Network Layer

Goals:

- understand principles behind network layer services, focusing on data plane:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - generalized forwarding
- instantiation, implementation in the Internet
What is internetwork

An arbitrary collection of networks interconnected to provide some sort of host-host to packet delivery service

A simple internetwork where H represents hosts and R represents routers
What is IP

- IP stands for Internet Protocol
- Key tool used today to build scalable, heterogeneous internetworks
- It runs on all the nodes in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single logical internetwork
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
The Internet network layer

host, router network layer functions:

- **Forwarding table**
- **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**
- 32 bits

**Header length (bytes)**
- 16 bits

**“Type” of data**
- 8 bits

**Max number remaining hops (decoded at each router)**
- 16 bits

**Upper layer protocol to deliver payload to**
- 16 bits

**Time to live**
- 8 bits

**Upper layer**
- 8 bits

**Header checksum**
- 16 bits

**16-bit identifier**
- 16 bits

**Flags**
- 3 bits

**Fragment offset**
- 13 bits

**Total datagram length (bytes)**
- 32 bits

**Type of service**
- 8 bits

**Fragmentation/reassembly**
- e.g. timestamp, record route taken, specify list of routers to visit.

<table>
<thead>
<tr>
<th>Field</th>
<th>bits</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version</td>
<td>4</td>
</tr>
<tr>
<td>Header length</td>
<td>4</td>
</tr>
<tr>
<td>“Type” of data</td>
<td>1</td>
</tr>
<tr>
<td>Max number remaining</td>
<td>2</td>
</tr>
<tr>
<td>Upper layer protocol</td>
<td>2</td>
</tr>
<tr>
<td>Time to live</td>
<td>2</td>
</tr>
<tr>
<td>Upper layer</td>
<td>2</td>
</tr>
<tr>
<td>Header checksum</td>
<td>2</td>
</tr>
<tr>
<td>16-bit identifier</td>
<td>16</td>
</tr>
<tr>
<td>Flags</td>
<td>3</td>
</tr>
<tr>
<td>Fragment offset</td>
<td>13</td>
</tr>
<tr>
<td>Total datagram length</td>
<td>4</td>
</tr>
<tr>
<td>Type of service</td>
<td>1</td>
</tr>
<tr>
<td>Fragmentation/reassembly</td>
<td>16</td>
</tr>
</tbody>
</table>

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

**Data**
- (variable length, typically a TCP or UDP segment)
Two key network-layer functions

**network-layer functions:** analogy: taking a trip

- **forwarding:** move packets from router’s input to appropriate router output
- **routing:** determine route taken by packets from source to destination
  - **routing algorithms**
  - **forwarding:** process of getting through single interchange
  - **routing:** process of planning trip from source to destination
Router architecture overview

- high-level view of generic router architecture:

  - **Routing processor**
  - **High-speed switching fabric**
  - **Router input ports**
  - **Router output ports**

- **Routing, management control plane** (software) operates in millisecond time frame.
- **Forwarding data plane** (hardware) operates in nanosecond timeframe.
Input port functions

- **line termination**
- **link layer protocol (receive)**
- **lookup, forwarding queueing**
- **switch fabric**

**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

- Physical layer: bit-level reception
- Data link layer: e.g., Ethernet
Input port functions

physical layer: bit-level reception

data link layer: e.g., Ethernet

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

### Forwarding Table

<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>

**Q:** but what happens if ranges don’t divide up so nicely?
Longest prefix matching

**longest prefix matching**

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

<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?

DA: 11001000 00010111 00011000 10101010  which interface?
Longest prefix matching

- we’ll see why longest prefix matching is used shortly, when we study addressing
- longest prefix matching: often performed using ternary content addressable memories (TCAMs)
  - *content addressable*: present address to TCAM: retrieve address in one clock cycle, regardless of table size
  - Cisco Catalyst: can up ~1M routing table entries in TCAM
Switching fabrics

- transfer packet from input buffer to appropriate output buffer
- switching rate: rate at which packets can be transferred from inputs to outputs
  - often measured as multiple of input/output line rate
  - N inputs: switching rate N times line rate desirable
- three types of switching fabrics

![Diagram of three types of switching fabrics: memory, bus, crossbar]
Switching via memory

**first generation routers:**

- traditional computers with switching under direct control of CPU
- packet copied to system’s memory
- speed limited by memory bandwidth (2 bus crossings per datagram)
Switching via a bus

