A note on the use of these ppt slides:

We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a lot of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) in substantially unaltered form, that you mention their source (after all, we'd like people to use our book!)
- If you post any slides in substantially unaltered form on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

Thanks and enjoy! JFK/KWR

All material copyright 1996-2005
J.F Kurose and K.W. Ross, All Rights Reserved
Chapter 4: Network Layer

Chapter goals:
- understand principles behind network layer services:
  - network layer service models
  - forwarding versus routing
  - how a router works
  - routing (path selection)
  - dealing with scale
  - advanced topics: IPv6, mobility
- instantiation, implementation in the Internet
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What's inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
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
Network layer functions

- Transport packet from sending to receiving hosts
- Network layer protocols in every host, router

- **Addressing**
  - flat vs. hierarchical
    - Routing table size?
  - global vs. local
    - NAT
  - variable vs. fixed length
    - processing cost
    - Header size
    - Address flexibility

- **Delivery semantics:**
  - Unicast, multicast (IPv4)
  - Anycast (IPv6)
  - Broadcast
  - In-order (ATM)
  - Any-order (IP)
Network layer functions

- Transport packet from sending to receiving hosts
- Network layer protocols in every host, router

- Security
  - secrecy, integrity, authenticity
- Fragmentation
  - break-up packets based on data-link layer properties
- Quality-of-service
  - provide predictable performance
- Routing
  - path selection and packet forwarding
- Demux to upper layer
  - next protocol
  - Can be either transport or network (tunneling)
- Connection setup
  - ATM, X.25, Frame-relay
  - Host-to-host network layer connection vs. process to process transport layer
Network service model

Combining the functions into a particular network

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

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 (jitter)
# Network layer service models:

<table>
<thead>
<tr>
<th>Network Architecture</th>
<th>Service Model</th>
<th>Guarantees?</th>
<th>Congestion feedback</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bandwidth</td>
<td>Loss</td>
</tr>
<tr>
<td>Internet</td>
<td>best effort</td>
<td>none</td>
<td>no</td>
</tr>
<tr>
<td>ATM</td>
<td>CBR</td>
<td>constant rate</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>VBR</td>
<td>guaranteed rate</td>
<td>yes</td>
</tr>
<tr>
<td>ATM</td>
<td>ABR</td>
<td>guaranteed minimum</td>
<td>no</td>
</tr>
<tr>
<td>ATM</td>
<td>UBR</td>
<td>none</td>
<td>no</td>
</tr>
</tbody>
</table>
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Network layer connection and connection-less service

- Datagram network provides network-layer connectionless service
- VC network provides network-layer connection service

- Analogous to the transport-layer services, but:
  - **Service**: host-to-host
  - **No choice**: network provides one or the other
  - **Implementation**: in the core
Connection-oriented virtual circuits

- Phone circuit abstraction (ATM, phone network)
  - Model
    - call setup and signaling for each call before data can flow
    - guaranteed performance during call
    - call teardown and signaling to remove call
  - Network support
    - each packet carries circuit identifier (not destination host ID)
    - every router on source-dest path maintains “state” for each passing circuit
    - link, router resources (bandwidth, buffers) allocated to VC to guarantee circuit-like performance
Connectionless datagram service

- Postal service abstraction (Internet)
  - Model
    - no call setup or teardown at network layer
    - no service guarantees
  - Network support
    - no state within network on end-to-end connections
    - packets forwarded based on destination host ID
    - packets between same source-dest pair may take different paths

1. Send data
2. Receive data
Datagram or VC network: why?

**Internet**
- data exchange among computers
  - “elastic” service, no strict timing req.
- “smart” end systems (computers)
  - can adapt, perform control, error recovery
  - simple inside network, complexity at “edge”
- many link types
  - different characteristics
  - uniform service difficult

**ATM**
- evolved from telephony
- human conversation:
  - strict timing, reliability requirements
  - need for guaranteed service
- “dumb” end systems
  - telephones
  - complexity inside network
Best of both worlds?

- Adding circuits to the Internet
  - Intserv, Diffserv (at the end of course if time permits)
  - Chapter 6 in book
- Support both modes from the start?
  - ATM
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
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"

Network layer

Transport layer: TCP, UDP

Link layer

physical layer
Chapter 4: Network Layer

4.1 Introduction
4.2 Virtual circuit and datagram networks
4.3 What’s inside a router
4.4 IP: Internet Protocol
   - Datagram format
   - IPv4 addressing
   - ICMP
   - IPv6
4.5 Routing algorithms
   - Link state
   - Distance Vector
   - Hierarchical routing
4.6 Routing in the Internet
   - RIP
   - OSPF
   - BGP
4.7 Broadcast and multicast routing
How is IP Design Standardized?

- **IETF**
  - Voluntary organization
  - Meeting every 4 months
  - Working groups and email discussions

- “We reject kings, presidents, and voting; we believe in rough consensus and running code” (Dave Clark 1992)
  - Need 2 independent, interoperable implementations for standard

- **IRTF**
  - End2End
  - Reliable Multicast, etc.
## IP datagram format

<table>
<thead>
<tr>
<th>Field</th>
<th>Length/Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP protocol version number</td>
<td>8 bits</td>
</tr>
<tr>
<td>header length (bytes)</td>
<td>4 bits</td>
</tr>
<tr>
<td>“type” of data</td>
<td>8 bits</td>
</tr>
<tr>
<td>max number of remaining hops</td>
<td>16 bits</td>
</tr>
<tr>
<td>(decremented at each router)</td>
<td></td>
</tr>
<tr>
<td>upper layer protocol</td>
<td></td>
</tr>
<tr>
<td>to deliver payload to</td>
<td></td>
</tr>
<tr>
<td>total datagram length (bytes)</td>
<td></td>
</tr>
<tr>
<td>for fragmentation/reassembly</td>
<td></td>
</tr>
<tr>
<td>upper layer protocol</td>
<td></td>
</tr>
<tr>
<td>Options (if any)</td>
<td></td>
</tr>
<tr>
<td>data</td>
<td></td>
</tr>
<tr>
<td>(variable length, typically a</td>
<td></td>
</tr>
<tr>
<td>TCP or UDP segment)</td>
<td></td>
</tr>
</tbody>
</table>

### 32 bit source IP address

### 32 bit destination IP address

### 16-bit identifier

### Fragment offset

### Time to live

### Upper layer

### Internet checksum

### Data (variable length, typically a TCP or UDP segment)

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

- **Version**
  - Currently at 4, next version 6

- **Header length**
  - Length of header (20 bytes plus options)

- **Type of Service**
  - Typically ignored
  - Values
    - 3 bits of precedence
    - 1 bit of delay requirements
    - 1 bit of throughput requirements
    - 1 bit of reliability requirements
  - Replaced by DiffServ and ECN

- **Length**
  - Length of IP fragment (payload)
IP header (cont)

- Identification
  - To match up with other fragments

- Flags
  - Don't fragment flag
  - More fragments flag

- Fragment offset
  - Where this fragment lies in entire IP datagram
  - Measured in 8 octet units (11 bit field)
IP header (cont)

- Time to live
  - Ensure packets exit the network

- Protocol
  - Demultiplexing to higher layer protocols

- Header checksum
  - Ensures some degree of header integrity
  - Relatively weak - 16 bit

- Source IP, Destination IP (32 bit addresses)

- Options
  - E.g. Source routing, record route, etc.
  - Performance issues
    - Poorly supported
IP quality of service

- IP originally had “type-of-service” (TOS) field to eventually support quality
  - Not used, ignored by most routers
- Then came int-serv (integrated services) and RSVP signalling
  - Per-flow quality of service through end-to-end support
    - Setup and match flows on connection ID
    - Per-flow signaling
    - Per-flow network resource allocation (*FQ, *RR scheduling algorithms)
IP quality of service

- **RSVP**
  - Provides end-to-end signaling to network elements
  - General purpose protocol for signaling information
  - Not used now on a per-flow basis to support int-serv, but being reused for diff-serv.

- **int-serv**
  - Defines service model (guaranteed, controlled-load)
  - Dozens of scheduling algorithms to support these services
    - WFQ, \(W^2FQ\), STFQ, Virtual Clock, DRR, etc.
    - If this class was being given 5 years ago....
IP quality of service

Why did RSVP, int-serv fail?

- Complexity
  - Scheduling
  - Routing
  - Per-flow signaling overhead

- Lack of scalability
  - Per-flow state
  - Route pinning

- Economics
  - Providers with no incentive to deploy
  - SLA, end-to-end billing issues

- QoS a weak-link property
  - Requires every device on an end-to-end basis to support flow
IP quality of service

- Now it’s diff-serv...
  - Use the “type-of-service” bits as a priority marking
    - http://www.rfc-editor.org/rfc/rfc2474.txt
  - Core network relatively stateless
  - AF
    - Assured forwarding (drop precedence)
  - EF
    - Expedited forwarding (strict priority handling)
IP Fragmentation & Reassembly

- Network links have MTU (max. transfer size) - largest possible link-level frame.
  - Different link types, different MTUs
- Large IP datagram (can be 64KB) “fragmented” within network
  - One datagram becomes several datagrams
  - IP header on each fragment
  - Bits used to identify, order fragments
IP Fragmentation & Reassembly

- Where to do reassembly?
  - End nodes
    - avoids unnecessary work
  - Dangerous to do at intermediate nodes
    - Buffer space
    - Must assume single path through network
    - May be re-fragmented later on in the route again

- fragmentation:
  in: one large datagram
  out: 3 smaller datagrams

- reassembly
IP Fragmentation and Reassembly

Example
- 4000 byte datagram
- MTU = 1500 bytes

One large datagram becomes several smaller datagrams

- ID = x, offset = 0, fragflag = 0, length = 4000
- ID = x, offset = 0, fragflag = 1, length = 1500
- ID = x, offset = 185, fragflag = 1, length = 1500
- ID = x, offset = 370, fragflag = 0, length = 1040

1480 bytes in data field

Offset = 1480/8

Network Layer 4-29
Fragmentation is Harmful

- Uses resources poorly
  - Forwarding costs per packet
  - Best if we can send large chunks of data
  - Worst case: packet just bigger than MTU

- Poor end-to-end performance
  - Loss of a fragment makes other fragments useless

- Reassembly is hard
  - Buffering constraints
Fragmentation

References


Fragmentation

- Path MTU Discovery
  - Remove fragmentation from the network
  - Mandatory in IPv6
    - Network layer does no fragmentation
  - Hosts dynamically discover minimum MTU of path
    - Algorithm:
      - Initialize MTU to MTU for first hop
      - Send datagrams with Don’t Fragment bit set
      - If ICMP “pkt too big” msg, decrease MTU
  - What happens if path changes?
    - Periodically (>5mins, or >1min after previous increase), increase MTU
  - Some routers will return proper MTU
IP demux to upper layer

Protocol type field

- 1 = ICMP
- 2 = IGMP
- 3 = GGP
- 4 = IP in IP
- 6 = TCP
- 8 = EGP
- 9 = IGP
- 17 = UDP
- 29 = ISO-TP4
- 80 = ISO-IP
- 88 = IGRP
- 89 = OSPFIGP
- 94 = IPIP
IP error detection

- IP checksum
  - IP has a header checksum, leaves data integrity to TCP/UDP
  - Catch errors within router or bridge that are not detected by link layer
  - Incrementally updated as routers change fields
IP delivery semantics

- The waist of the hourglass
  - Unreliable datagram service
  - Out-of-order delivery possible
  - Compare to ATM and phone network...

- Unicast mostly
  - IP broadcast not forwarded
  - IP multicast supported, but not widely used
IP security

- IP originally had no provisions for security
- IPsec
  - Retrofit IP network layer with encryption and authentication
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
IP Addressing

- **IP address**: fixed-length, 32-bit identifier for host, router, interface
  - semantics getting fuzzy, though (more later)

- **interface**: connection between host, router and physical link
  - router's typically have multiple interfaces
  - host may have multiple interfaces
  - IP addresses associated with interface, not host, router

![Diagram of IP addresses and interfaces]
IP Addressing

- **IP address:**
  - network part (high order bits)
  - host part (low order bits)

- **What’s a network?**
  - all device interfaces with same network part of IP address
  - all interfaces that can physically reach each other without intervening router

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

How to find the networks (subnets)?

