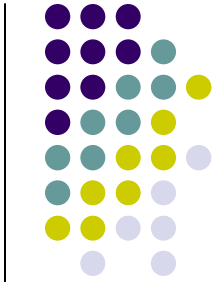
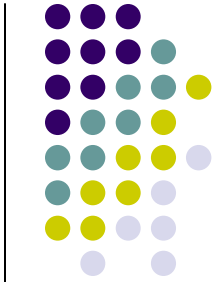
# Lecture 11 : Routing Algos + Scale



Shankar Balachandran
Assistant Professor
Dept. of CSE, IIT Madras

Short Term Course on “Teaching Computer Networks Effectively”. Sponsored by AICTE.

# Overview



- Forwarding vs Routing
  - forwarding: to select an output port based on destination address and routing table
  - routing: process by which routing table is built

**Routing table**

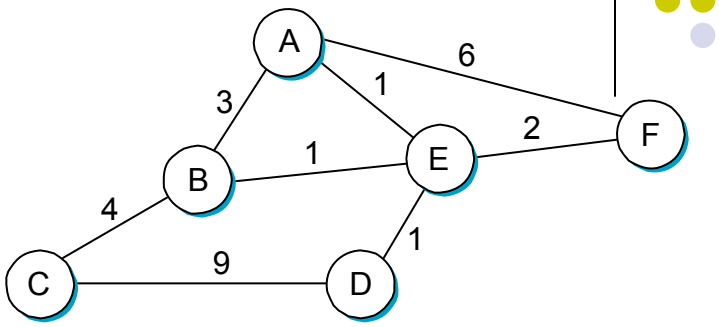
| NetworkNum | NextHop |
|---|---|
| 5 | 192.168.1.1 |

**Forwarding table**

| NetworkNum | Iface | MAC addr |
|---|---|---|
| 10 | eth0 | a:b:c:d:e:f |

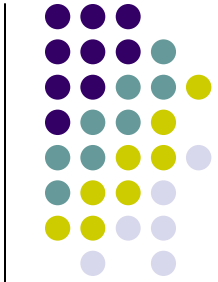
# Overview

- Network as a Graph



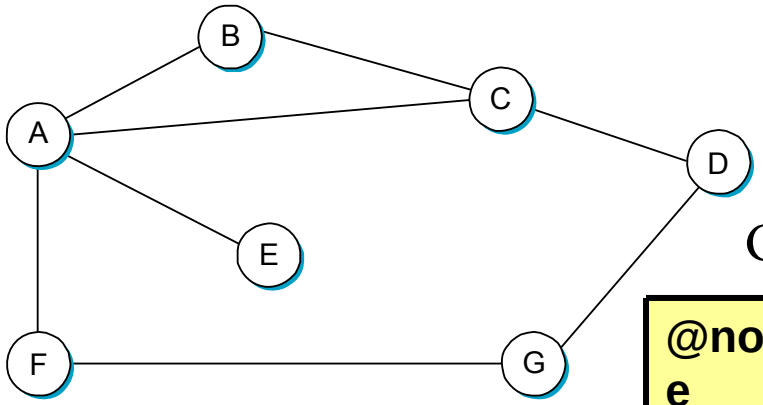

- Problem: Find lowest cost path between every pair of nodes
  - Sounds familiar??
- Factors
  - static: topology
  - dynamic: load

# 1. Distance Vector Routing



- Each node maintains a set of triples
  - **`(Destination, Cost, NextHop)`**
- Directly connected neighbors exchange updates
  - periodically (on the order of several seconds)
  - whenever table changes (called *triggered* update)
- Each update is a list of pairs:
  - **`(Destination, Cost)`**
- Update local table if receive a “better” route
  - smaller cost
  - came from next-hop
- Refresh existing routes; delete if they time out

# Distance Vector




A's view: vector

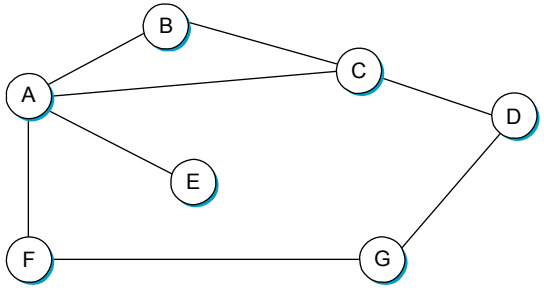
| Dest | Cost | NextHop |
|---|---|---|
| B | 1 | B |
| C | 1 | C |
| D | ∞ | - |
| E | 1 | E |
| F | 1 | F |
| G | ∞ | - |

Global view (Never available at any single node)

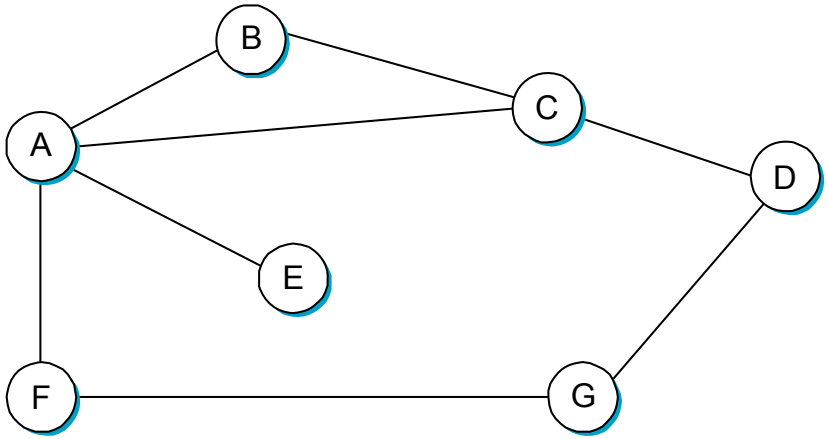
| @node | Distance to node | | | | | | |
|---|---|---|---|---|---|---|---|
| | A | B | C | D | E | F | G |
| A | 0 | 1 | 1 | ∞ | 1 | 1 | ∞ |
| B | 1 | 0 | 1 | ∞ | ∞ | ∞ | ∞ |
| C | 1 | 1 | 0 | 1 | ∞ | ∞ | ∞ |
| D | ∞ | ∞ | 1 | 0 | ∞ | ∞ | 1 |
| E | 1 | ∞ | ∞ | ∞ | 0 | ∞ | ∞ |
| F | 1 | ∞ | ∞ | ∞ | ∞ | 0 | 1 |
| G | ∞ | ∞ | ∞ | 1 | ∞ | 1 | 0 |

