| + 1)/2$ comparisons + $O(|E|)$ updates: $O(|N|^2)$
- more efficient implementations possible: $O(|N| \log |N| + |E|)$
Distance vector algorithm

Bellman-Ford equation (dynamic programming)

let

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

then

\[ d_x(y) = \min_v \{ c(x,v) + d_v(y) \} \]

\( c(x,v) \) is the cost to neighbor \( v \) of \( x \)
\( d_v(y) \) is the cost from neighbor \( v \) to destination \( y \)
\( \min \) taken over all neighbors \( v \) of \( x \)
Bellman-Ford example

clearly, \(d_v(z) = 5\), \(d_x(z) = 3\), \(d_w(z) = 3\)

B-F 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 achieving minimum is next hop in shortest path, used in forwarding table
Distance vector algorithm

- node \( x \):
  - knows cost to each neighbor \( v \): \( c(x, v) \)
  - \( x \) maintains distance vector \( D_x = [D_x(y) : y \in N] \)
    - \( D_x(y) = \) estimate of least cost from \( x \) to \( y \)
  - maintains its neighbors’ distance vectors
    - For each neighbor \( v \), \( x \) maintains \( D_v = [D_v(y) : y \in N] \)
Distance vector algorithm

key idea:

- from time-to-time, each node sends its own distance vector estimate to neighbors
- when x receives new DV estimate from neighbor, it updates its own DV using B-F equation:

\[ D_x(y) \leftarrow \min_v \{c(x, v) + D_v(y)\} \text{ for each node } y \in N \]
Distance vector algorithm

Each node $x$

- start with known costs to neighbors
- calculate initial estimate of $D_x = \{D_x(y), y \in N\}$
- send distance vector to neighbors
- **wait** for change in local link cost or msg from neighbor
- **recompute** $D_x$ using Bellman-Ford equation
- if $D_x(y)$ changed for any $y$, **notify** neighbors

- **distributed, asynchronous** algorithm
- under minor, natural conditions, the estimate $D_x(y)$ converge to the actual least cost $d_x(y)$
\[ D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} = \min\{2+0, 7+1\} = 2 \]

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

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

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

<table>
<thead>
<tr>
<th>node y table</th>
<th>cost to</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>node z table</th>
<th>cost to</th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>from</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>x</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>∞</td>
<td>∞</td>
<td>∞</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>time</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
</table>
Distance vector: link cost changes

*link cost changes:*
- node detects local link cost change
- updates routing info, recalculates distance vector
- if DV changes, notify neighbors

“good news travels fast”

$t_0$: $y$ detects link-cost change, updates its DV, informs its neighbors.

$t_1$: $z$ receives update from $y$, updates its table, computes new least cost to $x$, sends its neighbors its DV.

$t_2$: $y$ receives $z$’s update, updates its distance table. $y$’s least costs do not change, so $y$ does not send a message to $z$. 
Distance vector: link cost changes

**link cost changes:**
- node detects local link cost change
- may have routing loops during convergence
- *bad news travels slow* - “count-to-infinity” problem!

<table>
<thead>
<tr>
<th>$t$</th>
<th>$D_y(x)$</th>
<th>$D_z(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>$\min(60 + 0, 1 + 5) = 6$</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>6</td>
<td>$\min(50 + 0, 1 + 6) = 7$</td>
</tr>
<tr>
<td>3</td>
<td>$\min(60 + 0, 1 + 7) = 8$</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>8</td>
<td>$\min(50 + 0, 1 + 8) = 9$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>46</td>
<td>50</td>
<td>$\min(50 + 0, 1 + 50) = 50$</td>
</tr>
<tr>
<td>47</td>
<td>$\min(60 + 0, 1 + 50) = 51$</td>
<td>50</td>
</tr>
<tr>
<td>48</td>
<td>51</td>
<td>$\min(50 + 0, 1 + 51) = 50$</td>
</tr>
</tbody>
</table>
Distance vector: link cost changes

poisoned reverse:

- If Z routes through Y to get to X:
  - Z tells Y its (Z’s) distance to X is infinite (so Y won’t route to X via Z)
- will this completely solve count-to-infinity problem?