- Detach each interface from router, host
- Create “islands of isolated networks
- Each isolated network is called a subnet

Subnet mask: /24
Subnets

How many?
Classful IP Addressing (1981)

- Total IP address size: 4 billion
  - Initially one large class (8-bit network, 24-bit host)
  - Classful addressing for smaller networks (LANs)
    - Class A: 128 networks, 16M hosts
    - Class B: 16K networks, 64K hosts
    - Class C: 2M networks, 256 hosts

<table>
<thead>
<tr>
<th>High Order Bits</th>
<th>Format</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>7 bits of net, 24 bits of host</td>
<td>A</td>
</tr>
<tr>
<td>10</td>
<td>14 bits of net, 16 bits of host</td>
<td>B</td>
</tr>
<tr>
<td>110</td>
<td>21 bits of net, 8 bits of host</td>
<td>C</td>
</tr>
</tbody>
</table>
**IP address classes**

- **Class A**: 0 Network ID 8 Host ID
  - 1.0.0.0 to 127.255.255.255

- **Class B**: 1 0 Network ID 16 Host ID
  - 128.0.0.0 to 191.255.255.255

- **Class C**: 1 1 0 Network ID 24 Host ID
  - 192.0.0.0 to 223.255.255.255

- **Class D**: 11 10 Multicast Addresses
  - 224.0.0.0 to 239.255.255.255

- **Class E**: 11 11 Reserved for experiments
Special IP Addresses

- Private addresses
  - Class A: 10.0.0.0 - 10.255.255.255 (10/8 prefix)
  - Class B: 172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
  - Class C: 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

- 127.0.0.1: local host (a.k.a. the loopback address)

- 255.255.255.255
  - IP broadcast to local hardware that must not be forwarded
  - Same as network broadcast if no subnetting
    - IP of network broadcast=NetworkID+(all 1's for HostID)

- 0.0.0.0
  - IP address of unassigned host (BOOTP, ARP, DHCP)
  - Default route advertisement
IP Addressing Problem #1 (1984)

- Inefficient use of address space
  - Class A (rarely given out, not many of them given out by IANA)
  - Class B = 64k hosts
    - Very few LANs have close to 64K hosts
    - Electrical/LAN limitations, performance or administrative reasons
    - e.g., class B net allocated enough addresses for 64K hosts, even if only 2K hosts in that network
  - Need simple/address-efficient way to get multiple “networks”
    - Reduce the total number of addresses that are assigned, but not used

- Subnet addressing
  - Split up single large network address ranges into multiple smaller ones (subnet)
Subnetting

- Variable length subnet masks
  - Subnet a class B address space into several chunks

![Diagram showing subnetting](image.png)
**Subnetting Example**

- Assume an organization was assigned address 150.100

- Assume < 100 hosts per subnet
  - How many host bits do we need? Seven
  - What is the network mask?
    - 11111111 11111111 11111111 10000000
    - 255.255.255.128
IP Address Problem #2 (1991)

- **Address space depletion**
  - In danger of running out of classes A and B
  - **Class A**
    - very few in number, IANA frugal in giving them out
  - **Class B**
    - subnetting only applied to new allocations of class B
    - existing class B networks sparsely populated
    - people refuse to give it back
  - **Class C**
    - plenty available, but too small for most domains
    - giving out multiple class C to a domain explodes # of routes

- **Supernetting**
  - Assign multiple consecutive class C blocks as one block
CIDR

- Evolved into Classless Inter-Domain Routing (CIDR)
IP addressing: CIDR

- Original classful addressing
  - Use class structure (A, B, C) to determine network ID for route lookup

- CIDR: Classless InterDomain Routing
  - Do not use classes to determine network ID
  - network portion of address of arbitrary length
  - address format: a.b.c.d/x, where x is # bits in network portion of address

```
11001000  00010111 00010000  00000000
```

200.23.16.0/23
CIDR

- Assign any range of addresses to network
  - Use common part of address as network number
  - e.g., addresses 192.4.16.* to 192.4.31.* have the first 20 bits in common. Thus, we use this as the network number
  - netmask is /20, /xx is valid for almost any xx
  - 192.4.16.0/20

- Enables more efficient usage of address space (and router tables)

- More on how this impacts routing later....
IP addresses: how to get one?

Q: How does 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”
  - (more in next chapter)
IP addresses: how to get one?

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

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

- ISPs get it from **ICANN**: Internet Corporation for Assigned Names and Numbers
  - Allocates addresses, manages DNS, resolves disputes

<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>
**IP route lookups**

- **Original IP Route Lookup**
  - In the early days, address classes made it easy
    - A: 0 | 7 bit network | 24 bit host (16M each)
    - B: 10 | 14 bit network | 16 bit host (64K)
    - C: 110 | 21 bit network | 8 bit host (255)
  - Address would specify prefix for forwarding table
  - Simple lookup
Original IP Route Lookup - Example

- **www.pdx.edu** address 131.252.120.50
  - Class B address - class + network is 131.252
  - Lookup 131.252 in forwarding table
  - Prefix - part of address that really matters for routing

- Forwarding table contains
  - List of prefix entries
  - A few fixed prefix lengths (8/16/24)

- Large tables
  - 2 Million class C networks
  - Sites with multiple class C networks have multiple route entries at every router
Getting a datagram from source to dest.

Classful routing example

IP datagram:

<table>
<thead>
<tr>
<th>misc fields</th>
<th>source IP addr</th>
<th>dest IP addr</th>
<th>data</th>
</tr>
</thead>
</table>

• datagram remains unchanged, as it travels source to destination
• addr fields of interest here

dest. Net. next router Nhops
223.1.1 1
223.1.2 223.1.1.4 2
223.1.3 223.1.1.4 2

routing table in A
Getting a datagram from source to dest.

Starting at A, given IP datagram addressed to B:
- look up net. address of B
- find B is on same net. as A
- link layer will send datagram directly to B inside link-layer frame
  - B and A are directly connected
Getting a datagram from source to dest.

Starting at A, dest. E:
- look up network address of E
- E on different network
  - A, E not directly attached
- routing table: next hop router to E is 223.1.1.4
- link layer sends datagram to router 223.1.1.4 inside link-layer frame
- datagram arrives at 223.1.1.4
- continued.....

<table>
<thead>
<tr>
<th>Dest. Net.</th>
<th>next router</th>
<th>Nhops</th>
</tr>
</thead>
<tbody>
<tr>
<td>223.1.1</td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>223.1.2</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
<tr>
<td>223.1.3</td>
<td>223.1.1.4</td>
<td>2</td>
</tr>
</tbody>
</table>
Getting a datagram from source to dest.

Arriving at 223.1.4, destined for 223.1.2.2

- Look up network address of E
- E on same network as router’s interface 223.1.2.9
  - Router, E directly attached
- Link layer sends datagram to 223.1.2.2 inside link-layer frame via interface 223.1.2.9
- Datagram arrives at 223.1.2.2!!! (hooray!)
IP route lookup and CIDR

- Recall Classless routing (CIDR)
  - Advantages
    - Saves space in route tables
    - Makes more efficient use of address space
      - ISP allocated 8 class C chunks, 201.10.0.0 to 201.10.7.255
      - Allocation uses 3 bits of class C space
      - Remaining 21 bits are network number, written as 201.10.0.0/21
      - Replace 8 class C entries with 1 combined entry
    - Routing protocols carry prefix length with destination network address
  - But....Makes route lookup more complex
    - No longer separate class A/B/C route tables each with $O(1)$ lookup
    - One table containing many prefix lengths
    - Must match against all routes simultaneously via longest prefix match
**CIDR example**

ISP X given 16 class C networks
200.23.16.* to 200.23.31.* (or 200.23.16/20)

<table>
<thead>
<tr>
<th>Route</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>200.23.16/20</td>
<td>1</td>
</tr>
<tr>
<td>200.23.16/21</td>
<td>2</td>
</tr>
<tr>
<td>200.23.24/22</td>
<td>3</td>
</tr>
<tr>
<td>200.23.28/23</td>
<td>4</td>
</tr>
<tr>
<td>200.23.30/24</td>
<td>5</td>
</tr>
</tbody>
</table>

Adjacent ISP router

Large company
200.23.16.0/21
200.23.16.0/24, 200.23.18.0/24, 200.23.20.0/24, 200.23.22.0/24

Medium company
200.23.24.0/22
200.23.24.0/24, 200.23.25.0/24, 200.23.26.0/24, 200.23.27.0/24

Small company
200.23.28.0/23
200.23.28.0/24, 200.23.29.0/24

Tiny company
200.23.30.0/24
CIDR route aggregation

Hierarchical addressing allows efficient advertisement of routing information:

- 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

Fly-By-Night-ISP

ISPs-R-Us

“Send me anything with addresses beginning 200.23.16.0/20”

“Send me anything with addresses beginning 199.31.0.0/16”

Internet
Another CIDR example

- Routing to the network
  - Packet to 10.1.1.3 arrives
  - Path is R2 – R1 – H1 – H2
Another CIDR example

- Subnet Routing
  - Packet to 10.1.1.3
  - Matches 10.1.0.0/22

Routing table at R2

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>lo0</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>provider</td>
<td>10.1.16.1</td>
</tr>
<tr>
<td>10.1.8.0/24</td>
<td>10.1.8.1</td>
<td>10.1.8.1</td>
</tr>
<tr>
<td>10.1.2.0/24</td>
<td>10.1.2.1</td>
<td>10.1.2.1</td>
</tr>
<tr>
<td>10.1.0.0/22</td>
<td>10.1.2.2</td>
<td>10.1.2.1</td>
</tr>
</tbody>
</table>
Another CIDR example

- Subnet Routing
  - Packet to 10.1.1.3
  - Matches 10.1.1.2/31
    - Longest prefix match

Routing table at R1

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>lo0</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>10.1.2.1</td>
<td>10.1.2.2</td>
</tr>
<tr>
<td>10.1.3.0/24</td>
<td>10.1.3.1</td>
<td>10.1.3.1</td>
</tr>
<tr>
<td>10.1.1.0/24</td>
<td>10.1.1.1</td>
<td>10.1.1.1</td>
</tr>
<tr>
<td>10.1.2.0/24</td>
<td>10.1.2.2</td>
<td>10.1.2.2</td>
</tr>
<tr>
<td>10.1.1.2/31</td>
<td>10.1.1.4</td>
<td>10.1.1.1</td>
</tr>
</tbody>
</table>

10.1.1.3 matches both routes, use longest prefix match
Another CIDR example

- Subnet Routing
  - Packet to 10.1.1.3
  - Direct route
    - Longest prefix match

Routing table at H1

<table>
<thead>
<tr>
<th>Destination</th>
<th>Next Hop</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>lo0</td>
</tr>
<tr>
<td>Default or 0/0</td>
<td>10.1.1.1</td>
<td>10.1.1.4</td>
</tr>
<tr>
<td>10.1.1.0/24</td>
<td>10.1.1.4</td>
<td>10.1.1.4</td>
</tr>
<tr>
<td>10.1.1.2/31</td>
<td>10.1.1.2</td>
<td>10.1.1.2</td>
</tr>
</tbody>
</table>

10.1.1.3 matches both routes, use longest prefix match
CIDR Shortcomings

- Customer selecting a new provider
  - Renumbering required

Diagram:

- 201.10.0.0/21
- 201.10.0.0/22
- 201.10.4.0/24
- 201.10.5.0/24
- 201.10.6.0/23
- 199.31.0.0/16

Provider 1

Provider 2
**CIDR shortcomings**

- **More specific routes**
- **Multi-homing**

ISPs-R-Us 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

Fly-By-Night-ISP

ISPs-R-Us

Internet

"Send me anything with addresses beginning 200.23.16.0/20"

"Send me anything with addresses beginning 199.31.0.0/16 or 200.23.18.0/23"
Longest-prefix matching

- Algorithms and data structures for CIDR-based IP route lookups
    - Binary trie
    - Multi-bit trie
    - LC trie
    - Lulea trie
    - Full expansion/compression
    - Binary search on prefix lengths
    - Binary range search
    - Multiway range search
    - Multiway range trees
    - Binary search on hash tables (Waldvogel - SIGCOMM 97)
**Binary trie**

- Data structure to support longest-prefix match for forwarding
- Bit-wise traversal from left-to-right

Route Prefixes:
- A: 0*
- B: 01000*
- C: 011*
- D: 1*
- E: 100*
- F: 1100*
- G: 1101*
- H: 1110*
- I: 1111*

Network Layer 4-70
Path-compressed binary trie

- Eliminate single branch point nodes
- Compare address against all prefixes along path to leaf
  - Take deepest match
- Variants include PATRICIA and BSD tries

Route Prefixes
A     0*
B     01000*
C     011*
D     1*
E     100*
F     1100*
G     1101*
H     1110*
I     1111*

Diagram:
- Node A has labels [Bit=3, Bit=1, Bit=0, Bit=1]
- Node B has labels [Bit=3, Bit=0, Bit=1]
- Node C has labels [Bit=2, Bit=1]
- Node D has labels [Bit=2, Bit=0, Bit=1]
- Node E has labels [Bit=3, Bit=0]
- Node F has labels [Bit=4, Bit=0, Bit=0, Bit=1]
- Node G has labels [Bit=4, Bit=0, Bit=1]
- Node H has labels [Bit=4, Bit=0, Bit=0, Bit=1]
Example #2: Binary trie

Route Prefixes
A  0*
B  00010*
C  00011*
Example #2: Path-compressed binary trie

Route Prefixes
A  0*
B  00010*
C  00011*

Diagram:
- Node A: Bit=5
- Node B: Bit=0
- Node C: Bit=1
Multi-bit tries

- Compare multiple bits at a time
  - Stride = number of bits being examined
  - Reduces memory accesses
  - Increase memory required
    - Forces table expansion for prefixes falling in between strides
- Two types
  - Variable stride multi-bit tries
  - Fixed stride multi-bit tries