- datagram from input port memory to output port memory via a shared bus
- \textit{bus contention:} switching speed limited by bus bandwidth
- 32 Gbps bus, Cisco 5600: sufficient speed for access and enterprise routers
Switching via interconnection network

- overcome bus bandwidth limitations
- banyan networks, crossbar, other interconnection nets initially developed to connect processors in multiprocessor
- advanced design: fragmenting datagram into fixed length cells, switch cells through the fabric.
- Cisco 12000: switches 60 Gbps through the interconnection network
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 port contention:
- Only one red datagram can be transferred.
  - *Lower red packet is blocked*

One packet time later:
- Green packet experiences HOL blocking
Output ports

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

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

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

  Priority scheduling – who gets best performance, network neutrality
Output port queueing

- Buffering when arrival rate via switch exceeds output line speed
- Queueing (delay) and loss due to output port buffer overflow!
How much buffering?

- RFC 3439 rule of thumb: average buffering equal to “typical” RTT (say 250 msec) times link capacity C
  - e.g., C = 10 Gpbs link: 2.5 Gbit buffer
- recent recommendation: with $N$ flows, buffering equal to

$$\frac{\text{RTT} \cdot C}{\sqrt{N}}$$
Scheduling mechanisms

- **scheduling**: choose next packet to send on link
- **FIFO (first in first out) scheduling**: send in order of arrival to queue
  - 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

![Diagram of packet arrivals, queue (waiting area), link (server), and packet departures]
**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)
- real world example?

![Diagram showing packet arrivals, service, and departures.](image)
Weighted Fair Queuing (WFQ):

- generalized Round Robin
- each class gets weighted amount of service in each cycle
- real-world example?

![Diagram of a network with weighted fairness queueing]

classify arrivals

link

departures

\( w_1 \)

\( w_2 \)

\( w_3 \)
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

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>IP protocol version</code></td>
<td>IP protocol version number (bytes)</td>
</tr>
<tr>
<td><code>header length</code></td>
<td>“type” of data</td>
</tr>
<tr>
<td><code>time to live</code></td>
<td>Max number remaining hops (decremented at each router)</td>
</tr>
<tr>
<td><code>upper layer</code></td>
<td>Upper layer protocol to deliver payload to</td>
</tr>
<tr>
<td><code>options</code></td>
<td>E.g. timestamp, record route taken, specify list of routers to visit.</td>
</tr>
<tr>
<td><code>data</code></td>
<td>(variable length, typically a TCP or UDP segment)</td>
</tr>
</tbody>
</table>

**How much overhead?**

- 20 bytes of TCP
- 20 bytes of IP
- 40 bytes + app layer overhead
IP fragmentation, reassembly

- network links have MTU (max. transfer size) - 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

1480 bytes in data field

offset = 1480/8

---

One large datagram becomes several smaller datagrams

<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>4000</td>
<td>x</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>x</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1500</td>
<td>x</td>
<td>1</td>
<td>185</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>length</th>
<th>ID</th>
<th>fragflag</th>
<th>offset</th>
</tr>
</thead>
<tbody>
<tr>
<td>1040</td>
<td>x</td>
<td>0</td>
<td>370</td>
</tr>
</tbody>
</table>
IP Fragmentation and Reassembly

IP datagrams traversing the sequence of physical networks
IP Fragmentation and Reassembly

(a) Unfragmented packet; (b) fragmented packets.

Header fields used in IP fragmentation. (a) Unfragmented packet; (b) fragmented packets.
**IP addressing: introduction**

- **IP address:** 32-bit identifier for host, router interface
- **Interface:** connection between host/router and physical link
  - router’s typically have multiple interfaces
  - host typically has one or two interfaces (e.g., wired Ethernet, wireless 802.11)
- **IP addresses associated with each interface**

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

```
223 1 1 1 1
```
**Q:** how are interfaces actually connected?  
**A:** we’ll learn about that later.