<table>
<thead>
<tr>
<th>$t$</th>
<th>$D_Y(x)$</th>
<th>$D_Z(x)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>$\min(60 + 0, 1 + \infty) = 60$</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>$\min(50 + 0, 1 + 60) = 50$</td>
</tr>
<tr>
<td>3</td>
<td>$\min(60 + 0, 1 + 50) = 51$</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>51</td>
<td>$\min(50 + 0, 1 + \infty) = 50$</td>
</tr>
</tbody>
</table>
Comparison of LS and DV algorithms

**message complexity**
- **LS**: with \( n \) nodes, \( E \) links, \( O(nE) \) msgs sent
- **DV**: exchange between neighbors only
  - convergence time varies

**speed of convergence**
- **LS**: \( O(n^2) \) algorithm requires \( O(nE) \) msgs
- **DV**: convergence time varies
  - may be routing loops
  - count-to-infinity problem

**robustness**: what happens if router malfunctions?

**LS**:
- node can advertise incorrect *link* cost
- each node computes only its *own* table

**DV**:
- DV node can advertise incorrect *path* cost
- each node’s table used by others
  - error propagate thru network
Lab 3: Distance Vector Routing

Distance table at node 0
- $dt\cdot costs[4][4]$: 4-by-4 array of int's
- $dt\cdot costs[i,j]$: node 0's currently computed cost from 0 to j via direct neighbor i
(a) linear increase, with equal linear decrease

(b) linear increase, connection 1 decrease is twice that of connection 2
Outline

- Overview of network layer
- Forwarding (data plane)
- Routing (control plane)
- The Internet Protocol (IP): IPv4, DHCP, NAT, IPv6
- Routing in the Internet: OSPF, BGP
The Internet network layer

host, router network layer functions:

- **routing protocols**
  - path selection
  - RIP, OSPF, BGP

- **IP protocol**
  - addressing conventions
  - datagram format
  - packet handling conventions

- **ICMP protocol**
  - error reporting
  - router “signaling”
IP datagram format

- **IP protocol version number** (4 bytes)
- **Header length** (variable length)
- **“Type” of data** (20 bytes)
- **Max number of hops** (32 bits)
- **Upper layer protocol** to deliver payload to
- **Header checksum** (32 bits)
- **Time to live** (16 bits)
- **Upper layer** header
- **Options (if any)**
- **32-bit source IP address**
- **32-bit destination IP address**
- **Total datagram length**: header + data

**How much overhead?**
- 20 bytes of TCP
- 20 bytes of IP
- = 40 bytes + app layer overhead

Additional fields:
- **Type of service** (8 bits)
- **Fragment offset** (13 bits)
- **Flags** (3 bits)
- **Identifier** (16 bits)
- **Fragmentation**/reassembly
- **Checksum** (16 bits)
- **Routing** information
- **Options**
- **Data** (variable length, typically a TCP or UDP segment)
- **Total datagram length** (32 bits)
- **E.g. timestamp, record route taken, specify list of routers to visit.**
IP fragmentation, reassembly

- Network links have MTU (maximum transmission unit) - largest possible link-level frame
  - Different link types, different MTUs

- Large IP datagram divided ("fragmented") within net
  - One datagram becomes several datagrams
  - "reassembled" only at final destination
  - IP header bits used to identify, order related fragments
IP fragmentation, reassembly

example:
- 4000 byte datagram
- MTU = 1500 bytes

one large datagram becomes several smaller datagrams

1480 bytes in data field

offset = 1480/8
IP addressing: introduction

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

- **interface**: boundary between host/router and physical link
  - routers typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)

- **IP addresses associated with each interface**

```
223.1.1.2 = 11011111 00000010 00000010 00000010
223.2.1.9 = 11011111 00000000 00000000 00010001
```

```
223.1.1.1 = 11011111 00000001 00000001 00000001
```

```
223.1.2.2 = 11011111 11100000 00000000 00000010
223.1.3.2 = 11011111 00000100 00000000 00000010
```

```
223.1.3.27 = 11011111 00000000 00000000 01110111
```

```
223.1.2.1 = 11011111 00000000 00100000 00000001
223.1.3.1 = 11011111 00000000 10000000 00000001
```

```
223.1.1.3 = 11011111 00000001 10000000 00000001
223.1.3.27 = 11011111 00000000 00000000 01110111
```

```
223.1.1.4 = 11011111 00000000 00000000 10100010
223.1.3.2 = 11011111 00000100 00000000 00000010
```

```
223.1.2.9 = 11011111 00000000 00010001 00000000
223.1.3.27 = 11011111 00000000 00000000 01110111
```

```
223.1.1.1 = 11011111 00000001 00000001 00000001
```

```
223.1.2.1 = 11011111 00000000 00100000 00000001
223.1.3.1 = 11011111 00000000 10000000 00000001
```

```
223.1.1.3 = 11011111 00000001 10000000 00000001
223.1.3.27 = 11011111 00000000 00000000 01110111
```

```
223.1.1.4 = 11011111 00000000 00000000 10100010
223.1.3.2 = 11011111 00000100 00000000 00000010
```

```
223.1.2.9 = 11011111 00000000 00010001 00000000
223.1.3.27 = 11011111 00000000 00000000 01110111
```

```
223.1.1.1 = 11011111 00000001 00000001 00000001
```