- Most route entries are Class C
  - Optimize “stride” based on this
Variable stride multi-bit trie

- Single level has variable stride lengths

Route Prefixes
A 0*
B 01000*
C 011*
D 1*
E 100*
F 1100*
G 1101*
H 1110*
I 1111*
Fixed stride multi-bit trie

- Single level has equal strides

Route Prefixes
A 0*
B 01000*
C 011*
D 1*
E 100*
F 1100*
G 1101*
H 1110*
I 1111*
Issues

- Scaling
  - IPv6

- Stride choice
  - Tuning stride to route table
  - Bit shuffling
IP addressing and NAT

Network Address Translation (NAT)
- Alternate solution to address space depletion problem
  • Kludge (but useful)
- Sits between your network and the Internet
- Translates local, private, network layer addresses to global IP addresses
- Has a pool of global IP addresses (less than number of hosts on your network)

What if we only have few (or just one) IP address?
- Use NAPT (Network Address Port Translator)
- Both addresses and ports are translated
  • Translates Paddr + flow info to Gaddr + new flow info
  • Uses TCP/UDP port numbers
- Potentially thousands of simultaneous connections with one global IP address
**NAT Illustration**

**Operation:** Source (S) wants to talk to Destination (D):
- Create $S_g$-$S_p$ mapping
- Replace $S_p$ with $S_g$ for outgoing packets
- Replace $S_g$ with $S_p$ for incoming packets
NAPT: Network Address and Port 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

- **Advantages**
  - range of addresses not needed from ISP: just a small set of IP addresses 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

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: 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
  - violates end-to-end argument
    - NAT possibility must be taken into account by app designers, eg, P2P applications
  - address shortage should instead be solved by IPv6
Problems with NAT

- Hides the internal network structure
  - Some consider this an advantage
- Multiple NAT hops must ensure consistent mappings
- Some protocols carry addresses
  - e.g., FTP carries addresses in text
  - What is the problem?
- Encryption
- No inbound connections
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
ICMP: Internet Control Message Protocol

- Essentially a network-layer protocol for passing control messages
- 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>

http://www.rfc-editor.org/rfc/rfc792.txt
Traceroute and ICMP

- Source sends series of UDP segments to dest
  - First has TTL = 1
  - Second has TTL = 2, etc.
  - Unlikely port number

- When nth datagram arrives to nth router:
  - Router discards datagram
  - And sends to source an ICMP message (type 11, code 0)
  - Message includes name of router & IP address

- When ICMP message arrives, source calculates RTT

- Traceroute does this 3 times

Stopping criterion

- UDP segment eventually arrives at destination host
- Destination returns ICMP “host unreachable” packet (type 3, code 3)
- When source gets this ICMP, stops.
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
IPv6

- Redefine functions of IP (version 4)
  - What changes should be made in....
    - IP addressing
    - IP delivery semantics
    - IP quality of service
    - IP security
    - IP routing
    - IP fragmentation
    - IP error detection
IPv6

- **Initial motivation**: 32-bit address space soon to be completely allocated (est. 2008)

- **Additional motivation**:
  - Remove ancillary functionality
    - header format helps speed processing/forwarding
  - Add missing, but essential functionality
    - header changes to facilitate QoS
    - new “anycast” address: route to “best” of several replicated servers

**IPv6 datagram format**:
- fixed-length 40 byte header
- no fragmentation allowed
**IPv6 Header (Cont)**

**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

---

<table>
<thead>
<tr>
<th>ver</th>
<th>pri</th>
<th>flow label</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>payload len</th>
<th>next hdr</th>
<th>hop limit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

source address
(128 bits)

destination address
(128 bits)

data

32 bits
IPv6 Changes

- Scale - addresses are 128bit
  - Header size?
- Simplification
  - Removes infrequently used parts of header
  - 40 byte fixed header vs. 20+ byte variable header
- IPv6 removes checksum
  - IPv4 checksum = provide extra protection on top of data-link layer and below transport layer
  - End-to-end principle
    - Is this necessary?
    - IPv6 answer => No
  - Relies on upper layer protocols to provide integrity
  - Reduces processing time at each hop
IPv6 Changes

- IPv6 eliminates fragmentation
  - Requires path MTU discovery
- ICMPv6: new version of ICMP
  - additional message types, e.g. “Packet Too Big”
  - multicast group management functions
- Protocol field replaced by next header field
  - Unify support for protocol demultiplexing as well as option processing
- Option processing
  - Options allowed, but only outside of header, indicated by “Next Header” field
  - Options header does not need to be processed by every router
    - Large performance improvement
    - Makes options practical/useful
IPv6 Changes

- TOS replaced with traffic class octet
  - Support QoS via DiffServ
- FlowID field
  - Help soft state systems, accelerate flow classification
  - Maps well onto TCP connection or stream of UDP packets on host-port pair
- Easy configuration
  - Provides auto-configuration using hardware MAC address
- Additional requirements
  - Support for security
  - Support for mobility
Transition From IPv4 To IPv6

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

- Two proposed approaches:
  - Dual Stack: some routers with dual stack (v6, v4) can “translate” between formats
  - Tunneling: IPv6 carried as payload in an IPv4 datagram among IPv4 routers
**Tunneling**

**Logical view:**
- A (IPv6)
- B (IPv6)
- Tunnel
- E (IPv6)
- F (IPv6)

**Physical view:**
- A (IPv6)
- B (IPv6)
- IPv4
- IPv4
- E (IPv6)
- F (IPv6)
Tunneling

Logical view:

Physical view:

A-to-B: IPv6

B-to-C: IPv6 inside IPv4

E-to-F: IPv6 inside IPv4

Flow: X 
Src: A 
Dest: F 
data

Flow: X 
Src: A 
Dest: F 
data

Flow: X 
Src: A 
Dest: F 
data

Flow: X 
Src: A 
Dest: F 
data

Network Layer 4-98
Dual Stack Approach

- Dual-stack router translates b/w v4 and v6
  - v4 addresses have special v6 equivalents
  - Issue: how to translate "FlowField" of v6?
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Interplay between routing, forwarding

- Previously: Forward based on forwarding table
- Q: How to generate forwarding tables?
  - Routing algorithms and protocols

<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>
Routing

Routing protocol
Goal: determine “good” path (sequence of routers) thru network from source to dest.

Graph abstraction for routing algorithms:
- graph nodes are routers
- graph edges are physical links
  - link cost
    - Delay
    - $ cost
    - congestion level

“good” path:
- typically means minimum cost path
- other def’s possible
Who handles IP routing functions?

- **Source (IP source routing)**
  - Packet carries path

- **Network edge devices**
  - Map IP route into label, wavelength, or circuit at edges
  - Switch on label, wavelength, or circuit in the core
    - ATM
    - MPLS
    - lambda switching

- **Network routers**
  - Hop-by-hop forwarding based on destination IP carried by packet
  - Routers keep next hop for destination
  - IP route table calculated in network routers
  - Most common
Source Routing

- IP source route option
  - List entire path (strict) or partial path (loose) in packet
  - Attach list of IP addresses within header

- Router processing
  - Examine first step in directions
    - Increment pointer offset in header
    - Forward to step
    - Copy entire source route header on fragmentation
Source Routing Example

Packet → Sender → R1 → R1 → R2 → Receiver

Sender: 3,4,3
R1: 4,3
R1: 3
R2: 1,2
Receiver: 3
Source Routing

- Advantages
  - Switches can be very simple and fast

- Disadvantages
  - Variable (unbounded) header size
  - Sources must know or discover topology (e.g., failures)

- Typical use
  - Ad-hoc networks (DSR)
  - Machine room networks (Myrinet)
Network edge device routing

- Virtual circuits, tag switching

- Connection setup phase
  - IP route lookup at edges to generate appropriate label, wavelength, circuit
  - Switch on label, wavelength, circuit ID in core

- In-network processing
  - Lookup flow ID - simple table lookup
  - Potentially replace flow ID with outgoing flow ID
  - Forward to output port
Virtual Circuits Examples
Virtual Circuits

- Advantages
  - More efficient lookup (simple table lookup)
    - Easier for hardware implementations
  - More flexible (different path for each flow)
  - Can reserve bandwidth at connection setup

- Disadvantages
  - Still need to route connection setup request
  - More complex failure recovery - must recreate connection state

- Typical uses
  - ATM - combined with fix sized cells
  - MPLS - tag switching for IP networks
IP Datagrams on Virtual Circuits

- Challenge - when to setup connections
  - At bootup time - permanent virtual circuits (PVC)
    - Large number of circuits
  - For every packet transmission
    - Connection setup is expensive
  - For every connection
    - What is a connection?
    - How to route connectionless traffic?
  - Based on traffic
    - VC for long-lived flows
    - Normal IP forwarding for all other flows
Network routers (Global IP addresses)

- Most prevalent way to route on the Internet
  - Each packet has destination IP address
  - Each router has forwarding table of..
    - destination IP  $\rightarrow$ next hop IP address
  - Distributed routing algorithm for calculating forwarding tables
Global Address Example
Issues in Router Table Size

- One entry for every host on the Internet
  - 100M entries

- One entry for every LAN
  - Every host on LAN shares prefix
  - Still too many

- One entry for every organization
  - Every host in organization shares prefix
  - Requires careful address allocation
  - What constitutes an “organization”?
Global Addresses

- Advantages
  - Simple error recovery

- Disadvantages
  - Every router knows about every destination
    - Potentially large tables
  - All packets to destination take same route
## Comparison

<table>
<thead>
<tr>
<th></th>
<th>Source Routing</th>
<th>Global Addresses</th>
<th>Virtual Circuits</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Header Size</strong></td>
<td>Worst</td>
<td>OK – Large address</td>
<td>OK (larger than global if IP payload)</td>
</tr>
<tr>
<td><strong>Router Table Size</strong></td>
<td>None</td>
<td>Number of hosts (prefixes)</td>
<td>Number of circuits</td>
</tr>
<tr>
<td><strong>Forward Overhead</strong></td>
<td>Best</td>
<td>Prefix matching</td>
<td>Good (table index)</td>
</tr>
<tr>
<td><strong>Setup Overhead</strong></td>
<td>None</td>
<td>None</td>
<td>Connection Setup</td>
</tr>
<tr>
<td><strong>Error Recovery</strong></td>
<td>Tell all hosts</td>
<td>Tell all routers</td>
<td>Tell all routers, Tear down circuit and re-route</td>
</tr>
</tbody>
</table>
Graph abstraction

Graph: $G = (N,E)$

$N =$ set of routers $= \{ u, v, w, x, y, z \}$

$E =$ set of links $= \{ (u,v), (u,x), (v,x), (v,w), (x,w), (x,y), (w,y), (w,z), (y,z) \}$

Remark: Graph abstraction is useful in other network contexts

Example: 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 $(x_1, x_2, x_3, ..., x_p) = c(x_1,x_2) + c(x_2,x_3) + ... + c(x_{p-1},x_p)$

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

Routing algorithm: algorithm that finds least-cost path
Routing Algorithm classification

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

Static or dynamic?

Static:
- routes change slowly over time

Dynamic:
- routes change more quickly
  - periodic update
  - in response to link cost changes
Other characteristics

- Communication costs
- Processing costs
- Optimality
- Stability
  - Convergence time
  - Loop freedom
  - Oscillation damping
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
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
Dijkstra’s algorithm

- **Start condition**
  - Each node assumed to know state of links to its neighbors

- **Step 1: Link state broadcast**
  - Each node broadcasts its local link states to all other nodes
  - Reliable flooding mechanism

- **Step 2: Shortest-path tree calculation**
  - Each node locally computes shortest paths to all other nodes from global state
  - Dijkstra’s shortest path tree (SPT) algorithm
Link state broadcast

- Link State Packets (LSPs) to broadcast state to all nodes
- Periodically, each node creates a link state packet containing:
  - Node ID
  - List of neighbors and link cost
  - Sequence number
  - Time to live (TTL)
  - Node outputs LSP on all its links
Link state broadcast

- **Reliable Flooding**
  - When node J receives LSP from node K
    - If LSP is the most recent LSP from K that J has seen so far, J saves it in database and forwards a copy on all links except link LSP was received on
    - Otherwise, discard LSP
  - How to tell more recent
    - Use sequence numbers
      - Same method as sliding window protocols
      - Needed to avoid stale information from flood
      - Problem: sequence number wrap-around
        » Lollipop sequence space
wrapped sequence numbers

- wrapped sequence numbers
  - 0-N where N is large
  - If difference between numbers is large, assume a wrap
  - A is older than B if:
    - A < B and |A-B| < N/2 or...
    - A > B and |A-B| > N/2

- What about new nodes or rebooted nodes that are out of sync with sequence number space?
  - Lollipop sequence (Perlman 1983)
Lollipop sequence numbers

- Divide sequence number space
- Special negative sequence for recovering from reboot
  - New and rebooted nodes use negative sequence numbers
  - Upon receipt of negative number, other nodes inform these nodes of current “up-to-date” sequence number
- A older than B if
  - A < 0 and A < B
  - A > 0, A < B and (B - A) < N/4
  - A > 0, A > B and (A - B) > N/4
Shortest-path tree calculation

Notation:

- $c(x,y)$: link cost from node $x$ to $y$; $= 8$ 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 \( D(v) = 8 \)
   
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 \( N' \):
       \[ D(v) = \min(D(v), D(w) + c(w,v)) \]
   12. /* new cost to \( v \) is either old cost to \( v \) or known shortest path cost to \( w \) plus cost from \( w \) to \( v \) */
   15. **until all nodes in \( N' \)**
Shortest-path tree calculation
(Dijkstra’s algorithm example)

D(v) = min( D(v), D(w) + c(w,v) )

---

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>
## Dijkstra's algorithm example

Dijkstra's algorithm is used to find the shortest path(s) from one node to another in a graph. The algorithm maintains two sets: a set of nodes already included in the shortest path tree (SPT) and a set of nodes not yet included. At each step, the algorithm selects the node with the smallest shortest path estimate and adds it to the SPT. The distance from the starting node to each newly added node is then updated based on the shortest path estimate from the starting node through the newly added node.