**A:** wired Ethernet interfaces connected by Ethernet switches

**For now:** don’t need to worry about how one interface is connected to another (with no intervening router)

**A:** 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*

**network consisting of 3 subnets**
**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**

```plaintext
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
223.1.2.1
223.1.2.2
223.1.2.9
223.1.3.1
223.1.3.2
223.1.3.27

subnet
```
Global Addresses

- **Properties**
  - globally unique
  - hierarchical: network + host
  - 4 Billion IP address, half are A type, ¼ is B type, and 1/8 is C type

- **Format**

<table>
<thead>
<tr>
<th>Class</th>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>7</td>
<td>14</td>
<td>21</td>
</tr>
<tr>
<td>Network</td>
<td>Host</td>
<td>Network</td>
<td>Host</td>
</tr>
<tr>
<td>0</td>
<td></td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td>1 1 0</td>
<td></td>
</tr>
</tbody>
</table>

- **Dot notation**
  - 10.3.2.4
  - 128.96.33.81
  - 192.12.69.77
Subnetting

- Add another level to address/routing hierarchy: subnet
- *Subnet masks* define variable partition of host part of class A and B addresses
- Subnets visible only within site

```
  Network number  |  Host number
   ---------------|-------------
        Class B address

  111111111111111111111111 | 00000000
  Subnet mask (255.255.255.0)

  Network number  |  Subnet ID  |  Host ID
   ---------------|-------------|-------------
                  Subnetted address
```
Subnetting

Forwarding Table at Router R1

<table>
<thead>
<tr>
<th>SubnetNumber</th>
<th>SubnetMask</th>
<th>NextHop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.96.34.0</td>
<td>255.255.255.128</td>
<td>Interface 0</td>
</tr>
<tr>
<td>128.96.34.128</td>
<td>255.255.255.128</td>
<td>Interface 1</td>
</tr>
<tr>
<td>128.96.33.0</td>
<td>255.255.255.0</td>
<td>R2</td>
</tr>
</tbody>
</table>
Subnetting

Forwarding Algorithm

D = destination IP address
for each entry < SubnetNum, SubnetMask, NextHop>
    D1 = SubnetMask & D
    if D1 = SubnetNum
        if NextHop is an interface
            deliver datagram directly to destination
        else
            deliver datagram to NextHop (a router)
Subnets

how many?
Classless Addressing

Classless Inter-Domain Routing

- A technique that addresses two scaling concerns in the Internet
  - The growth of backbone routing table as more and more network numbers need to be stored in them
  - Potential exhaustion of the 32-bit address space

- Address assignment efficiency
  - Arises because of the IP address structure with class A, B, and C addresses
  - Forces us to hand out network address space in fixed-size chunks of three very different sizes
    - A network with two hosts needs a class C address
      - Address assignment efficiency = 2/255 = 0.78
    - A network with 256 hosts needs a class B address
      - Address assignment efficiency = 256/65535 = 0.39
Classless Addressing

- Exhaustion of IP address space centers on exhaustion of the class B network numbers

- Solution
  - Say “NO” to any Autonomous System (AS) that requests a class B address unless they can show a need for something close to 64K addresses
  - Instead give them an appropriate number of class C addresses
  - For any AS with at least 256 hosts, we can guarantee an address space utilization of at least 50%

- What is the problem with this solution?
Classless Addressing

- Problem with this solution
  - Excessive storage requirement at the routers.
- If a single AS has, say 16 class C network numbers assigned to it,
  - Every Internet backbone router needs 16 entries in its routing tables for that AS
  - This is true, even if the path to every one of these networks is the same
- If we had assigned a class B address to the AS
  - The same routing information can be stored in one entry
  - Efficiency = $16 \times 255 / 65,536 = 6.2\%$
Classless Addressing

- CIDR tries to balance the desire to minimize the number of routes that a router needs to know against the need to hand out addresses efficiently.

- CIDR uses aggregate routes
  - Uses a single entry in the forwarding table to tell the router how to reach a lot of different networks
  - Breaks the rigid boundaries between address classes
Consider an AS with 16 class C network numbers.

Instead of handing out 16 addresses at random, hand out a block of contiguous class C addresses.

Suppose we assign the class C network numbers from 192.4.16 through 192.4.31.

Observe that top 20 bits of all the addresses in this range are the same (11000000 00000100 0001).

- We have created a 20-bit network number (which is in between class B network number and class C number).