# Distance Vector Routing

- Every node starts by building it's own local view of what nodes are 1 hop away.
- Next, every node sends its vector to its directly connected neighbors.
- Example :
  - C tells B it can reach D in one hop
  - B updates D's distance to 2 ( B to C + C to D)
  - If any other path comes along with higher cost, discard
  - After a few iterations, the routing table converges



# Example

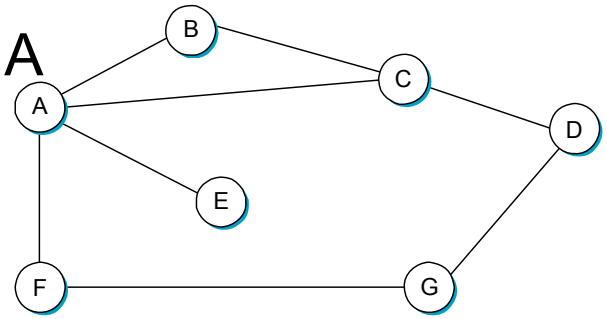



## At B

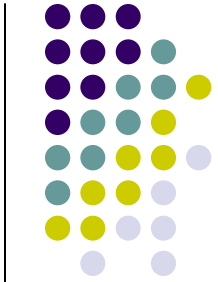
| Destination | Cost | NextHop |
|---|---|---|
| A | 1 | A |
| C | 1 | C |
| D | 2 | C |
| E | 2 | A |
| F | 2 | A |
| G | 3 | A |

# Routing Loops

- Example 1
  - F detects that link to G has failed
  - F sets distance to G to infinity and sends update t o A
  - A sets distance to G to infinity since it uses F to reach G
  - A receives periodic update from C with 2-hop path to G
  - A sets distance to G to 3 and sends update to F
  - F decides it can reach G in 4 hops via A
- Example 2
  - link from A to E fails
  - A advertises distance of infinity to E
  - B and C advertise a distance of 2 to E
  - B decides it can reach E in 3 hops; advertises this to A
  - A decides it can read E in 4 hops; advertises this to C
  - C decides that it can reach E in 5 hops…
  - Count to infinity problem



# Loop-Breaking Heuristics

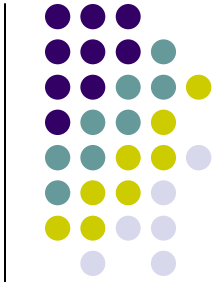


- Set infinity to 16 (or some such value)
  - Any problem with the technique?
- Split horizon
  - When you update a neighbor, do not send a route that you learned from the neighbor
- Split horizon with poison reverse
  - When you update a neighbor, “poison” the routing information that you learnt from the neighbor
    - Set value to infinity
- Split horizon techniques break 2-node loops only

# 2. Link State

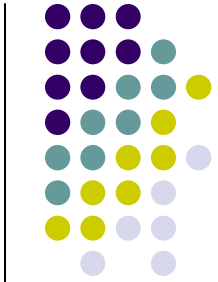
- Assumptions :
  - As before; Links status, cost of links
- Strategy
  - send to all nodes (not just neighbors) information about directly connected links (not entire routing table)
  - every node can reconstruct the graph and hence find the shortest path
  - a sufficient condition (not necessary though) to find shortest path
- Two mechanisms needed
  - Reliable dissemination of link information
    - Reliable flooding
  - Calculation of routes from the link states collected

# Link State Packet (LSP)



- id of the node that created the LSP
  - to know who you got the information from
- cost of link to each directly connected neighbor
- sequence number (SEQNO)
  - recent update overrides older one
- time-to-live (TTL) for this packet
  - old ones must eventually die

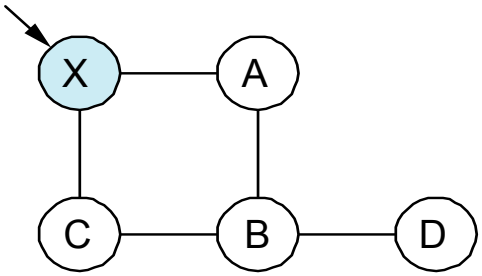
# Link State (cont)



- Reliable flooding
  - store most recent LSP from each node
  - forward LSP to all nodes but one that sent it
  - generate new LSP periodically
    - increment SEQNO
  - start SEQNO at 0 when reboot
  - decrement TTL of each stored LSP
    - discard when TTL=0

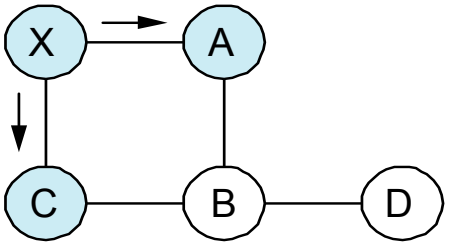
# Example

X gets LSP

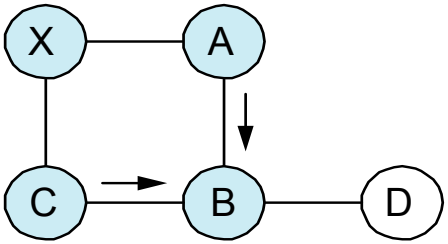



(a)