### Steps of Dijkstra's algorithm example

1. **Step 0**
   - **SPT**: A
   - **D(A)**: 0
   - **D(B)**: 2
   - **D(C)**: 5
   - **D(D)**: 1
   - **D(E)**: ~
   - **D(F)**: ~

2. **Step 1**
   - **SPT**: AD
   - **D(A)**: 2
   - **D(B)**: 4
   - **D(C)**: 4
   - **D(D)**: 2
   - **D(E)**: 2
   - **D(F)**: ~

### Graph with edge weights

```
A --2-- B --3-- C
 
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

A --1-- D --3-- E
 
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>
```

### Update rules

- **D(v) = min( D(v), D(w) + c(w,v) )**

### Table: Dijkstra's algorithm example

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</td>
</tr>
</tbody>
</table>
Dijkstra's algorithm example

\[ D(v) = \min( D(v), D(w) + c(w,v) ) \]

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2, A</td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's algorithm example

\[ D(v) = \min(D(v), D(w) + c(w,v)) \]

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2, A</td>
<td>3, E</td>
<td>4, E</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td>3, E</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra’s algorithm example

D(v) = \min( D(v), D(w) + c(w,v) )

A

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2, A</td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's algorithm example

D(v) = min( D(v), D(w) + c(w,v) )

<table>
<thead>
<tr>
<th>step</th>
<th>SPT</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>1, A</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>1</td>
<td>AD</td>
<td>2, A</td>
<td>4, D</td>
<td>2, D</td>
<td>~</td>
<td>~</td>
</tr>
<tr>
<td>2</td>
<td>ADE</td>
<td>2, A</td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>ADEB</td>
<td></td>
<td>3, E</td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ADEBC</td>
<td></td>
<td></td>
<td></td>
<td>4, E</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>ADEBCF</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Dijkstra's algorithm example

Resulting shortest-path tree from A:

Resulting forwarding table 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>
Link state algorithm characteristics

- **Computation overhead**
  - 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: $O(n \log(n))$

- **Space requirements**

- **Bandwidth requirements**

- **Stability**
  - Inconsistencies can cause transient loops
  - Consistent LSDBs required for loop-free paths

Packet from C→A may loop around BDC if B knows about failure and C & D do not
**Link-state algorithm issues**

Oscillations possible:
- e.g., link cost = amount of carried traffic
- Example: path to A flaps as traffic routed clockwise and counter-clockwise
- Common problem in load-based link metrics

```
A  D  A
\text{initially}

\begin{align*}
D & \rightarrow A: 1+e \\
D & \rightarrow C: 0 \\
D & \rightarrow B: 0
\end{align*}
```

```
A  D  A
\text{... recomputed routing}

\begin{align*}
D & \rightarrow A: 2+e \\
D & \rightarrow C: 1+e \\
D & \rightarrow B: 0
\end{align*}
```

```
A  D  A
\text{... recomputed}

\begin{align*}
A & \rightarrow D: 1 \\
A & \rightarrow B: 2+e \\
A & \rightarrow C: 1+e
\end{align*}
```

```
A  D  A
\text{... recomputed}

\begin{align*}
D & \rightarrow A: 2+e \\
D & \rightarrow C: 1+e \\
D & \rightarrow B: 0
\end{align*}
```
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Distance vector routing algorithms

- Variants used in
  - Early ARPAnet
  - RIP (intra-domain routing protocol)
  - BGP (inter-domain routing protocol)

- Distributed next hop computation
  - “Gossip with immediate neighbors until you find the best route”
  - Best route is achieved when there are no more changes

- Unit of information exchange
  - Vector of distances to destinations
Distance Vector Algorithm

Bellman-Ford Equation

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

Then

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

where \( \min \) is 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:

\[
d_u(z) = \min \{ c(u,v) + d_v(z), \]
\[
\quad c(u,x) + d_x(z), \]
\[
\quad c(u,w) + d_w(z) \}
\]

\[
= \min \{ 2 + 5, \quad 1 + 3, \quad 5 + 3 \} = 4
\]

Node that achieves minimum is next hop in shortest path? forwarding table
## Bellman algorithm

- Update distance information iteratively
- Example (Bellman 1957)
  - Start with link table (as with Dijkstra), calculate distance table iteratively
  - Distance table data structure
    - table of known distances and next hops kept per node
    - row for each possible destination
    - column for each directly-attached neighbor to node
    - example: in node X, for dest. Y via neighbor Z:
Bellman algorithm

- Centralized version

For node $i$
while there is a change in $D$
  for all $k$ not neighbor of $i$
    for each $j$ neighbor of $i$
      $D_i(k,j) = c(i,j) + D_j(k,*)$
      if $D_i(k,j) < D_i(k,*)$ {
        $D_i(k,*) = D_i(k,j)$
        $H_i(k) = j$

$D_X(Y,Z) = \text{distance from } X \text{ to } Y, \text{ via } Z \text{ as next hop}$

$D_X(Y,*) = \text{Minimum known distance from } X \text{ to } Y$

$H_X(Y) = \text{Next hop node from } X \text{ to } Y$
Distance table example

\[ D^{E}(C,D) = c(E,D) + \min_w \{ D^D(C,w) \} \]
\[ = 2 + 2 = 4 \]

\[ D^{E}(A,D) = c(E,D) + \min_w \{ D^D(A,w) \} \]
\[ = 2 + 3 = 5 \quad \text{loop!} \]

\[ D^{E}(A,B) = c(E,B) + \min_w \{ D^B(A,w) \} \]
\[ = 8 + 6 = 14 \quad \text{loop!} \]
Distance table gives forwarding table

Distance table:

<table>
<thead>
<tr>
<th>Source</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Routing table:

<table>
<thead>
<tr>
<th>Destination</th>
<th>Outgoing link to use, cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>A,1</td>
</tr>
<tr>
<td>B</td>
<td>D,5</td>
</tr>
<tr>
<td>C</td>
<td>D,4</td>
</tr>
<tr>
<td>D</td>
<td>D,4</td>
</tr>
</tbody>
</table>
Distributed Bellman-Ford

- Make Bellman algorithm distributed (Ford-Fulkerson 1962)
  - Each node $i$ has distance vector estimates to other nodes
  - Iterate
    - Each node sends around and recalculates $D[i,*]$
    - When a node $x$ receives new DV estimate from neighbor, it updates its own DV using B-F equation:
      $$D_x(y) = \min_v \{c(x,v) + D_v(y)\} \quad \text{for each node } y \in N$$
    - If estimates change, broadcast entire table to neighbors
      - continues until no nodes exchange info.
      - self-terminating: no “signal” to stop
  - $D[i,*]$ eventually converges to shortest distance
Distributed Bellman-Ford overview

Asynchronous:
- “triggered updates”
  - no need to exchange info/iterate in lock step!

Iterative:
- When local link costs change
- When neighbor sends a message that its least cost path has changed for a node

Distributed:
- nodes communicate only with directly-attached neighbors
- each node notifies neighbors only when its least cost path to any destination changes
  - neighbors then notify their neighbors if necessary

Each node:
- \textit{wait} for (change in local link cost of msg from neighbor)
- \textit{recompute} distance table
- if least cost path to any dest has changed, \textit{notify} neighbors
Distributed Bellman-Ford algorithm

At all nodes, \( X \):

1. Initialization:
2. for all adjacent nodes \( v \):
3. \( D^X(*,v) = \text{infinity} \) \( /* \) the * operator means "for all rows" */
4. \( D^X(v,v) = c(X,v) \)
5. for all destinations, \( y \)
6. send \( \min_w D^X(y,w) \) to each neighbor \( /* \) w over all X's neighbors */
Distributed Bellman-Ford algorithm

8 loop
9 wait (until I see a link cost change to neighbor V or until I receive update from neighbor V)
11
12 if (c(X,V) changes by d)
13 /* change cost to all dest's via neighbor v by d */
14 /* note: d could be positive or negative */
15 for all destinations y: D^X(y,V) = D^X(y,V) + d
16
17 else if (update received from V wrt destination Y)
18 /* shortest path from V to some Y has changed */
19 /* V has sent a new value for its \( \min_w D^V(Y,w) \) */
20 /* call this received new value is "newval" */
21 for the single destination y: D^X(Y,V) = c(X,V) + newval
22
23 if we have a new \( \min_w D^X(Y,w) \) for any destination Y
24 send new value of \( \min_w D^X(Y,w) \) to all neighbors
25
26 forever
DBF example

Initial Distance Vectors

<table>
<thead>
<tr>
<th>Info at Node</th>
<th>Distance to Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>~</td>
</tr>
<tr>
<td>D</td>
<td>~</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
</tbody>
</table>
DBF example

E Receives D’s Routes
Updates cost to C

```

<table>
<thead>
<tr>
<th>Info at Node</th>
<th>Distance to Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0 7 ~ ~ 1</td>
</tr>
<tr>
<td>B</td>
<td>7 0 1 ~ 8</td>
</tr>
<tr>
<td>C</td>
<td>~ 1 0 2 ~</td>
</tr>
<tr>
<td>D</td>
<td>~ ~ 2 0 2</td>
</tr>
<tr>
<td>E</td>
<td>1 8 4 2 0</td>
</tr>
</tbody>
</table>

```
**DBF example**

A receives B’s update

Updates cost to C, but cost to E unchanged

---

**Diagram:**

A receives B’s update

**Table:**

<table>
<thead>
<tr>
<th>Info at Node</th>
<th>Distance to Node</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0    7   8    ~   1</td>
</tr>
<tr>
<td>B</td>
<td>7    0    1    ~   8</td>
</tr>
<tr>
<td>C</td>
<td>~    1    0    2    ~</td>
</tr>
<tr>
<td>D</td>
<td>~    ~    2    0    2</td>
</tr>
<tr>
<td>E</td>
<td>1    8    4    2    0</td>
</tr>
</tbody>
</table>
**DBF example**

A receives E’s routes
Updates cost to C (new min) and D

![Diagram showing network nodes A, B, C, D, E with distances and routes]

<table>
<thead>
<tr>
<th>Info at Node</th>
<th>Distance to Node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>A</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>C</td>
<td>~</td>
</tr>
<tr>
<td>D</td>
<td>~</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
</tr>
</tbody>
</table>
DBF example

And so on, until final distances....
DBF example

E’s routing table

<table>
<thead>
<tr>
<th>dest</th>
<th>A</th>
<th>B</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>B</td>
<td>7</td>
<td>8</td>
<td>5</td>
</tr>
<tr>
<td>C</td>
<td>6</td>
<td>9</td>
<td>4</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>11</td>
<td>2</td>
</tr>
</tbody>
</table>
**DBF (another example)**

- See book for explanation of this example

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>X</td>
<td>2</td>
<td>∞</td>
</tr>
<tr>
<td>dest</td>
<td>Z</td>
<td>∞</td>
<td>7</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>X</td>
<td>2</td>
<td>∞</td>
</tr>
<tr>
<td>dest</td>
<td>Z</td>
<td>∞</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>X</td>
<td>7</td>
<td>∞</td>
</tr>
<tr>
<td>dest</td>
<td>Y</td>
<td>1</td>
<td>∞</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>D</th>
<th>Y</th>
<th>Z</th>
</tr>
</thead>
<tbody>
<tr>
<td>dest</td>
<td>X</td>
<td>7</td>
<td>3</td>
</tr>
<tr>
<td>dest</td>
<td>Y</td>
<td>9</td>
<td>1</td>
</tr>
</tbody>
</table>
DBF (another example)

\[ D^X(Y,Z) = c(X,Z) + \min_w \{D^Z(Y,w)\} = 7+1 = 8 \]

\[ D^X(Z,Y) = c(X,Y) + \min_w \{D^Y(Z,w)\} = 2+1 = 3 \]
DBF (good news example)

Link cost changes:

- node detects local link cost change
- updates distance table (line 15)
- if cost change in least cost path, notify neighbors (lines 23, 24)
- fast convergence (see book for details)

"good news travels fast"
At time $t_0$, $y$ detects the link-cost change, updates its DV, and informs its neighbors.