Requires to hand out blocks of class C addresses that share a common prefix.
Classless Addressing

- Requires to hand out blocks of class C addresses that share a common prefix.
- The convention is to place a /X after the prefix where X is the prefix length in bits.
- For example, the 20-bit prefix for all the networks 192.4.16 through 192.4.31 is represented as 192.4.16/20.
- By contrast, if we wanted to represent a single class C network number, which is 24 bits long, we would write it 192.4.16/24.
Classless Addressing

- How do the routing protocols handle this classless addresses
  - It must understand that the network number may be of any length
- Represent network number with a single pair
  
  <length, value>

- All routers must understand CIDR addressing
Classless Addressing

Route aggregation with CIDR
IP Forwarding Revisited

- IP forwarding mechanism assumes that it can find the network number in a packet and then look up that number in the forwarding table.

- We need to change this assumption in case of CIDR.

- CIDR means that prefixes may be of any length, from 2 to 32 bits.
IP Forwarding Revisited

- It is also possible to have prefixes in the forwarding tables that overlap
  - Some addresses may match more than one prefix

- For example, we might find both 171.69 (a 16 bit prefix) and 171.69.10 (a 24 bit prefix) in the forwarding table of a single router

- A packet destined to 171.69.10.5 clearly matches both prefixes.
  - The rule is based on the principle of “longest match”
    - 171.69.10 in this case

- A packet destined to 171.69.20.5 would match 171.69 and not 171.69.10
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

**Example: 200.23.16.0/23**

```
11001000 00010111 00010000 00000000
```

Subnet part: 11001000
Host part: 00010111 00010000 00000000

Netmask: 255.255.254.0
**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 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 overview:**

- host broadcasts “DHCP discover” msg [optional]
- DHCP server responds with “DHCP offer” msg [optional]
- host requests IP address: “DHCP request” msg
- DHCP server sends address: “DHCP ack” msg
DHCP client-server scenario

DHCP server

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

DHCP server: 223.1.2.5

DHCP discover

Broadcast: is there a DHCP server out there?

DHCP offer

Broadcast: I’m a DHCP server! Here’s an IP address you can use

DHCP request

Broadcast: OK. I’ll take that IP address!

DHCP ACK

Broadcast: OK. You’ve got that IP address!
DHCP: more than IP addresses

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

- address of first-hop router for client
- name and IP address of DNS server
- network mask (indicating network versus host portion of address)
DHCP: example

- connecting laptop needs its IP address, addr of first-hop router, addr of DNS server: use DHCP

- DHCP request encapsulated in UDP, encapsulated in IP, encapsulated in 802.1 Ethernet

- Ethernet frame broadcast (dest: FFFFFFFF) on LAN, received at router running DHCP server

- Ethernet demuxed to IP demuxed, UDP demuxed to DHCP
DHCP: example

- DCP server formulates DHCP ACK containing client’s IP address, IP address of first-hop router for client, name & IP address of DNS server
  - encapsulation of DHCP server, frame forwarded to client, demuxing up to DHCP at client
  - client now knows its IP address, name and IP address of DNS server, IP address of its first-hop router
**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>Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Organization 0</td>
<td>11001000 00010111 00010000 00000000 200.23.16.0/23</td>
</tr>
<tr>
<td>Organization 1</td>
<td>11001000 00010111 00010010 00000000 200.23.18.0/23</td>
</tr>
<tr>
<td>Organization 2</td>
<td>11001000 00010111 00010100 00000000 200.23.20.0/23</td>
</tr>
<tr>
<td>...</td>
<td>.....</td>
</tr>
<tr>
<td>Organization 7</td>
<td>11001000 00010111 00011110 00000000 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/
  • allocates addresses
  • manages DNS
  • assigns domain names, resolves disputes
NAT: network address translation

All 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**

*S 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**

*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

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>
</tr>
</thead>
<tbody>
<tr>
<td>WAN side addr</td>
</tr>
<tr>
<td>138.76.29.7, 5001</td>
</tr>
<tr>
<td>……</td>
</tr>
</tbody>
</table>

1: host 10.0.0.1 sends datagram to 128.119.40.186, 80

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

- 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
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, e.g., P2P applications
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

**priority:** identify priority among datagrams in flow

**flow Label:** identify datagrams in same “flow.”
(concept of “flow” not well defined).

**next header:** identify upper layer protocol for data
Other changes from IPv4

- **checksum**: removed entirely to reduce processing time at each hop
- **options**: allowed, but outside of header, indicated by “Next Header” field
- **ICMPv6**: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
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

**logical view:**

A (IPv6) — B (IPv6) — **IPv4 tunnel connecting IPv6 routers** — E (IPv6) — F (IPv6)