X floods A and B;
Nothing backwards

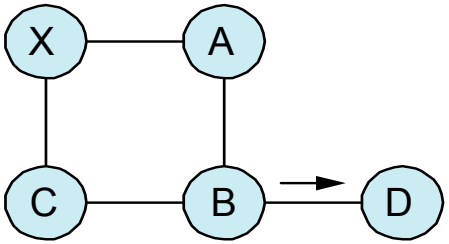



(b)




(c)

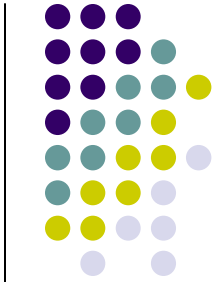
A and C flood B




(d)

Flooding is complete

# Route Calculation



- Dijkstra's shortest path algorithm
- Let
  - *N* denotes set of nodes in the graph
  - *l* (*i*, *j*) denotes non-negative cost (weight) for edge (*i*, *j*) or ∞
  - *s* denotes this node
  - *M* denotes the set of nodes incorporated so far
  - *C*(*n*) denotes cost of the path from *s* to node *n*

```
M = {s}
for each n in N - {s}
    C(n) = l(s, n)
while (N != M)
    M = M union {w} such that C(w) is the minimum
    for
         all w in (N - M)
    for each n in (N - M)
       C(n) = MIN(C(n), C (w) + l(w, n ))
```

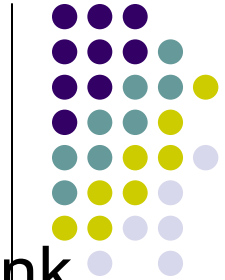
# Properties of Link State Routing

- Stabilizes quickly
- Keeps routing control traffic low
- Responds rapidly to topology changes
- Doesn’t scale well: the amount of information stored in each node is large.

# Difference Between DV and LS

- In DV
  - Talk only to neighbor
  - Tell the neighbor everything that you learnt
- In LS
  - Talk to everyone
  - Tell them about the only thing that you are sure of (namely the neighbor's link state)

# Metrics

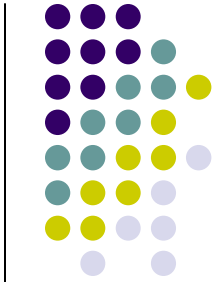


- Original ARPANET metric
  - measures number of packets queued on each link
  - took  neither latency or bandwidth into consideration
- New ARPANET metric
  - stamp each incoming packet with its arrival time (**`AT`**)
  - record departure time (**`DT`**)
  - when link-level ACK arrives, compute

    ```
    Delay = (DT - AT) + Transmit + Latency
    ```

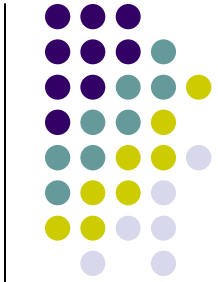
  - if timeout, reset **`DT`** to departure time for retransmission
  - link cost = average delay over some time period
- Fine Tuning
  - compressed dynamic range
  - replaced **`Delay`** with link utilization

# 11.2 How to Make Routing Scale?

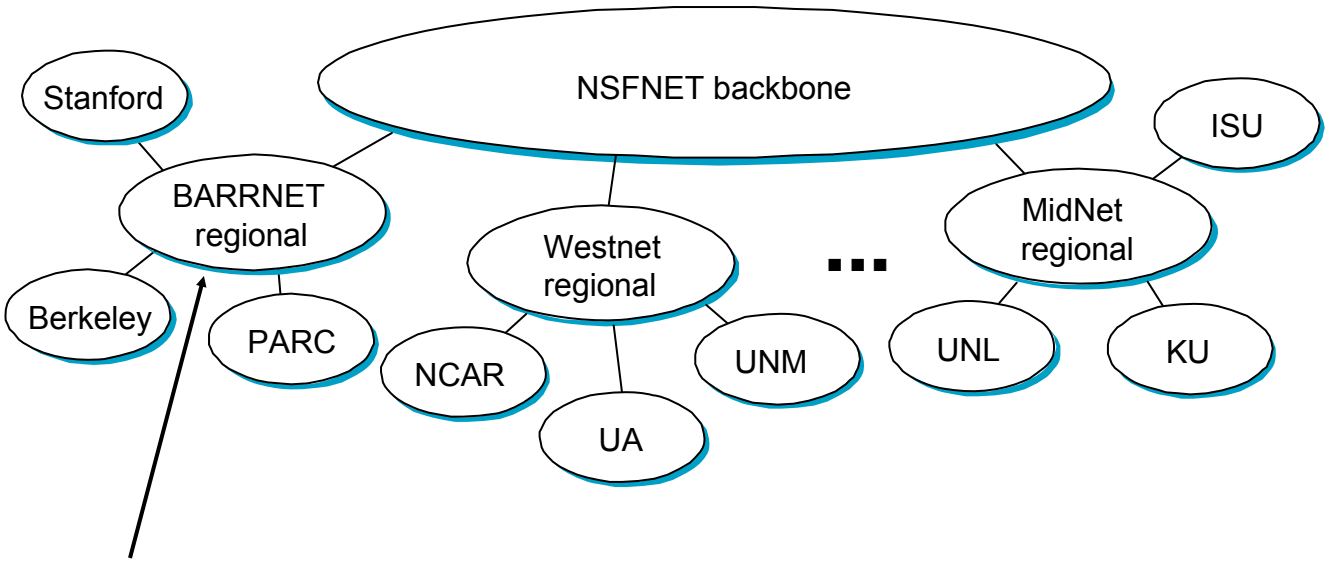


- Flat versus Hierarchical Addresses
- Inefficient use of Hierarchical Address Space
  - class C with 2 hosts (2/255 = 0.78% efficient)
  - class B with 256 hosts (256/65535 = 0.39% efficient)
- Still Too Many Networks
  - routing tables do not scale
  - route propagation protocols do not scale