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

At time $t_2$, $y$ receives $z$'s update and updates its distance table. $y$'s least costs do not change and hence $y$ does not send any message to $z$. 
DBF (count-to-infinity example)

Link cost changes:

• good news travels fast
• bad news travels slow - “count to infinity” problem!
• alternate route implicitly used link that changed

![Diagram](image-url)
DBF: (count-to-infinity example)

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>25</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

Network Layer 4-161
DBF: (count-to-infinity example)

C Sends Routes to B

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>~</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>
DBF: (count-to-infinity example)

B Updates Distance to A

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

![Diagram showing network nodes A, B, and C with distances and costs]
DBF: (count-to-infinity example)

B Sends Routes to C

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>3</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>dest</th>
<th>cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
</tr>
</tbody>
</table>

Diagram showing the network with nodes A, B, and C, and their costs.
DBF: (count-to-infinity example)

C Sends Routes to B
Analyzing Distributed Bellman-Ford

- Continuously send local distance tables of best known routes to all neighbors until your table converges
  - Computation diffuses until all nodes converge
  - Will computation converge quickly and deterministically?
    - Not all the time, pathologic cases possible (count-to-infinity)
    - Several algorithms for minimizing such cases
How are loops caused?

- **Observation 1:**
  - B’s metric increases

- **Observation 2:**
  - C picks B as next hop to A
  - But, the implicit path from C to A includes itself!
Solutions to looping

- **Split horizon**
  - Do not advertise route to X to an adjacent neighbor if your route to X goes through that neighbor
  - If C routes through B to get to A, C does not advertise (C=>A) route to B.

- **Poisoned reverse**
  - Advertise an infinite distance route to X to an adjacent neighbor if your route to X goes through that neighbor
  - If C routes through B to get to A, C advertises to B that its distance to A is infinity

- **Works for two node loops**
  - Does not work for loops with more nodes
Split-horizon with 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?

![Diagram showing network layer communication and split-horizon with poisoned reverse](image)

- New route to X not involving Y
- Route to X through Y goes thru Z
- Poison it!

Network Layer 4-169
Solutions to looping
Solutions to looping

- Route poisoning
  - Advertise infinite cost on a route to everyone (not just next hop) when lowest cost route increases
  - Gets rid of stale information throughout network
  - Used in conjunction with Path Holdown

- Path Holddown
  - Freeze route for a fixed time
    - Do not switch to an alternate while route poisoning is happening
    - In our example, A and B delay changing and advertising new routes
    - A and B both set route to D to infinity after single step
  - Configuring holddown delay
    - Delay too large: Slow convergence
    - Delay too small: Count-to-infinity more probable
Solutions to looping

- Path vector
  - Select loop-free paths
  - Each route advertisement carries entire path
  - If a router sees itself in path, it rejects the route
  - BGP does it this way
  - Space proportional to diameter of network
Solutions to looping

Do solutions completely eliminate loops?
- No! Transient loops are still possible
- Why? Because implicit path information may be stale
- See this in BGP convergence

Only way to fix this
- Ensure that you have up-to-date information by explicitly querying
Link State vs. Distance Vector

Message complexity, network bandwidth

- **LS:** with \( n \) nodes, \( E \) links, \( O(nE) \) msgs sent
  - Send info about your neighbors to everyone
  - Small messages broadcast globally

- **DV:** exchange between neighbors only
  - Send everything you know to your neighbors
  - Large messages, but transfers only to neighbors
  - convergence time varies
Link State vs. Distance Vector

Speed of Convergence

- **LS**: $O(n^2)$ algorithm requires $O(nE)$ msgs
  - Faster - can forward LSPs before processing
  - Single SPT calculation

- **DV**: convergence time varies
  - Fast with triggered updates
  - Count-to-infinity problem
  - May be routing loops
Link State vs. Distance Vector

Space requirements:

- **LS**
  - maintains entire topology

- **DV**
  - maintains only neighbor state
  - path vector maintains routes proportional to network diameter
Link State vs. Distance Vector

Robustness:

- LS can broadcast incorrect/corrupted LSP
  - Can be made robust since sources are aware of alternate paths within topology
- DV can advertise incorrect paths to all destinations
  - Incorrect calculation can spread to entire network
DUAL

- Distributed Update Algorithm
  - Garcia-Luna-Aceves 1989
  - Goal: Avoid transient loops in DV and LS algorithms
    - Similar in flavor to route poisoning and path holddown
  - 2 ideas
    - A path shorter than current path cannot contain a loop
    - Based on diffusing computation (Dijkstra-Scholten 1980)
      - Wait until computation completes before changing routes in response to a new update
      - Similar to path-holddown
  - 3 kinds of messages
    - Update, query, reply
  - 2 states for routers
    - Active (queries outstanding), passive
DUAL

On update
if (lower cost) adopt
else if (higher cost) {
    if (from next hop) {
        if (any path exists < old length from next hop)
            switch path
    else
        freeze route
    send query to all neighbors except next hop
    go into active
    wait for reply from all neighbors
    update route
    return to passive
}
send reply to all querying neighbors
}
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Hierarchical Routing

Our routing study thus 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!
- Flat routing does not scale

administrative autonomy
- internet = network of networks
- each network admin may want to control routing in its own network
Routing Hierarchies

- **Key observation**
  - Need less information with increasing distance to destination

- **Two radically different approaches for routing**
  - The area hierarchy
  - The landmark hierarchy
    - Covered in advanced topics at end of course...
Areas

- **Divide network into areas**
  - Areas can have nested sub-areas
  - No path between two sub-areas of an area can exit that area
  - Within area, each node has routes to every other node
  - Outside area
    - Each node has routes for other top-level areas only
    - Inter-area packets are routed to nearest appropriate border router
    - Can result in sub-optimal paths

- **Hierarchically address nodes in a network**
  - Sequentially number top-level areas
  - Sub-areas of area are labeled relative to that area
  - Nodes are numbered relative to the smallest containing area
Hierarchical Routing on the Internet

- aggregate routers into regions, “autonomous systems” (AS)
  - administrative autonomy
- routers in same AS run same routing protocol
  - “intra-AS” routing protocol (IGP)
  - routers in different AS can run different intra-AS routing protocol

Gateway router
- Direct link to router in another AS
- special routers in AS
- run intra-AS routing protocol with all other routers in AS
- also responsible for routing to destinations outside AS
- run inter-AS routing protocol or exterior gateway protocol (EGP) with other gateway routers in other AS’s
Example #1
Example #2

Gateways:
- perform inter-AS routing amongst themselves
- perform intra-AS routing with other routers in their AS

inter-AS, intra-AS routing in gateway A.c

network layer
link layer
physical layer

Routing Table
- Intra-AS routing algorithm
- Inter-AS routing algorithm

Network Layer 4-186
Path Sub-optimality

3 hop red path vs. 2 hop green path
AS Categories

- **Stub**: an AS that has only a single connection to one other AS - carries only local traffic.
- **Multi-homed**: an AS that has connections to more than one AS, but does not carry transit traffic.
- **Transit**: an AS that has connections to more than one AS, and carries both transit and local traffic (under certain policy restrictions).
AS categories example
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Intra-AS Routing

Also known as Interior Gateway Protocols (IGP)

Most common Intra-AS routing protocols:

- **RIP**: Routing Information Protocol
  - Distance-vector

- **OSPF**: Open Shortest Path First
  - Link-state

- **IGRP**: Interior Gateway Routing Protocol (Cisco proprietary)
  - Distance-vector
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
**RIP (Routing Information Protocol)**

- Distance vector algorithm
  - Distance metric: # of hops (max = 15 hops)
  - Vectors exchanged every 30 sec and when triggered
  - Static update period leads to synchronization problems
  - Split horizon with poisonous reverse

- Included in BSD-UNIX Distribution in 1982
- RIP-2 in 1993 adds prefix mask for CIDR

**From router A to subsets:**

<table>
<thead>
<tr>
<th>Destination</th>
<th>Hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>u</td>
<td>1</td>
</tr>
<tr>
<td>v</td>
<td>2</td>
</tr>
<tr>
<td>w</td>
<td>2</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
</tr>
<tr>
<td>y</td>
<td>3</td>
</tr>
<tr>
<td>z</td>
<td>2</td>
</tr>
</tbody>
</table>
RIP advertisements

- Distance vectors: exchanged among neighbors every 30 sec via Response Message (also called advertisement)
- Each advertisement: list of up to 25 destination nets within AS
RIP: Example

<table>
<thead>
<tr>
<th>Destination Network</th>
<th>Next Router</th>
<th>Num. of hops to dest.</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>A</td>
<td>2</td>
</tr>
<tr>
<td>y</td>
<td>B</td>
<td>2</td>
</tr>
<tr>
<td>z</td>
<td>B</td>
<td>7</td>
</tr>
<tr>
<td>x</td>
<td>--</td>
<td>1</td>
</tr>
<tr>
<td>....</td>
<td>....</td>
<td>....</td>
</tr>
</tbody>
</table>

Routing table in D
RIP: Example

<table>
<thead>
<tr>
<th>Dest</th>
<th>Next</th>
<th>hops</th>
</tr>
</thead>
<tbody>
<tr>
<td>w</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>x</td>
<td>-</td>
<td>1</td>
</tr>
<tr>
<td>z</td>
<td>C</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Routing table in D

Destination Network | Next Router | Num. of hops to dest.
-------------------|-------------|----------------------
 w                 | A           | 2                    |
 y                 | B           | 2                    |
 z                 | B           | 2                    |
 x                 | --          | 1                    |
 ....              | ....        | ....                 |

Advertisement from A to D

Network Layer 4-196
RIP: Link Failure and Recovery

If no advertisement heard after 180 sec -->
neighbor/link declared dead
  - routes via neighbor invalidated
  - new advertisements sent to neighbors
  - neighbors in turn send out new advertisements (if tables changed)
  - link failure info quickly propagates to entire net
  - poison reverse used to prevent ping-pong loops (infinite distance = 16 hops)
RIP Table processing

- RIP routing tables managed by application-level process called route-d (daemon)
- advertisements sent in UDP packets, periodically repeated
IGRP (Interior Gateway Routing Protocol)

- CISCO proprietary; successor of RIP (mid 80s)
  - Distance Vector, like RIP
  - several cost metrics (delay, bandwidth, reliability, load etc)
  - 90 sec update with triggered updates
  - Split horizon
    - V1: path holddown
    - V2: route poisoning
    - multiple path support
  - uses TCP to exchange routing updates
- EIGRP
  - Loop-free routing via DUAL (based on diffused computation)
  - CIDR support
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
OSPF (Open Shortest Path First)

- “open”: publicly available
- Uses Link State algorithm
  - LS packet dissemination
  - Topology map at each node
  - Route computation using Dijkstra’s algorithm
- OSPF advertisement carries one entry per neighbor router
- Advertisements disseminated to entire AS (via flooding)
  - Carried in OSPF messages directly over IP (rather than TCP or UDP)
OSPF “advanced” features (not in RIP)

- **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 set “low” for best effort; high for real time)
- Integrated uni- and multicast support:
  - Multicast OSPF (MOSPF) uses same topology database as OSPF
- **Hierarchical** OSPF in large domains.
Hierarchical OSPF

- **Two-level hierarchy:** local area, backbone.
  - Link-state advertisements only in area
  - Each node 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’s.
Hierarchical OSPF
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Inter-AS routing

- EGP
- BGP
Why different Intra- and Inter-AS routing?

Policy:
- Inter-AS: ISP wants control over how its traffic routed, who routes through its net.
  - Policy and monetary factors dominate over performance
- Intra-AS: single administrative policy
  - No policy decisions needed, performance dominates
  - Focus on performance

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

- Mid-80s: EGP (Exterior Gateway Protocol)
  - Used in original ARPAnet
  - Reachability protocol (no shortest path)
    - Single bit for reachability information
  - Topology restricted to a tree (no cycles allowed)
    - ARPA-managed packet switches at top of tree
  - Unacceptable once Internet grew to multiple independent backbones

- Result: BGP development
Inter-AS routing: BGP

- Link state or distance vector?
  - Problems with distance-vector:
    - Bellman-Ford algorithm may not converge
  - More problems with link state:
    - Everyone sees every link
      - LS database too large – entire Internet
      - Can’t easily control who uses the network (i.e. an ISP may want to hide particular links from being used by others, but link states are broadcast)
    - Metric used by routers not the same – loops
      - No universal routing metric
      - Policy drives routing decisions
**BGP**

- **BGP (Border Gateway Protocol):** the de facto standard
  - Predecessor: EGP (Exterior Gateway Protocol)

- **BGP provides each AS a means to:**
  1. Obtain subnet reachability information from neighboring ASs.
  2. Propagate the reachability information to all routers internal to the AS.
  3. Determine “good” routes to subnets based on reachability information and policy.