**physical view:**

A (IPv6) — B (IPv6) — C (IPv4) — D (IPv4) — E (IPv6) — F (IPv6)
Tunneling

**logical view:**

- **A** (IPv6) to **B** (IPv6) via **IPv4 tunnel connecting IPv6 routers** to **E** (IPv6) to **F** (IPv6)

**physical view:**

- **A** (IPv6) to **B** (IPv6) via **C** (IPv4) to **D** (IPv4) to **E** (IPv6) to **F** (IPv6)

  - Flow: X
  - src: A
  - dest: F
  - data

  - src: B
  - dest: E
  - Flow: X
  - Src: A
  - Dest: F
  - data

  - src: B
  - dest: E
  - Flow: X
  - Src: A
  - Dest: F
  - data

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

---

**IPv4 tunnel connecting IPv6 routers**
IPv6: adoption

- Google: 11% of clients access services via IPv6
- NIST: 1/3 of all US government domains are IPv6 capable

- Long (long!) time for deployment, use
  - More than 20 years and counting! [IETF initiated standardization of IPv6 in 1994]
  - think of application-level changes in last 20 years: WWW, Facebook, streaming media, Skype, ...

Internet Control Message Protocol (ICMP)

- Defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully
  - Destination host unreachable due to link/node failure
  - Reassembly process failed
  - TTL had reached 0 (so datagrams don't cycle forever)
  - IP header checksum failed

- ICMP-Redirect
  - From router to a source host
  - With a better route information
Internet Control Message Protocol (ICMP)

- Defines a collection of error messages that are sent back to the source host whenever a router or host is unable to process an IP datagram successfully
  - Destination host unreachable due to link/node failure
  - Reassembly process failed
  - TTL had reached 0 (so datagrams don't cycle forever)
  - IP header checksum failed

- ICMP-Redirect
  - From router to a source host
  - With a better route information
ICMP: internet control message protocol

- used by hosts & routers to communicate network-level information
  - error reporting: unreachable host, network, port, protocol
  - echo request/reply (used by ping)
- network-layer “above” IP:
  - ICMP msgs carried in IP datagrams
- ICMP message: type, code plus first 8 bytes of IP datagram causing error

<table>
<thead>
<tr>
<th>Type</th>
<th>Code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>echo reply (ping)</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>dest. network unreachable</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>dest host unreachable</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>dest protocol unreachable</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>dest port unreachable</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>dest network unknown</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>dest host unknown</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>source quench (congestion control - not used)</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>echo request (ping)</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>route advertisement</td>
</tr>
<tr>
<td>10</td>
<td>0</td>
<td>router discovery</td>
</tr>
<tr>
<td>11</td>
<td>0</td>
<td>TTL expired</td>
</tr>
<tr>
<td>12</td>
<td>0</td>
<td>bad IP header</td>
</tr>
</tbody>
</table>
Traceroute and ICMP

- Source sends series of UDP segments to destination
  - First set has TTL = 1
  - Second set has TTL = 2, etc.
  - Unlikely port number
- When datagram in nth set arrives to nth router:
  - Router discards datagram and sends source ICMP message (type 11, code 0)
  - ICMP message includes name of router & IP address
  - When ICMP message arrives, source records RTTs

Stopping criteria:
- UDP segment eventually arrives at destination host
- Destination returns ICMP “port unreachable” message (type 3, code 3)
- Source stops
Routing protocols

*Routing protocol 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') = cost of link (x,x')

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

cost could always be 1, or

inversely related to bandwidth, or

inversely related to

congestion

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

*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
  - It does not deal with node or link failures
  - It does not consider the addition of new nodes or links
  - It implies that edge costs cannot change

dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
Link State Routing Algorithm
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 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 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
14 shortest path cost to \( w \) plus cost from \( w \) to \( v \ */
15 **until all nodes in \( N' \)**
### 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>p(v)</td>
<td>p(w)</td>
<td>p(x)</td>
<td>p(y)</td>
<td>p(z)</td>
<td></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>11,w</td>
<td>14,x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uwxv</td>
<td></td>
<td>10,v</td>
<td>14,x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uwxvy</td>
<td></td>
<td></td>
<td></td>
<td>12,y</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uwxvz</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).
### Dijkstra’s algorithm: another example