# Internet Structure

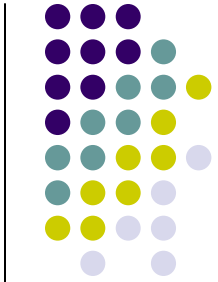


## Recent Past

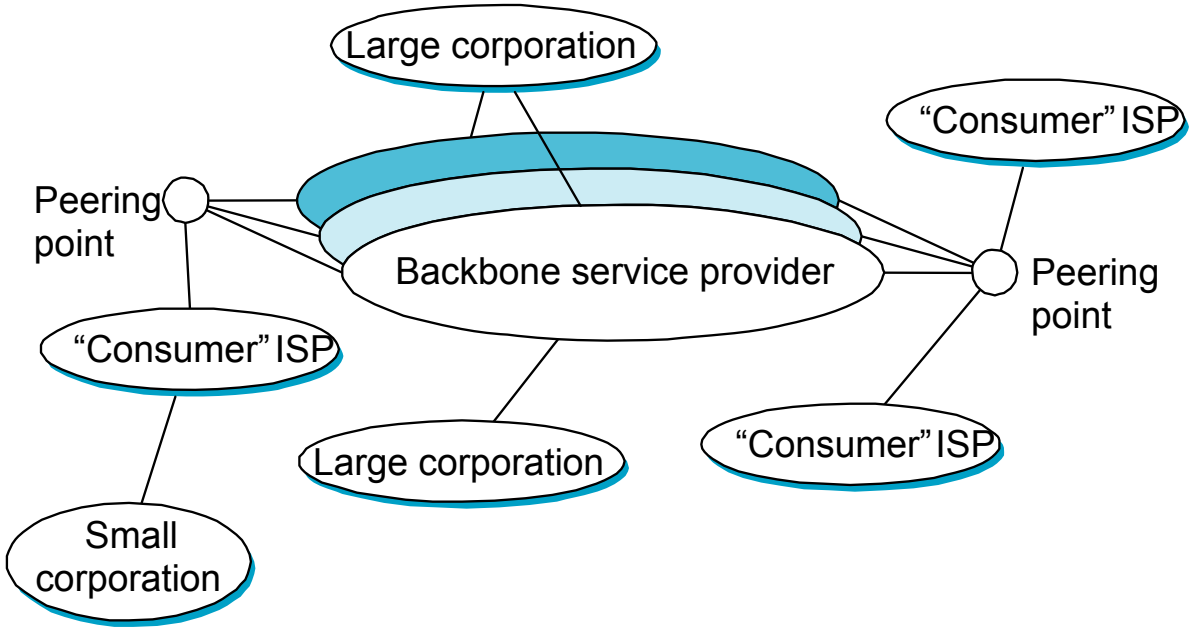



**Autonomous System (AS)**

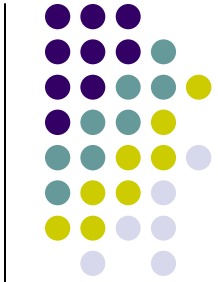
# Internet Structure

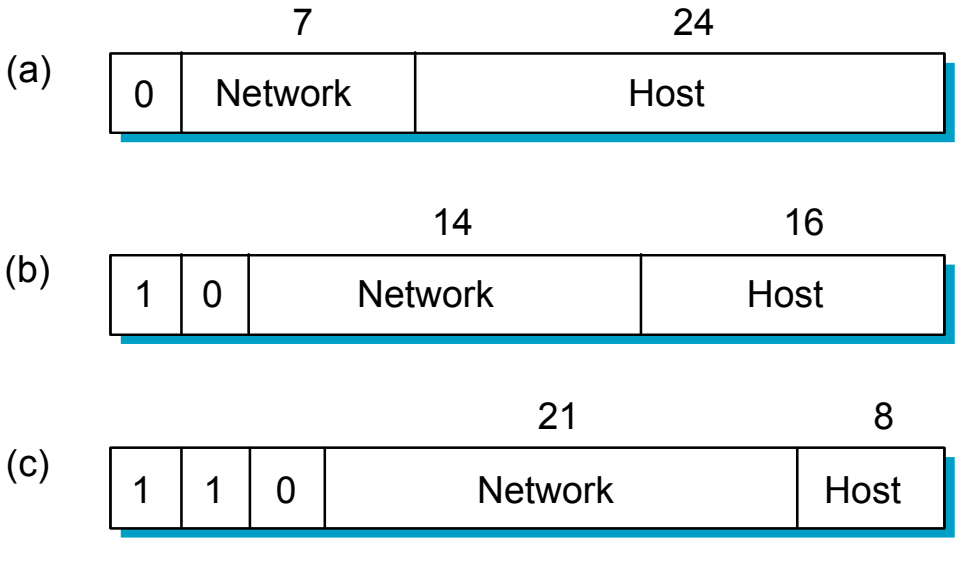


Today



# Global Addresses






**CLASS A**

**CLASS B**

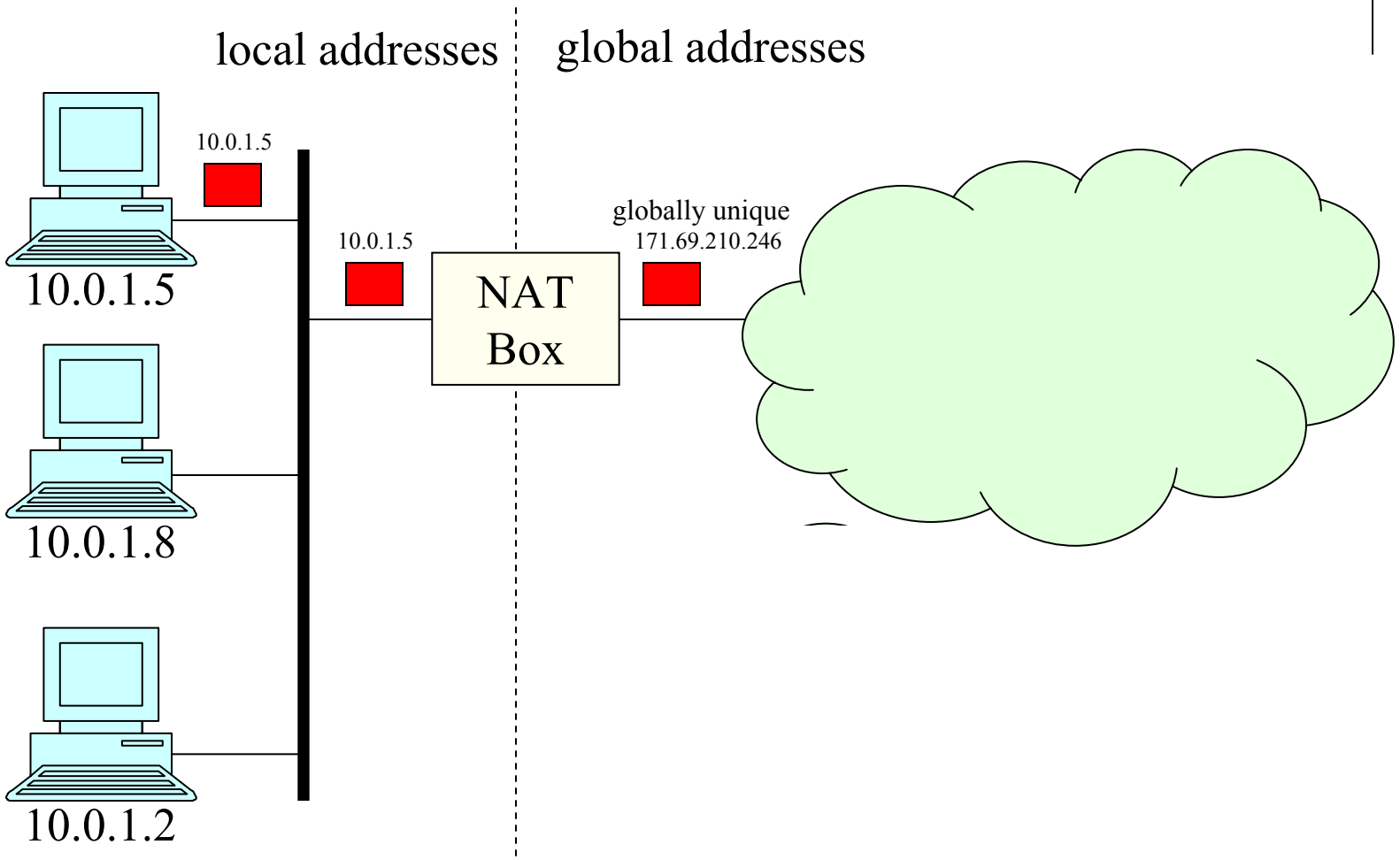
**CLASS C**

- What will you assign if I have 2 hosts on the network?
- What if I have 280 machines that needed global addresses?

# Scalability

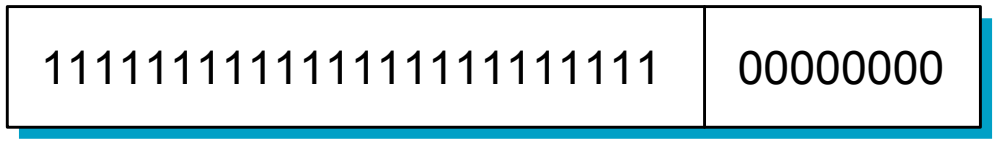
- Routing
  - Minimize the number of network numbers