- Allows a subnet to advertise its existence to rest of the Internet: “I am here”
BGP messages

BGP messages exchanged using TCP.

- **Advantages:**
  - Simplifies BGP
  - No need for periodic refresh - routes are valid until withdrawn, or the connection is lost
  - Note recent news on BGP TCP spoofing attack
  - Incremental updates

- **Disadvantages:**
  - Congestion control on a routing protocol?
  - Poor interaction during high load (Code Red)

- **BGP messages:**
  - **OPEN:** opens TCP connection to peer and authenticates sender
  - **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**

- **Path Vector** protocol:
  - similar to Distance Vector protocol
  - each Border Gateway broadcast to neighbors (peers) entire path (I.e, sequence of ASs) to destination
    - E.g., Gateway X sends its path to dest. Z:
      - Path $(X, Z) = X, Y_1, Y_2, Y_3, \ldots, Z$
  - When AS gets route check if AS already in path
    - If yes, reject route
    - If no, add self and (possibly) advertise route further
  - Allows for policy application (different metrics)
    - Metrics are local - AS chooses path, protocol ensures no loops

Supports CIDR aggregation (BGP4)
Supports alternative routes
**BGP basics**

- Pairs of routers (BGP peers) exchange routing info over semi-permanent TCP connections: **BGP sessions**
- Note that BGP sessions do not correspond to physical links.
- 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

![Diagram showing BGP sessions between AS1, AS2, and AS3]
Distributing reachability info

- With eBGP session between 3a and 1c, AS3 sends prefix reachability info to AS1.
- 1c can then use iBGP to distribute this new prefix reach info to all routers in AS1.
- 1b can then re-advertise the new reach info to AS2 over the 1b-to-2a eBGP session.
- When router learns about a new prefix, it creates an entry for the prefix in its forwarding table.
Policy with BGP

- BGP provides capability for enforcing various policies
- Policies are **not** part of BGP: they are provided to BGP as configuration information
- BGP enforces policies by choosing paths from multiple alternatives and controlling advertisement to other AS's
Path Selection Criteria

- Path attributes + external (policy) information

Examples:
- Hop count
- Policy considerations
  - Preference for AS
  - Presence or absence of certain AS
- Path origin
- Link dynamics
- Early-exit
  - Hot-potato routing for transit packets
Examples of BGP Policies

- A multi-homed AS refuses to act as transit
  - Limit path advertisement
- A multi-homed AS can become transit for some AS’s
  - Only advertise paths to some AS’s
- An AS can favor or disfavor certain AS’s for traffic transit from itself
BGP routing policy

- A, B, C are provider networks
- X, W, Y are customers (of provider networks)
- X is dual-homed: attached to two networks
  - X does not want to route from B via X to C
  - .. so X will not advertise to B a route to C

Legend:
- Blue circles represent provider networks
- White circles represent customer networks
BGP routing policy (2)

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!
Extra slides
Interplay between routing and forwarding

Routing algorithm

Local forwarding table

<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

Network Layer 4-221
Dijkstra's algorithm: 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>8</td>
<td>8</td>
</tr>
<tr>
<td>1</td>
<td>ux</td>
<td>2,u</td>
<td>4,x</td>
<td>2,x</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>uxy</td>
<td>2,u</td>
<td>3,y</td>
<td></td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>uxyv</td>
<td>3,y</td>
<td></td>
<td></td>
<td>4,y</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>uxyvw</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>uxyvwz</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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>
Distance Vector Algorithm

- $D_x(y) = \text{estimate of least cost from } x \text{ to } y$
- Distance vector: $D_x = [D_x(y): y \in N]$
- Node $x$ knows cost to each neighbor $v$: $c(x,v)$
- Node $x$ maintains $D_x = [D_x(y): y \in N]$
- Node $x$ also maintains its neighbors’ distance vectors
  - For each neighbor $v$, $x$ maintains $D_v = [D_v(y): y \in N]$
\[
D_x(y) = \min\{c(x,y) + D_y(y), c(x,z) + D_z(y)\} \\
= \min\{2+0, 7+1\} = 2
\]

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

**node x table**

<table>
<thead>
<tr>
<th></th>
<th>x</th>
<th>y</th>
<th>z</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>0</td>
<td>2</td>
<td>7</td>
</tr>
<tr>
<td>y</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>z</td>
<td>8</td>
<td>8</td>
<td>8</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>x</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>y</td>
<td>0</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>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>x</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>y</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>z</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

Network Layer 4-225
VC implementation

A VC consists of:

1. Path from source to destination
2. VC numbers, one number for each link along path
3. Entries in forwarding tables in routers along path

- Packet belonging to VC carries a VC number.
- VC number must be changed on each link.
  - New VC number comes from forwarding table
# Forwarding table

## Forwarding table in northwest router:

<table>
<thead>
<tr>
<th>Incoming interface</th>
<th>Incoming VC #</th>
<th>Outgoing interface</th>
<th>Outgoing VC #</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>12</td>
<td>3</td>
<td>22</td>
</tr>
<tr>
<td>2</td>
<td>63</td>
<td>1</td>
<td>18</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>2</td>
<td>17</td>
</tr>
<tr>
<td>1</td>
<td>97</td>
<td>3</td>
<td>87</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Routers maintain connection state information!
### 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>

4 billion possible entries
# Longest prefix matching

<table>
<thead>
<tr>
<th>Prefix Match</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?
RIP Table example (continued)

Router: giroflee.eurocom.fr

<table>
<thead>
<tr>
<th>Destination</th>
<th>Gateway</th>
<th>Flags</th>
<th>Ref</th>
<th>Use</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>127.0.0.1</td>
<td>127.0.0.1</td>
<td>UH</td>
<td>0</td>
<td>26492</td>
<td>lo0</td>
</tr>
<tr>
<td>192.168.2.</td>
<td>192.168.2.5</td>
<td>U</td>
<td>2</td>
<td>13</td>
<td>fa0</td>
</tr>
<tr>
<td>193.55.114.</td>
<td>193.55.114.6</td>
<td>U</td>
<td>3</td>
<td>58503</td>
<td>le0</td>
</tr>
<tr>
<td>192.168.3.</td>
<td>192.168.3.5</td>
<td>U</td>
<td>2</td>
<td>25</td>
<td>qaa0</td>
</tr>
<tr>
<td>224.0.0.0</td>
<td>193.55.114.6</td>
<td>U</td>
<td>3</td>
<td>0</td>
<td>le0</td>
</tr>
<tr>
<td>default</td>
<td>193.55.114.129</td>
<td>UG</td>
<td>0</td>
<td>143454</td>
<td></td>
</tr>
</tbody>
</table>

- Three attached class C networks (LANs)
- Router only knows routes to attached LANs
- Default router used to “go up”
- Route multicast address: 224.0.0.0
- Loopback interface (for debugging)
Hierarchical routing

- Unused slides
BGP route selection

- Router may learn about more than 1 route to some prefix. Router must select route.
- Elimination rules:
  1. Local preference value attribute: policy decision, hot potato routing
  2. Shortest AS-PATH
  3. Closest NEXT-HOP router
  4. Additional criteria
Path attributes & BGP routes

- When advertising a prefix, advert includes BGP attributes.
  - prefix + attributes = “route”
- Two important attributes:
  - AS-PATH: contains the ASs through which the advert for the prefix passed: AS 67 AS 17
  - NEXT-HOP: Indicates the specific internal-AS router to next-hop AS. (There may be multiple links from current AS to next-hop-AS.)
- When gateway router receives route advert, uses import policy to accept/decline.
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
Inter-AS tasks

- Suppose router in AS1 receives datagram for which dest is outside of AS1
  - Router should forward packet towards one of the gateway routers, 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!
Example: Setting forwarding table in router 1d

- Suppose AS1 learns from the inter-AS protocol that subnet x is reachable from AS3 (gateway 1c) but not from AS2.
- Inter-AS protocol propagates reachability info to all internal routers.
- Router 1d determines from intra-AS routing info that its interface I is on the least cost path to 1c.
- Puts in forwarding table entry (x,I).
Example: Choosing among multiple ASes

- Now suppose AS1 learns from the inter-AS protocol that subnet x is reachable from AS3 and from AS2.
- To configure forwarding table, router 1d must determine towards which gateway it should forward packets for dest x.
- This is also the job on inter-AS routing protocol!
- Hot potato routing: send packet towards closest of two routers.

Learn from inter-AS protocol that subnet x is reachable via multiple gateways

Use routing info from intra-AS protocol to determine costs of least-cost paths to each of the gateways

Hot potato routing: Choose the gateway that has the smallest least cost

Determine from forwarding table the interface I that leads to least-cost gateway. Enter (x,I) in forwarding table
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What’s inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Router Architecture Overview

Two key router functions:

- **Routing**
  - Determine route taken by packets from source to destination
  - Run protocol (RIP, OSPF, BGP)
    - Generate forwarding table from routing algorithms
    - Algorithms based on either (LS,DV)

- **Forwarding**
  - Process of moving packets from input port to output port
  - Lookup forwarding table given information in packet
  - Switch/forward datagrams from incoming to outgoing link based on route
What Does a Router Look Like?

- Routing processor/controller
  - Handles routing protocols, error conditions
- Line cards
  - Network interface cards
- Forwarding engine
  - Fast path routing (hardware vs. software)
- Backplane
  - Switch or bus interconnect
Typical mode of operation

- Packet arrives at inbound line card
- Header transferred to forwarding engine
- Forwarding engine determines output interface given a table initialized by routing processor
- Forwarding engine signals result to line card
- Packet copied to outbound line card
Routing Processor

- Runs routing protocol
- Uploads forwarding table to forwarding engines
  - Forwarding engines with two forwarding tables to allow easy switchover (double buffering)
- Typically performs “slow-path” processing
  - ICMP error messages
  - IP option processing
  - IP fragmentation
  - IP multicast packets
Input Port Functions

Decentralized switching:
- given datagram dest., lookup output port using forwarding table in input port memory
- 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

see chapter 5
**Input Port Queuing**

- Fabric slower than input ports combined => queuing may occur at input queues
- Head-of-the-Line (HOL) blocking: queued datagram at front of queue prevents others in queue from moving forward
- queueing delay and loss due to input buffer overflow!

![Diagram](image)

- Output port contention at time t - only one red packet can be transferred
- Green packet experiences HOL blocking
Input Port Queuing

- Possible solution
  - Virtual output buffering
    - Maintain per output buffer at input
    - Solves head of line blocking problem
    - Each of MxN input buffer places bid for output
Forwarding Engine

- Two major components
  - Lookup logic/software
    - Data structures and algorithms to lookup route table
    - See previous section on IP route lookup
  - Caches
    - Small, fast memory storing recent lookups
  - Alternatives
    - Hardware-support
    - Hints
Caches

- Leverage temporal locality
- Many packets to same destination
  - Long flows help, short flows do not
- Similar to idea behind IP switching (ATM/MPLS) where long-lived flows map into single label
- Example
  - 8KB L1 Icache
    - Holds full forwarding code
  - 96KB L2 cache
    - Forwarding table cache
  - 16MB L3 cache
    - Full forwarding table x 2 - double buffered for updates
Alternatives

- Lookup via content addressable memory (CAM)
  - Hardware based route lookup
  - Input = tag, output = value associated with tag
  - Requires exact match with tag
    - Multiple cycles (1 per prefix length searched) with single CAM
    - Multiple CAMs (1 per prefix) searched in parallel
  - Ternary CAM
    - 0,1,don’t care values in tag match
    - Priority (i.e. longest prefix) by order of entries in CAM

- “Spatial caching” via protocol acceleration
  - Add clue (5 bits) to IP header
  - Indicate where IP lookup ended on previous node (Bremler-Barr SIGCOMM 99)
Types of network switching fabrics

- Memory
- Multistage interconnection
- Crossbar interconnection
- Bus
Types of network switching fabrics

- Issues
  - Switch contention
    - Packets arrive faster than switching fabric can switch
    - Speed of switching fabric versus line card speed determines input queuing vs. output queuing
Switching Via Memory

First generation routers:
- packet copied by system’s (single) CPU
- 2 bus crossings per datagram
- speed limited by memory bandwidth

Second generation routers:
- input port processor performs lookup, copy into memory
- Cisco Catalyst 8500

Diagram:
- Input Port
- Memory
- Output Port
- System Bus

Network Layer 4-251
Switching Via Bus

- Datagram from input port memory directly to output port memory via a shared bus

- Issues
  - Bus contention: switching speed limited by bus bandwidth

- Examples
  - 1 Gbps bus, Cisco 1900: sufficient speed for access and enterprise routers (not regional or backbone)
Switching Via An Interconnection Network

- **Overcome bus bandwidth limitations**
- **Crossbar networks**
  - Fully connected (n² elements)
  - All one-to-one, invertible permutations supported
- **Issues**
  - Crossbar with N² elements hard to scale
Switching Via An Interconnection Network

- Multi-stage interconnection networks (Banyan)
  - Initially developed to connect processors in multiprocessor
  - Typically \( O(n \log n) \) elements
  - Datagram fragmented fixed length cells, switched through the fabric