<table>
<thead>
<tr>
<th>Step</th>
<th>N'</th>
<th>(D(v),p(v))</th>
<th>(D(w),p(w))</th>
<th>(D(x),p(x))</th>
<th>(D(y),p(y))</th>
<th>(D(z),p(z))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>u</td>
<td>2,u</td>
<td>5,u</td>
<td>1,u</td>
<td>(\infty)</td>
<td>(\infty)</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>(\infty)</td>
<td>4,y</td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>3,y</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td>3,y</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
<tr>
<td>5</td>
<td>uxyvz</td>
<td>3,y</td>
<td>3,y</td>
<td>4,y</td>
<td>4,y</td>
<td>4,y</td>
</tr>
</tbody>
</table>

The network with edge weights and the steps of Dijkstra’s algorithm are shown in the diagram.
Dijkstra’s algorithm: example (2)

resulting shortest-path tree from u:

resulting forwarding table in u:

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

**algorithm complexity:** \( n \) nodes

- each iteration: need to check all nodes, \( w \), not in \( N \)
- \( n(n+1)/2 \) comparisons: \( O(n^2) \)
- more efficient implementations possible: \( O(n \log n) \)

**oscillations possible:**

- e.g., support link cost equals amount of carried traffic:

Execute the algorithm at same period, but start at different times.
**Link State Routing**

**Strategy:** Send to all nodes (not just neighbors) information about directly connected links (not entire routing table).

- **Link State Packet (LSP)**
  - id of the node that created the LSP
  - cost of link to each directly connected neighbor
  - sequence number (SEQNO)
  - time-to-live (TTL) for this packet

- **Reliable Flooding**
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically (timer); increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP; discard when TTL=0
Flooding of link-state packets.
(a) LSP arrives at node X;
(b) X floods LSP to A and C;
(c) A and C flood LSP to B (but not X);
(d) flooding is complete
Distance Vector Routing Algorithm
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 from neighbor \( v \) to destination \( y \)
- \( d_v(y) \) is the cost to neighbor \( v \)
- \( \min \) taken over all neighbors \( v \) of \( x \)
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

- \( D_x(y) \) = estimate of least cost from \( x \) to \( y \)
  - \( x \) maintains distance vector \( D_x = [D_x(y) : y \in N] \)

- node \( x \):
  - knows cost to each neighbor \( v \): \( c(x,v) \)
  - 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:

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

- under minor, natural conditions, the estimate \( D_x(y) \) converge to the actual least cost \( d_x(y) \)
Distance vector algorithm

Iterative, asynchronous: each local iteration caused by:
- local link cost change
- DV update message from neighbor

Distributed:
- each node notifies neighbors *only* when its DV changes
  - neighbors then notify their neighbors if necessary

Each node:

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

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

\[
D_x(z) = \min\{c(x,y) + D_y(z), c(x,z) + D_z(z)\} \\
= \min\{2+1, 7+0\} = 3
\]
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
- *bad news travels slow* - “count to infinity” problem!
- 44 iterations before algorithm stabilizes

![Diagram of network with nodes X, Y, and Z, showing link costs and flow of information.]

**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?
Routing Information Protocol (RIP)

Example Network running RIP

RIPv2 Packet Format

An example Distance Vector Protocol
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
  - may have oscillations
- **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 propagates thru network
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
- gateway router: at “edge” of its own AS, has link(s) to router(s) in other AS’es

inter-AS routing

routing among 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
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!*
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)
  - link state: for each attached link
- *IS-IS routing* protocol: nearly identical to OSPF
Open Shortest Path First (OSPF)

5 different types

OSPF Header Format

<table>
<thead>
<tr>
<th>Version</th>
<th>Type</th>
<th>Message length</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SourceAddr

Areald

Checksum

Authentication type

Authentication

OSPF Link State Advertisement

<table>
<thead>
<tr>
<th>LS Age</th>
<th>Options</th>
<th>Type = 1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Link-state ID

Advertising router

LS sequence number

LS checksum

Length

0 Flags 0 Number of links

Link ID

Link data

Link type Num_TOS Metric

Optional TOS information

More links
OSPF “advanced” features

- **security:** all OSPF messages authenticated (to prevent malicious intrusion)
- **multiple same-cost paths** allowed (only one path in RIP)
- for each link, **multiple cost metrics** for different TOS (e.g., satellite link cost (more delay involved) set low for best effort ToS; high for real-time ToS)
- integrated uni- and **multi-cast** support:
  - Multicast OSPF (MOSPF) uses same topology data base as OSPF
- **hierarchical** OSPF in large domains.
Hierarchical OSPF