  - Routing table information reduces
- Address Utilization
  - IP address space should not get used up quickly
- Problem
  - Every network would uniquely identify one physical network
    - 2^14 networks = 16K networks would exhaust Class B addresses

# Network Address Translation

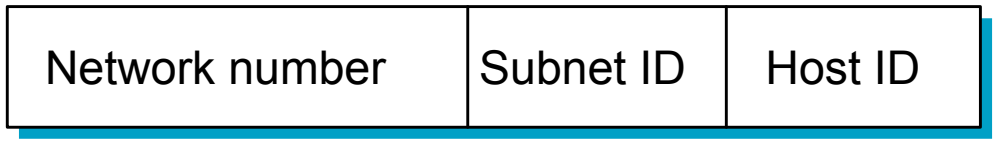


# Subnetting


Class B address




Subnet mask (255.255.255.0)




Subnetted address

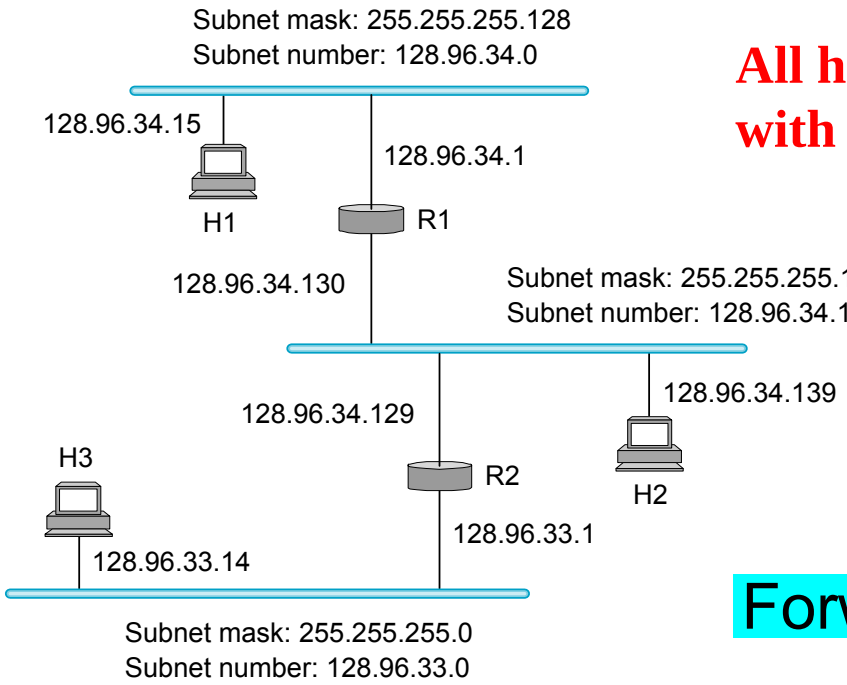
Take a ***network address*** and break it up into subnets that can be assigned to individual physical networks.