IP addressing: introduction

wired Ethernet interfaces connected by Ethernet switches

wireless WiFi interfaces connected by WiFi base station
Subnets

- **IP address:**
  - subnet part - high order bits
  - host part - low order bits

- **what's a subnet?**
  - device interfaces with same subnet part of IP address
  - can physically reach each other *without* intervening router

**subnet mask: /24**

```
223.1.0/24
223.1.1.0/24
223.1.2.0/24
223.1.3.0/24
```

```
223.1.1.1   223.1.1.2   223.1.1.3   223.1.1.4
223.1.2.9   223.1.2.1   223.1.3.27
```

```
223.1.3.1   223.1.3.2
```

223.1.2.2
Subnets

*recipe*

- to determine the subnets, detach each interface from its host or router, creating islands of isolated networks
- each isolated network is called a subnet

*subnet mask: /24*
Subnets

how many subnets?
IP addressing: CIDR

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

subnet part  host part

11001000  00010111  0001000  00000000

200.23.16.0/23
Hierarchical addressing: route aggregation

Hierarchical addressing allows efficient advertisement of routing information:

ISP 1

Organization 0
200.23.16.0/23

Organization 1
200.23.18.0/23

Organization 2
200.23.20.0/23

Organization 7
200.23.30.0/23

ISP 2

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16”

Internet

Send me anything with addresses beginning 200.23.16.0/20
Hierarchical addressing: route aggregation

ISP 2 has a more specific route to Organization 1

Organization 0
- 200.23.16.0/23

Organization 2
- 200.23.20.0/23

Organization 7
- 200.23.30.0/23

Organization 1
- 200.23.18.0/23

ISP 2

“Send me anything with addresses beginning 200.23.16.0/20”

ISP 1

“Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23”
IP addresses: how to get one?

**Q:** how does network get subnet part of IP addr?

**A:** gets allocated portion of its provider ISP’s address space

<table>
<thead>
<tr>
<th>ISP’s block</th>
<th>11001000 00010111 00010000 00000000</th>
<th>200.23.16.0/20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization 0</td>
<td>11001000 00010111 00010000 00000000</td>
<td>200.23.16.0/23</td>
</tr>
<tr>
<td>Organization 1</td>
<td>11001000 00010111 00010010 00000000</td>
<td>200.23.18.0/23</td>
</tr>
<tr>
<td>Organization 2</td>
<td>11001000 00010111 00010100 00000000</td>
<td>200.23.20.0/23</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
<td>.....</td>
</tr>
<tr>
<td>Organization 7</td>
<td>11001000 00010111 00011110 00000000</td>
<td>200.23.30.0/23</td>
</tr>
</tbody>
</table>

**Q:** how does an ISP get block of addresses?

**A:** ICANN: Internet Corporation for Assigned Names and Numbers [http://www.icann.org](http://www.icann.org)
IP addresses: how to get one?

Q: How does a host get IP address?