- Issues
  - Blocking (not all one-to-one, invertible permutations supported)

- Example
  - Cisco 12000: Gbps through an interconnection network
Output Ports

- Output contention
  - Datagrams arrive from fabric faster than output port's transmission rate
  - Buffering required
  - Scheduling discipline chooses among queued datagrams for transmission
Output port queueing

- buffering when arrival rate via switch exceeds output line speed
- queueing (delay) and loss due to output port buffer overflow!
Chapter 4: Network Layer

- 4.1 Introduction
- 4.2 Virtual circuit and datagram networks
- 4.3 What's inside a router
- 4.4 IP: Internet Protocol
  - Datagram format
  - IPv4 addressing
  - ICMP
  - IPv6
- 4.5 Routing algorithms
  - Link state
  - Distance Vector
  - Hierarchical routing
- 4.6 Routing in the Internet
  - RIP
  - OSPF
  - BGP
- 4.7 Broadcast and multicast routing
Broadcast Routing

- Deliver packets from source to all other nodes
- Source duplication is inefficient:

Source duplication: how does source determine recipient addresses?
In-network duplication

- Flooding: when node receives brdcst pckt, sends copy to all neighbors
  - Problems: cycles & broadcast storm
- Controlled flooding: node only brdcsts pkt if it hasn’t brdcst same packet before
  - Node keeps track of pckt ids already brdcsted
  - Or reverse path forwarding (RPF): only forward pckt if it arrived on shortest path between node and source

- Spanning tree
  - No redundant packets received by any node
Spanning Tree

- First construct a spanning tree
- Nodes forward copies only along spanning tree

(a) Broadcast initiated at A
(b) Broadcast initiated at D
Spanning Tree: Creation

- Center node
- Each node sends unicast join message to center node
  - Message forwarded until it arrives at a node already belonging to spanning tree

(a) Stepwise construction of spanning tree
(b) Constructed spanning tree
Multicast Routing: Problem Statement

- **Goal:** find a tree (or trees) connecting routers having local mcast group members
  - **tree:** not all paths between routers used
  - **source-based:** different tree from each sender to rcvrs
  - **shared-tree:** same tree used by all group members
Approaches for building mcast trees

Approaches:
- **source-based tree**: one tree per source
  - shortest path trees
  - reverse path forwarding
- **group-shared tree**: group uses one tree
  - minimal spanning (Steiner)
  - center-based trees

...we first look at basic approaches, then specific protocols adopting these approaches
Shortest Path Tree

- mcast forwarding tree: tree of shortest path routes from source to all receivers
  - Dijkstra’s algorithm

LEGEND:
- router with attached group member
- router with no attached group member
- link used for forwarding, i indicates order link added by algorithm
Reverse Path Forwarding

- rely on router’s knowledge of unicast shortest path from it to sender
- each router has simple forwarding behavior:

```plaintext
if (mcast datagram received on incoming link on shortest path back to center)
    then flood datagram onto all outgoing links
else ignore datagram
```
Reverse Path Forwarding: example

- result is a source-specific reverse SPT
  - may be a bad choice with asymmetric links
Reverse Path Forwarding: pruning

- forwarding tree contains subtrees with no mcast group members
  - no need to forward datagrams down subtree
  - "prune" msgs sent upstream by router with no downstream group members

**LEGEND**
- router with attached group member
- router with no attached group member
- prune message
- links with multicast forwarding
Shared-Tree: Steiner Tree

- **Steiner Tree**: minimum cost tree connecting all routers with attached group members
- Problem is NP-complete
- Excellent heuristics exist
- Not used in practice:
  - Computational complexity
  - Information about entire network needed
  - Monolithic: rerun whenever a router needs to join/leave
Center-based trees

- single delivery tree shared by all
- one router identified as "center" of tree

To join:
- edge router sends unicast join-msg addressed to center router
- join-msg "processed" by intermediate routers and forwarded towards center
- join-msg either hits existing tree branch for this center, or arrives at center
- path taken by join-msg becomes new branch of tree for this router
Center-based trees: an example

Suppose R6 chosen as center:

LEGEND
- router with attached group member
- router with no attached group member
- path order in which join messages generated
Internet Multicasting Routing: DVMRP

- **DVMRP**: distance vector multicast routing protocol, RFC1075
- **flood and prune**: reverse path forwarding, source-based tree
  - RPF tree based on DVMRP’s own routing tables constructed by communicating DVMRP routers
  - no assumptions about underlying unicast
  - initial datagram to mcast group flooded everywhere via RPF
  - routers not wanting group: send upstream prune msgs
DVMRP: continued...

- **soft state:** DVMRP router periodically (1 min.) "forgets" branches are pruned:
  - mcast data again flows down unpruned branch
  - downstream router: reprune or else continue to receive data

- routers can quickly regraft to tree
  - following IGMP join at leaf

- odds and ends
  - commonly implemented in commercial routers
  - Mbone routing done using DVMRP
Tunneling

**Q:** How to connect “islands” of multicast routers in a “sea” of unicast routers?

- mcast datagram encapsulated inside “normal” (non-multicast-addressed) datagram
- normal IP datagram sent thru “tunnel” via regular IP unicast to receiving mcast router
- receiving mcast router unencapsulates to get mcast datagram
PIM: Protocol Independent Multicast

- not dependent on any specific underlying unicast routing algorithm (works with all)
- two different multicast distribution scenarios:

  **Dense:**
  - group members densely packed, in “close” proximity.
  - bandwidth more plentiful

  **Sparse:**
  - # networks with group members small wrt # interconnected networks
  - group members “widely dispersed”
  - bandwidth not plentiful
Consequences of Sparse-Dense Dichotomy:

<table>
<thead>
<tr>
<th>Dense</th>
<th>Sparse</th>
</tr>
</thead>
<tbody>
<tr>
<td>- group membership by routers assumed until routers explicitly prune</td>
<td>- no membership until routers explicitly join</td>
</tr>
<tr>
<td>- data-driven construction on mcast tree (e.g., RPF)</td>
<td>- receiver-driven construction of mcast tree (e.g., center-based)</td>
</tr>
<tr>
<td>- bandwidth and non-group-router processing profligate</td>
<td>- bandwidth and non-group-router processing conservative</td>
</tr>
</tbody>
</table>
PIM- Dense Mode

flood-and-prune RPF, similar to DVMRP but

- underlying unicast protocol provides RPF info for incoming datagram
- less complicated (less efficient) downstream flood than DVMRP reduces reliance on underlying routing algorithm
- has protocol mechanism for router to detect it is a leaf-node router
**PIM - Sparse Mode**

- center-based approach
- router sends join msg to rendezvous point (RP)
  - intermediate routers update state and forward join
- after joining via RP, router can switch to source-specific tree
  - increased performance: less concentration, shorter paths

![Diagram of PIM Sparse Mode]
PIM - Sparse Mode

sender(s):
- unicast data to RP, which distributes down RP-rooted tree
- RP can extend mcast tree upstream to source
- RP can send stop msg if no attached receivers
  - “no one is listening!”
NL: Advanced topics

- Routing synchronization
- Routing instability
- Routing metrics
- Overlay networks
- Routing alternatives: Landmark routing
Dynamic robustness issue to consider...

- Intuitive assumption that independent streams will not synchronize is not always valid
- Abrupt transition from unsynchronized to synchronized system states
NL: How Synchronization Occurs

Weak coupling can result in eventual synchronization.

Weak Coupling when A’s behavior is triggered off of B’s message arrival!
NL: Examples/Sources of Synchronization

- TCP congestion window behavior
- Periodic transmission by audio/video applications
- Synchronized client restart
- Routing
  - Periodic routing protocol messages from different routers
  - Lots of this in initial routing protocols....
**NL: Routing Source of Synchronization**

- Router resets timer after processing its own and incoming updates
- Creates weak coupling among routers

**Solutions**

- Set timer based on clock event that is not a function of processing other routers’ updates, or
- Add randomization, or reset timer before processing update
  - With increasing randomization, **abrupt** transition from predominantly synchronized to predominantly unsynchronized
  - Most protocols now incorporate some form of randomization
NL: Routing Instability

References


Record of BGP messages at major exchanges

Discovered orders of magnitude larger than expected updates

• Bulk were duplicate withdrawals
  • Stateless implementation of BGP - did not keep track of information passed to peers
  • Impact of few implementations

• Strong frequency (30/60 sec) components
  • Interaction with other local routing/links etc.
NL: Route Flap Storm

- Overloaded routers fail to send Keep_Alive message and marked as down
- BGP peers find alternate paths
- Overloaded router re-establishes peering session
- Must send large updates
- Increased load causes more routers to fail!
NL: Route Flap Dampening

- Routers now give higher priority to BGP/Keep Alive to avoid problem
- Associate a penalty with each route on change
  - Increase when route flaps
  - Exponentially decay penalty with time
- When penalty reaches threshold, suppress route
NL: Overlay Routing

- **Basic idea:**
  - Treat multiple hops through IP network as one hop in an overlay network
  - Run routing protocol on overlay nodes

- **Why?**
  - For performance - can run more clever protocol on overlay
  - For efficiency - can make core routers very simple
  - For functionality - can provide new features such as multicast, active processing, IPv6
References
- Savage et. al. “The End-to-End Effects of Internet Path Selection”, SIGCOMM 99
- Anderson et. al. “Resilient Overlay Networks”, SOSP 2001

Why would IP routing not give good performance?
- Policy routing - limits selection/advertisement of routes
- Early exit/hot-potato routing - local not global incentives
- Lack of performance based metrics - AS hop count is the wide area metric

How bad is it really?
- Look at performance gain an overlay provides
NL: Quantifying Performance Loss

- Measure round trip time (RTT) and loss rate between pairs of hosts
  - ICMP rate limiting
- Alternate path characteristics
  - 30-55% of hosts had lower latency
  - 10% of alternate routes have 50% lower latency
  - 75-85% have lower loss rates
NL: Bandwidth Estimation

- RTT & loss for multi-hop path
  - RTT by addition
  - Loss either worst or combine of hops – why?
    - Large number of flows → combination of probabilities
    - Small number of flows → worst hop

- Bandwidth calculation
  - TCP bandwidth is based primarily on loss and RTT

- 70-80% paths have better bandwidth
- 10-20% of paths have 3x improvement
NL: Overlay for Efficiency

- Multi-path routing
  - More efficient use of links or QOS
  - Need to be able to direct packets based on more than just destination address \(\rightarrow\) can be computationally expensive
  - What granularity? Per source? Per connection? Per packet?
    - Per packet \(\rightarrow\) re-ordering
    - Per source, per flow \(\rightarrow\) coarse grain vs. fine grain
  - Take advantage of relative duration of flows
    - Most bytes on long flows
**NL: Overlay for Features**

- How do we add new features to the network?
  - Does every router need to support new feature?
  - **Choices**
    - Reprogram all routers → active networks
    - Support new feature within an overlay
  - **Basic technique: tunnel packets**

- **Tunnels**
  - IP-in-IP encapsulation
  - Poor interaction with firewalls, multi-path routers, etc.
**NL: Examples**

- **IP V6 & IP Multicast**
  - Tunnels between routers supporting feature

- **Mobile IP**
  - Home agent tunnels packets to mobile host's location

- **QOS**
  - Needs some support from intermediate routers
NL: Overlay Challenges

- How do you build efficient overlay
  - Probably don’t want all $N^2$ links - which links to create?
  - Without direct knowledge of underlying topology how to know what’s nearby and what is efficient?
NL: Future of Overlay

- Application specific overlays
  - Why should overlay nodes only do routing?
- Caching
  - Intercept requests and create responses
- Transcoding
  - Changing content of packets to match available bandwidth
- Peer-to-peer applications
**NL: Routing alternatives: Landmark routing**

- Details about things nearby and less information about things far away
- Not defined by arbitrary boundaries
  - Thus, not well suited to the real world that does have administrative boundaries
- **Example: My apartment**
  - From Beaverton
    - Go towards Mt. Hood
    - See USBancorpTower before running into Mt.Hood
    - See PearlDistrict before running into USBancorpTower
    - Reach PearlDistrict and route to Kearney Plaza 2 blocks away
  - From The Dalles
    - Go towards Mt. Hood, reach it
    - Go towards Portland, see USBancorpTower
    - Go towards and reach USBancorpTower
    - Go towards and reach PearlDistrict, route to Kearney Plaza 2 blocks away
Router 1 is a landmark of radius 2
NL: Landmark Overview

- Landmark routers have “height” which determines how far away they can be seen (visibility)
  - Routers within radius n can see a landmark router LM_n
    - See = routers have LM_n’s address and know next hop to reach it.
  - Router x as an entry for router y if x is within radius of y
  - Routing table: Landmark (LM_2(d)), Level(2), Next hop

- Intuition
  - Everyone knows how to get to the highest landmark (level N)
  - Highest landmark knows how to get you to any landmark at level N-1 (i.e. the N-1 level landmark that matches your destination)
  - That level N-1 landmark, knows how to get you to your level N-2, etc.
  - Along the way, you may find a router that lets you short-circuit path to higher landmarks and take you to destination
NL: LM Hierarchy Definition