Define a ***subnet mask*** to help create a new level of hierarchy in the addressing scheme.

The bitwise AND of the subnet mask with the full address gives the subnet number.

**Added Hierarchy**

# Subnet Example

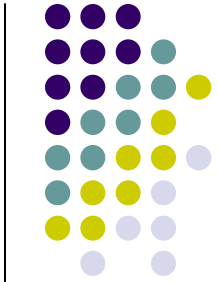



**All hosts in a subnet are configured with the same subnet mask.**

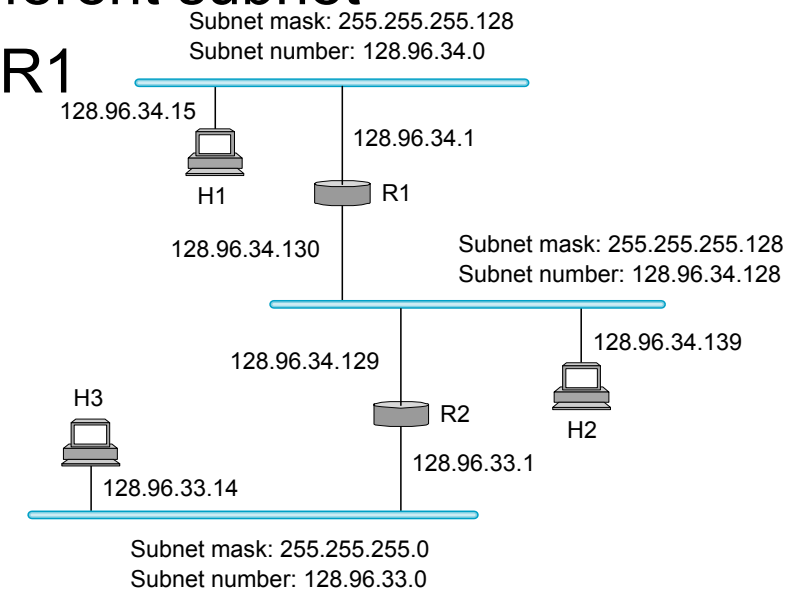
Forwarding table at router R1

| Subnet Number | Subnet Mask | Next Hop |
|---|---|---|
| 128.96.34.0 | 255.255.255.128 | interface 0 |
| 128.96.34.128 | 255.255.255.128 | interface 1 |
| 128.96.33.0 | 255.255.255.0 | R2 |

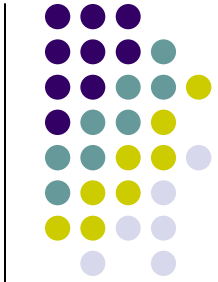
# Example



- H1 wants to send to H2
  - H1 ANDs its subnet mask with address of H2
    - 255.255.255.128 w/ 128.96.34.139
    - Result : 128.96.34.128
    - Does not match with 128.96.34.0
    - H1 now knows H2 is on a different subnet
    - H1 sends it to default router R1



# Forwarding Algorithm



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

- Use a default router if nothing matches
- Not necessary for all 1s in subnet mask to be contiguous
- Can put multiple subnets on one physical network
- Subnets not visible from the rest of the Internet

# Subnetting

- Assign close subnet numbers to closer networks
  - Addresses the issue of reducing routing table sizes
  - Does not deal with inefficiencies

# Supernetting

- Assign block of contiguous network numbers to nearby networks
  - Aggregation
- Called CIDR: Classless Inter-Domain Routing
  - Pronounced “cider”
- Example :
  - Suppose we need to assign 192.4.16 to 192.4.31
  - Observe:Top 20 bits are the same (11000000 00000100 0001)
  - Effectively a 20-bit number
  - Something between a Class B and Class C network

# CIDR

- Needs classless addressing
  - Routers must understand that networks can be of any length

- Represent blocks with a single pair

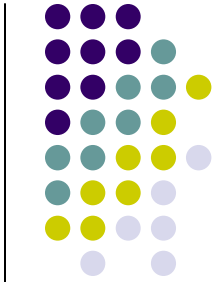
  **`(first_network_address, count)`**

- Restrict block sizes to powers of 2
- Use a bit mask (CIDR mask) to identify block size
- All routers must understand CIDR addressing

# IP Forwarding Revisited

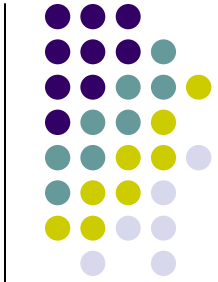
- We so far assumed that we could find the network number in a packet
  - CIDR packets can be of any length (2 to 32)
- Possible to have prefixes that “overlap”
- Example :
  - 171.69 and 171.69.10 can be both in the forwarding table of a router
  - What if you want to route to 171.69.10.5?
    - Both match
- Do the *longest prefix match :* 171.69.10
- 171.69.20.5 would match 171.69
- Enormous potential for hardware design