- hard-coded by system admin in a file
  - Windows: 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”
DHCP: Dynamic Host Configuration Protocol

**goal:** allow host to *dynamically* obtain its IP address from network server when it joins network

• can renew its lease on address in use
• allows reuse of addresses (only hold address while connected/“on”)
• support for mobile users who want to join network (more shortly)
DHCP client-server scenario

arriving DHCP client needs address in this network
DHCP client-server scenario

- DHCP messages exchanged through UDP
- 255.255.255.255 - IP broadcast address: message delivered to all hosts on the same subnet
DHCP: Dynamic Host Configuration Protocol

DHCP can return more than just allocated IP address on subnet:

• address of first-hop router for client
• name and IP address of DNS sever
• network mask (indicating network versus host portion of address)
NAT: network address translation

- IPv4 has ~4.3 billion IP addresses, but we have
  - ~7.6 billion people in 2018, each with multiple devices
  - ~30 billion Internet of Things (IoT) devices in 2020

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
  - devices inside local net not explicitly addressable, visible by outside world (a security plus)
NAT: network address translation

**Private IP addresses:**

- 10.x.x.x
- 192.168.x.x
- 172.16.0.0 – 172.31.255.255

---

**Diagram:**

- **Local network** (e.g., home network) 10.0.0/24
- **Rest of Internet**
- **Source NAT IP address:** 138.76.29.7
- **Source port numbers**
- **Datagrams leaving local network** have **same** single source NAT IP address: 138.76.29.7, different source port numbers
- **Datagrams with source or destination in this network** have 10.0.0/24 address for source, destination (as usual)
**NAT: network address translation**

**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

**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 translation table**

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

---

**Notes:**
- The NAT router performs network address translation (NAT) to map private IP addresses on the local network to public IP addresses on the Internet.
- This allows multiple devices on the local network to share a single public IP address.
- The translation table keeps track of the mappings for efficient routing of incoming and outgoing traffic.
NAT: network address translation

- 16-bit port-number field:
  - 60,000 simultaneous connections with a single LAN-side address!

- NAT is controversial:
  - routers should only process up to layer 3
  - address shortage should be solved by IPv6
  - NAT traversal: what if client wants to connect to server behind NAT?
IPv6: motivation

- **initial motivation**: 32-bit address space soon to be completely allocated.

- **additional motivation**:
  - header format helps speed processing/forwarding
  - header changes to facilitate QoS

*IPv6 datagram format*:

- fixed-length 40 byte header
- no fragmentation allowed
**IPv6 datagram format**

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
<th>payload len</th>
<th>next hdr</th>
<th>hop limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>source address (128 bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>destination address (128 bits)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Priority (traffic class):** identify priority among datagrams in flow
- **flow Label:** identify datagrams in same “flow”
- **next header:** identify upper layer protocol for data
- **header checksum:** removed entirely to reduce processing time at each hop
- **options:** allowed, but outside of header, indicated by “Next Header” field
Transition from IPv4 to IPv6

- not all routers can be upgraded simultaneously
  - no “flag days”
  - how will network operate with mixed IPv4 and IPv6 routers?

- **tunneling**: IPv6 datagram carried as *payload* in IPv4 datagram among IPv4 routers
Tunneling

physical view:

logical view:

A IPv6
B IPv6
C IPv4
D IPv4
E IPv6
F IPv6

A IPv6
B IPv6
C IPv4
D IPv4
E IPv6
F IPv6

flow: X
src: A
dest: E
data

Flow: X
Src: A
Dest: F
data

A-to-B: IPv6
B-to-C: IPv6 inside IPv4
D-to-E: IPv6 inside IPv4
E-to-F: IPv6

IPv4 tunnel connecting IPv6 routers

src: B
dest: E
data

src: B
dest: E
data

src: B
dest: E
data

Flow: X
Src: A
Dest: F
data

Flow: X
Src: A
Dest: F
data

Flow: X
Src: A
Dest: F
data
Outline

- Overview of network layer
- Forwarding (data plane)
- Routing (control plane)
- The Internet Protocol (IP)
- Routing in the Internet: OSPF, BGP (not required for final)
Making routing scalable

our routing study thus far - idealized

- all routers identical
- network “flat”

... not true in practice

**scale:** with billions of destinations:

- can’t store all destinations in routing tables!
- routing table exchange would swamp links!