- Each LM \( i \) associated with level (i) and radius (\( r_i \))
- Every node is an LM\(_0\) landmark
- Recursion: some LM\(_i\) are also LM\(_{i+1}\)
  - Every LM\(_i\) sees at least one LM\(_{i+1}\)
- Terminating state when all level j LMs are seen by entire network
**NL: LM Self-configuration**

- **Bottom-up hierarchy construction algorithm**
  - Every router is $L_0$ landmark
  - All $L_i$ landmarks run election to self-promote one or more $L_{i+1}$ landmarks

- **LM level maps to radius (part of configuration), e.g.:**
  - LM level 0: radius 2
  - LM level 1: radius 4
  - LM level 2: radius 8

- **Dynamic algorithm to adapt to topology changes** - Efficient hierarchy in terms of storage required
NL: LM Addresses

- LM(2).LM(1).LM(0) (C.B.A)
- If destination is far away, will not have complete routing information, refer to LM(1) portion of address, if not known then refer to LM(2)
NL: LM Routing

- LM does not imply hierarchical forwarding
  - En route to $LM_n$, packet may encounter router that is within $LM_0$ radius of destination address (like longest match)
- NOT a source route
- Paths may be asymmetric
**NL: Landmark Routing: Basic Operation**

- Source wants to reach $LM_0[a]$, whose address is $c.b.a$:
  - Source can see $LM_2[c]$, so sends packet towards $c$
  - Entering $LM_1[b]$ area, first router diverts packet to $b$
  - Entering $LM_0[a]$ area, packet delivered to $a$

- Not shortest path
- Packet may not reach landmarks
NL: Landmark Routing: Example
**NL: Routing Table for Router g**

<table>
<thead>
<tr>
<th>Landmark</th>
<th>Level</th>
<th>Next hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>LM₂[d]</td>
<td>2</td>
<td>f</td>
</tr>
<tr>
<td>LM₁[i]</td>
<td>1</td>
<td>k</td>
</tr>
<tr>
<td>LM₀[e]</td>
<td>0</td>
<td>f</td>
</tr>
<tr>
<td>LM₀[k]</td>
<td>0</td>
<td>k</td>
</tr>
<tr>
<td>LM₀[f]</td>
<td>0</td>
<td>f</td>
</tr>
</tbody>
</table>

r₀ = 2, r₁ = 4, r₂ = 8 hops

- How to go from d.i.g to d.n.t? g-f-e-d-u-t
- How does path length compare to shortest path? g-k-l-u-t
NL: Network layer summary

- Network layer functions
- Specific network layers (IPv4, IPv6)
- Specific network layer devices (routers)
- Advanced network layer topics
Issues with Multi-homing

- Symmetric routing
  - While preference symmetric paths, many are asymmetric

- Packet re-ordering
  - May trigger TCP’s fast retransmit algorithm

- Other concerns:
  - Addressing, DNS, aggregation
Multi-homing to a Single Provider

- Easy solution:
  - Use IMUX or Multi-link PPP

- Hard solution:
  - Use BGP
  - Makes assumptions about traffic (same amount of prefixes can be reached from both links)
Multi-homing to a Single Provider

- If multiple prefixes, may use MED
  - Good if traffic load to prefixes is equal
- If single prefix, load may be unequal
  - Break-down prefix and advertise different prefixes over different links

```
ISP
R1
R2
Customer
138.39/16
R3
204.70/16
```
Multi-homing to a Single Provider

- For traffic to customer, same as before:
  - Use MED
  - Good if traffic load to prefixes is equal
- For traffic to ISP:
  - R3 alternates links
  - Multiple default routes
Multi-homing to a Single Provider

- Most reliable approach
  - No equipment sharing
- Use MED

```
ISP

R1  R2

R3  R4

Customer

138.39/16 204.70/16
```
Outline

- External BGP (E-BGP)
- Internal BGP (I-BGP)
- Multi-Homing
- Stability Issues
Multi-homing

- With multi-homing, a single network has more than one connection to the Internet.

- Improves reliability and performance:
  - Can accommodate link failure
  - Bandwidth is sum of links to Internet

- Challenges
  - Getting policy right (MED, etc..)
  - Addressing
Multi-homing to Multiple Providers

- **Major issues:**
  - Addressing
  - Aggregation

- **Customer address space:**
  - Delegated by ISP1
  - Delegated by ISP2
  - Delegated by ISP1 and ISP2
  - Obtained independently
Address Space from one ISP

- Customer uses address space from ISP1
- ISP1 advertises /16 aggregate
- Customer advertises /24 route to ISP2
- ISP2 relays route to ISP1 and ISP3
- ISP2-3 use /24 route
- ISP1 routes directly
- Problems with traffic load?
Pitfalls

- ISP1 aggregates to a /19 at border router to reduce internal tables.
- ISP1 still announces /16.
- ISP1 hears /24 from ISP2.
- ISP1 routes packets for customer to ISP2!
- Workaround: ISP1 must inject /24 into I-BGP.
Address Space from Both ISPs

- ISP1 and ISP2 continue to announce aggregates.
- Load sharing depends on traffic to two prefixes.
- Lack of reliability: if ISP1 link goes down, part of customer becomes inaccessible.
- Customer may announce prefixes to both ISPs, but still problems with longest match as in case 1.
Address Space Obtained Independently

- Offers the most control, but at the cost of aggregation.
- Still need to control paths
Measurement of Real Ethernet

- Evaluate performance in some typical scenarios
  - Scenario 1
    - Topology: 4 clusters of 6 hosts - similar to office configuration
    - Fixed pkt size
    - Throughput decreases with number of hosts & increases with pkt size - as expected
    - Fairness improves with number of hosts - capture effects less likely
    - Only linear increase in delay with number of hosts - unexpected
Measurement of Real Ethernet

- Scenario 2
  - Topology: 23 hosts on short net
  - Load: fixed pkt size
  - Improvement in bit rate over scenario 1
- Scenario 3
  - Topology: 4 clusters
  - Load: bimodal pkt size
  - 7/1 ratio of small to large pkts is sufficient to greatly improve total bit rate
How to Improve Performance

- No long cables
- Fewer hosts per cable
- Use large packets
- Don't mix real-time with bulk-data if possible
- Can't provide good efficiency/throughput and good latency

Ethernet Packet Traces
- Ethernet traffic is “self-similar” (fractal)
- Bursty at every time scale (msecs to months)
- Implication?
  - On average, low load
  - Occasional peaks
***MISC_IP_ROUTING***
Problems

- Routing table size
  - Need an entry for all paths to all networks

- Required memory = $O((N + M*A) * K)$
  - $N$: number of networks
  - $M$: mean AS distance (in terms of hops)
  - $A$: number of AS’s
  - $K$: number of BGP peers
# Routing Table Size

<table>
<thead>
<tr>
<th>Networks</th>
<th>Mean AS Distance</th>
<th>Number of AS’s</th>
<th>BGP Peers/Net</th>
<th>Memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>2,100</td>
<td>5</td>
<td>59</td>
<td>3</td>
<td>27,000</td>
</tr>
<tr>
<td>4,000</td>
<td>10</td>
<td>100</td>
<td>6</td>
<td>108,000</td>
</tr>
<tr>
<td>10,000</td>
<td>15</td>
<td>300</td>
<td>10</td>
<td>490,000</td>
</tr>
<tr>
<td>100,000</td>
<td>20</td>
<td>3,000</td>
<td>20</td>
<td>1,040,000</td>
</tr>
</tbody>
</table>

- Problem reduced with CIDR
Routing Information Bases (RIB)

- Routes are stored in RIBs
- Adj-RIBs-In: routing info that has been learned from other routers (unprocessed routing info)
- Loc-RIB: local routing information selected from Adj-RIBs-In (routes selected locally)
- Adj-RIBs-Out: info to be advertised to peers (routes to be advertised)
BGP Common Header

0 1 2 3

Marker (security and message delineation)
16 bytes

Length (2 bytes)  Type (1 byte)

Types: OPEN, UPDATE, NOTIFICATION, KEEPALIVE
BGP Messages

- Open
  - Announces AS ID
  - Determines hold timer - interval between keep_alive or update messages, zero interval implies no keep_alive

- Keep_alive
  - Sent periodically (but before hold timer expires) to peers to ensure connectivity.
  - Sent in place of an UPDATE message

- Notification
  - Used for error notification
  - TCP connection is closed immediately after notification
**BGP UPDATE Message**

- List of withdrawn routes
- Network layer reachability information
  - List of reachable prefixes
- Path attributes
  - Origin
  - Path
  - Metrics
- All prefixes advertised in message have same path attributes
LOCAL PREF

Local (within an AS) mechanism to provide relative priority among BGP routers.

- R1: Local Pref = 500
- R2: Local Pref = 800
- R3: Local Pref = 500
- R4: Local Pref = 800
- R5

Network Layer
AS PATH

- List of traversed AS's

- AS 200 (170.10.0.0/16)
- AS 300
- AS 500 (180.10.0.0/16)
- AS 100 (180.10.0.0/16)

180.10.0.0/16 300 200 100
170.10.0.0/16 300 200
**CIDR and BGP**

What should T announce to Z?
Options

- Advertise all paths:
  - Path 1: through T can reach 197.8.0.0/23
  - Path 2: through T can reach 197.8.2.0/24
  - Path 3: through T can reach 197.8.3.0/24

- But this does not reduce routing tables!
  We would like to advertise:
  - Path 1: through T can reach 197.8.0.0/22
Sets and Sequences

Problem: what do we list in the route?
- List T: omitting information not acceptable, may lead to loops
- List T, X, Y: misleading, appears as 3-hop path

Solution: restructure AS Path attribute as:
- Path: (Sequence (T), Set (X, Y))
- If Z wants to advertise path:
  - Path: (Sequence (Z, T), Set (X, Y))
- In practice used only if paths in set have same attributes
Multi-Exit Discriminator (MED)

- Hint to external neighbors about the preferred path into an AS
  - Non-transitive attribute (we will see later why)
  - Different AS choose different scales
- Used when two AS’s connect to each other in more than one place
**MED**

- **Hint to R1 to use R3 over R4 link**
- **Cannot compare AS40’s values to AS30’s**
MED

- MED is typically used in provider/subscriber scenarios
- It can lead to unfairness if used between ISP because it may force one ISP to carry more traffic:

  - ISP1 ignores MED from ISP2
  - ISP2 obeys MED from ISP1
  - ISP2 ends up carrying traffic most of the way
Other Attributes

- **ORIGIN**
  - Source of route (IGP, EGP, other)

- **NEXT_HOP**
  - Address of next hop router to use
  - Used to direct traffic to non-BGP router

- Check out [http://www.cisco.com](http://www.cisco.com) for full explanation
Decision Process

- Processing order of attributes:
  - Select route with highest LOCAL-PREF
  - Select route with shortest AS-PATH
  - Apply MED (if routes learned from same neighbor)
Outline

- External BGP (E-BGP)
- Internal BGP (I-BGP)
- Multi-Homing
- Stability Issues
Internal vs. External BGP

- BGP can be used by R3 and R4 to learn routes
- How do R1 and R2 learn routes?
- Option 1: Inject routes in IGP
  - Only works for small routing tables
- Option 2: Use I-BGP
Internal BGP (I-BGP)

- Same messages as E-BGP
- Different rules about re-advertising prefixes:
  - Prefix learned from E-BGP can be advertised to I-BGP neighbor and vice-versa, but
  - Prefix learned from one I-BGP neighbor cannot be advertised to another I-BGP neighbor
  - Reason: no AS PATH within the same AS and thus danger of looping.
Internal BGP (I-BGP)

• R3 can tell R1 and R2 prefixes from R4
• R3 can tell R4 prefixes from R1 and R2
• R3 cannot tell R2 prefixes from R1

R2 can only find these prefixes through a direct connection to R1
Result: I-BGP routers must be fully connected (via TCP)!
  • contrast with E-BGP sessions that map to physical links


## Link Failures

- Two types of link failures:
  - Failure on an E-BGP link
  - Failure on an I-BGP Link

- These failures are treated completely different in BGP

- Why?
Failure on an E-BGP Link

- If the link R1-R2 goes down
  - The TCP connection breaks
  - BGP routes are removed
- This is the *desired* behavior
Failure on an I-BGP Link

• If link R1-R2 goes down, R1 and R2 should still be able to exchange traffic
• The indirect path through R3 must be used
• Thus, E-BGP and I-BGP must use different conventions with respect to TCP endpoints

![Diagram showing network topology and IP addresses for R1, R2, and R3 with a dashed line indicating I-BGP connection and a solid line indicating physical link.](attachment:image.png)
Distance Vector in Practice

- RIP and RIP2
  - Uses split-horizon/poison reverse
- BGP
  - Propagates entire path
  - Path also used for effecting policies
Route Prefixes
A 0*
B 01000*
C 011*
D 1*
E 100*
F 1100*
G 1101*
H 1110*
I 1111*

NL: Binary trie