# Route Propagation



- Know a smarter router
  - hosts know local router
  - local routers know site routers
  - site routers know core router
  - core routers know everything
- Autonomous System (AS)
  - corresponds to an administrative domain
  - examples: University, company, backbone network
  - assign each AS a 16-bit number
- Two-level route propagation hierarchy
  - interior gateway protocol (each AS selects its own)
  - exterior gateway protocol (Internet-wide standard)

# Popular Interior Gateway Protocols

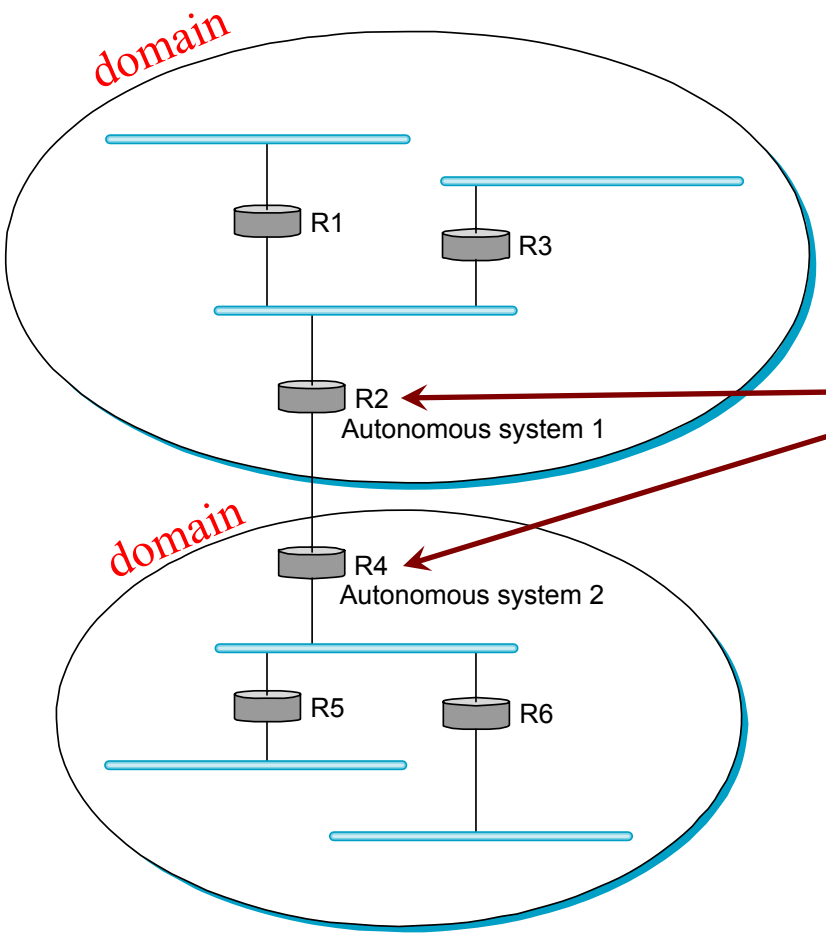


- RIP: Route Information Protocol
  - developed for XNS
  - distributed with Unix
  - distance-vector algorithm
  - based on hop-count
- OSPF: Open Shortest Path First
  - recent Internet standard
  - uses link-state algorithm
  - supports load balancing
  - supports authentication

# EGP: Exterior Gateway Protocol

- Overview
  - designed for tree-structured Internet
  - concerned with *reachability*, not optimal routes
- Protocol messages
  - neighbor acquisition: one router requests that another be its peer; peers exchange reachability information
  - neighbor reachability: one router periodically tests if the another is still reachable; exchange HELLO/ACK messages; uses a k-out-of-n rule
  - routing updates: peers periodically exchange their routing tables (distance-vector)

# Autonomous Systems



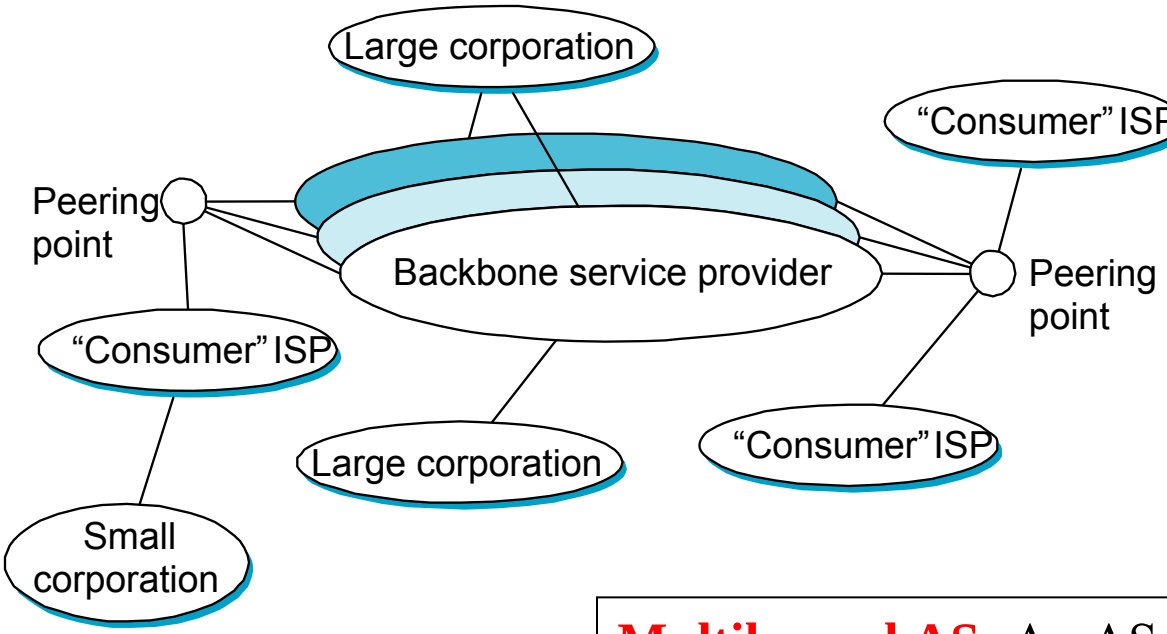

**Goal:** To hierarchically aggregate routing information in a large internetwork to improve scalability.

***Border routers:*** this is the default destination of all packets bound to hosts outside the AS.

The AS model decouples the intradomain routing that happens within one AS from what is done in another AS.