- **Boundary router**
- **Backbone router**
- **Backbone area**
- **Area 1**
- **Area 2**
- **Area 3**
- **Internal routers**
- **Area border routers**
Hierarchical OSPF

- **two-level hierarchy**: local area, backbone.
  - link-state advertisements only in area
  - each nodes has detailed area topology; only know direction (shortest path) to nets in other areas.
- **area border routers**: “summarize” distances to nets in own area, advertise to other Area Border routers.
- **backbone routers**: run OSPF routing limited to backbone.
- **boundary routers**: connect to other AS’es.
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:
  - **eBGP:** obtain subnet reachability information from neighboring ASes
  - **iBGP:** propagate reachability information to all AS-internal routers.
  - determine “good” routes to other networks based on reachability information and *policy*

- allows subnet to advertise its existence to rest of Internet:
  “*I am here*”
eBGP, iBGP connections

gateway routers run both eBGP and iBGP protocols
**BGP basics**

- **BGP session**: two BGP routers ("peers") exchange BGP messages over semi-permanent TCP connection (port 179):
  - advertising *paths* to different destination network prefixes (BGP is a "path vector" protocol)

  ![Diagram of BGP basics](image)

  ➢ when AS3 gateway router 3a advertises path *AS3,X* to AS2 gateway router 2c:
    - **AS3 promises** to AS2 it will forward datagrams towards X
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
- **Policy-based routing**:
  - gateway receiving route advertisement uses *import policy* to accept/decline path (e.g., never route through AS Y).
  - AS policy also determines whether to *advertise* path to other other neighboring ASes
AS2 router 2c receives path advertisement \textit{AS3,X} (via eBGP) from AS3 router 3a

Based on AS2 policy, AS2 router 2c accepts path \textit{AS3,X}, propagates (via iBGP) to all AS2 routers

Based on AS2 policy, AS2 router 2a advertises (via eBGP) path \textit{AS2,AS3,X} to AS1 router 1c
AS1 gateway router 1c learns path $AS2, AS3, X$ from 2a

---

As1 gateway router may learn about **multiple** paths to destination:

- AS1 gateway router 1c learns path $AS3, X$ from 3a
- Based on policy, AS1 gateway router 1c chooses path $AS3, X$, and advertises path within AS1 via iBGP
BGP messages

- BGP messages exchanged between peers over TCP connection
- BGP messages:
  - **OPEN**: opens TCP connection to remote BGP peer and authenticates sending BGP peer
  - **UPDATE**: advertises new path (or withdraws old)
  - **KEEPALIVE**: keeps connection alive in absence of UPDATES; also ACKs OPEN request
  - **NOTIFICATION**: reports errors in previous msg; also used to close connection
BGP, OSPF, forwarding table entries

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”

Q: how does router set forwarding table entry to distant prefix?

- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
BGP, OSPF, forwarding table entries

- recall: 1a, 1b, 1c learn about dest X via iBGP from 1c: “path to X goes through 1c”
- Q: how does router set forwarding table entry to distant prefix?

<table>
<thead>
<tr>
<th>dest</th>
<th>interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>2</td>
</tr>
</tbody>
</table>

- 1d: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 1
- 1a: OSPF intra-domain routing: to get to 1c, forward over outgoing local interface 2
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
  2. shortest AS-PATH
  3. closest NEXT-HOP router: hot potato routing
  4. additional criteria
Hot Potato Routing

2d learns (via iBGP) it can route to X via 2a or 2c

*hot potato routing*: choose local gateway that has least intra-domain cost (e.g., 2d chooses 2a, even though more AS hops to X): don’t worry about inter-domain cost!
BGP: achieving policy via advertisements

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- 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
BGP: achieving policy via advertisements

Suppose an ISP only wants to route traffic to/from its customer networks (does not want to carry transit traffic between other ISPs)

- 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
Why different Intra-, 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
Summary

- **IP**
  - Datagram, Fragmentation, IPv4, IPv6

- **Router Architecture**

- **Routing**
  - Link State
  - Distance Vector
  - Intra- and Inter-AS Routing