**administrative autonomy**

- internet = network of networks
- each network admin may want to control routing in its own network
Internet approach to scalable routing

aggregate routers into regions known as “autonomous systems” (AS) (a.k.a. “domains”)

intra-AS routing

- routing among hosts, routers in same AS (“network”)
- all routers in AS must run same intra-domain protocol
- routers in different AS can run different intra-domain routing protocol

inter-AS routing

- routing among AS’es
- gateway router: at “edge” of its own AS, has link(s) to router(s) in other AS’es
- gateways perform inter-domain routing (as well as intra-domain routing)
Interconnected ASes

- forwarding table configured by both intra- and inter-AS routing algorithm
  - intra-AS routing determine entries for destinations within AS
  - inter-AS & intra-AS determine entries for external destinations
Intra-AS Routing

- also known as *interior gateway protocols (IGP)*
- most common intra-AS routing protocols:
  - RIP: Routing Information Protocol
  - OSPF: Open Shortest Path First (IS-IS protocol essentially same as OSPF)
  - IGRP: Interior Gateway Routing Protocol (Cisco proprietary for decades, until 2016)
OSPF (Open Shortest Path First)

- “open”: publicly available
- uses link-state algorithm
  - link state packet dissemination
  - topology map at each node
  - route computation using Dijkstra’s algorithm
- router floods OSPF link-state advertisements to all other routers in entire AS
  - carried in OSPF messages directly over IP (rather than TCP or UDP)
- “advanced” features: security, multiple same-cost paths, etc.
Hierarchical OSPF

- **Area 1**: Internal routers
- **Area 2**: Internal routers, area border routers
- **Area 3**: Internal routers
- **Backbone**: Backbone router, area border routers
- **Boundary Router**
Inter-AS tasks

- suppose router in AS1 receives datagram destined outside of AS1:
  - router should forward packet to gateway router, but which one?

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

  *job of inter-AS routing!*
Internet inter-AS routing: BGP

- BGP (Border Gateway Protocol): *the* de facto inter-domain routing protocol
  - “glue that holds the Internet together”

- BGP provides each AS a means to:
  - allows subnet to advertise its existence to rest of Internet: “*I am here*”
  - obtain subnet reachability information from neighboring ASes
  - propagate reachability information to all AS-internal routers.
  - determine “good” routes to other networks based on reachability information and *policy*
BGP connections

- **BGP connection**: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection.

Gateway routers run both eBGP and iBGP protocols.
BGP basics

- **BGP connection**: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection:
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)

- When AS3 gateway router 3a advertises path *AS3,X* to AS2 gateway router 2c:
  - AS3 *promises* to AS2 it will forward datagrams towards X
BGP path advertisement

- AS2 router 2c receives path advertisement AS3,X (via eBGP) from AS3 router 3a
- Based on AS2 policy, AS2 router 2c accepts path AS3,X, propagates (via iBGP) to all AS2 routers
- Based on AS2 policy, AS2 router 2a advertises (via eBGP) path AS2, AS3, X to AS1 router 1c
BGP path advertisement

gateway router may learn about multiple paths to destination:

- AS1 gateway router 1c learns path AS2,AS3,X from 2a
- AS1 gateway router 1d learns path AS3,X from 3d
Path attributes and BGP routes

- advertised prefix includes BGP attributes
  - prefix + attributes = “route”

- two important attributes:
  - **AS-PATH**: list of ASes through which prefix advertisement has passed
  - **NEXT-HOP**: indicates specific internal-AS router to next-hop AS
Path attributes and BGP routes

- IP address of leftmost interface for router 2a; **AS2,AS3;X**
- IP address of leftmost interface for router 3d; **AS3;X**
1b learns (via iBGP) it can route to X via 2a or 3d

*hot potato routing*: choose route with the least cost to NEXT-HOP router: get packets out of its AS as quickly as possible!

1b and 1d may choose different AS paths to the same prefix
BGP route selection

- router may learn about more than one route to destination AS, selects route based on:
  1. local preference value attribute: policy decision
     • e.g., never route through AS Y
     • AS policy also determines whether to *advertise* path to other other neighboring ASes
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria
BGP: achieving policy via advertisements

- A, B, C are provider networks
- X, W, Y are customer (of provider networks)
- X is dual-homed: attached to two networks
- policy to enforce: X does not want to route from B to C via X
  - so X will not advertise to B a route to C
A advertises path Aw to B and to C

B *chooses not to advertise* BAw to C:

- B gets no “revenue” for routing CBAw, since none of C, A, w are B’s customers
- C does not learn about CBAw path

C will route CAw (not using B) to get to w

Usually, an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)
Why different Intra-, Inter-AS routing?

*policy:*
- intra-AS: single admin, no policy decisions needed
- inter-AS: admin wants control over how its traffic routed, who routes through its net.

*scale:*
- hierarchical routing saves table size, reduced update traffic

*performance:*
- intra-AS: can focus on performance
- inter-AS: policy may dominate over performance