***Prefix:*** as in classless routing, this is expressed as IP/length, i.e. **192.4.16/20**.

# The BGP Model of the Internet

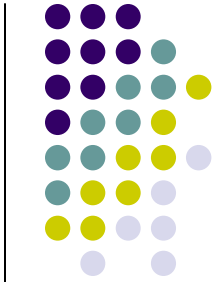



**Stub AS:** Only one connection to another AS.

**Multihomed AS:** An AS with connections to multiple ASs, which refuses to carry external traffic.

**Transit AS:** An AS with connections to multiple AS and carries internal and external traffic.

# BGP Routing

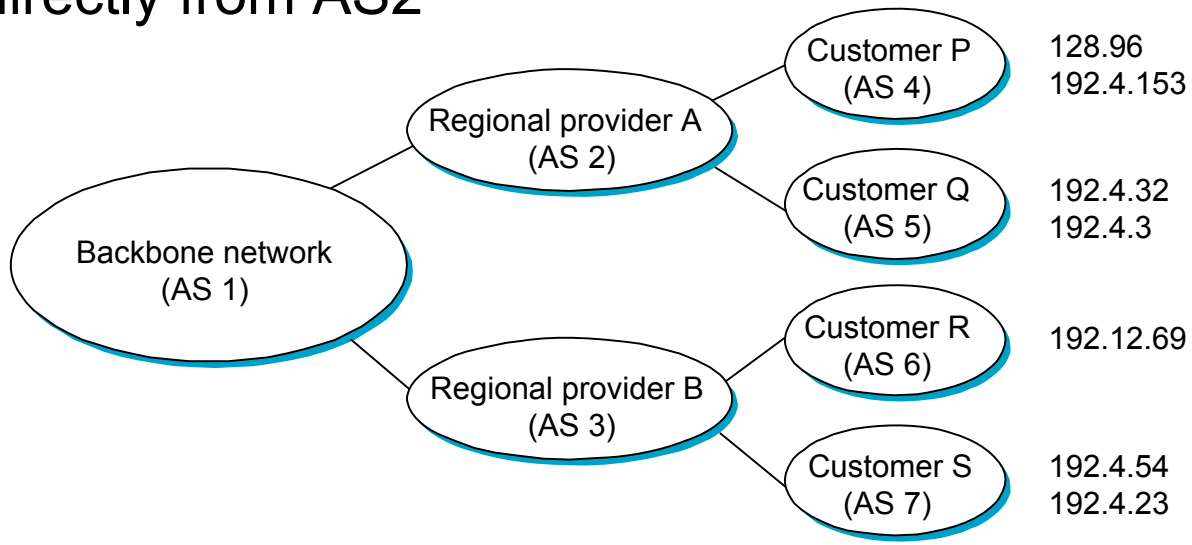


**Challenges:**

- Scale (over 140,000 prefixes),
- Domains are autonomous and can assign arbitrary metrics to its internal paths,
- ASs can only cooperate if they can trust one another to publicize accurate routing information and to carry out their promises,
- Policies should be flexible to allow ASs freedom of action. This choice may override optimal paths and determine the use of paths that are “good enough”.

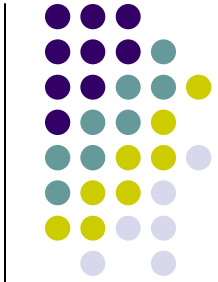
# BGP Example

- Speaker for AS2 advertises reachability to P and Q
  - network 128.96, 192.4.153, 192.4.32, and 192.4.3, can be reached directly from AS2




- Speaker for backbone advertises
  - networks 128.96, 192.4.153, 192.4.32, and 192.4.3 can be reached along the path (AS1, AS2).
- Speaker can cancel previously advertised paths

# Integrating Interdomain and Intradomain Routing

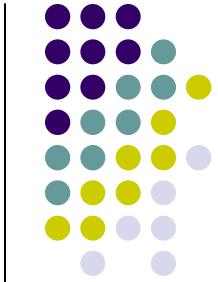


**Stub AS:** The border router ***injects*** a default route into the intradomain routing protocol.

**Multihomed and Transit ASs:** The border routers inject routes that they have learned from outside the AS.

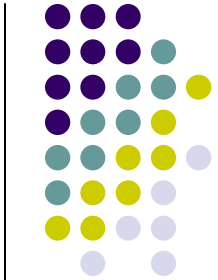
**Transit ASs:** The information learned from BGP may be “too much” to inject into the intradomain protocol: if a large number of prefixes is inserted, large link-state packets will be circulated and path calculations will get very complex.

# IP Version 6



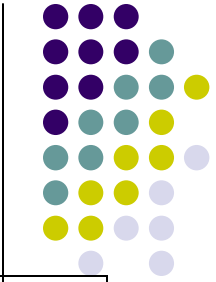
- Features
  - 128-bit addresses (classless)
  - multicast
  - real-time service
  - authentication and security
  - autoconfiguration
  - end-to-end fragmentation
  - protocol extensions
- Header
  - 40-byte “base” header
  - extension headers (fixed order, mostly fixed length)
    - fragmentation
    - source routing
    - authentication and security
    - other options

# Address Space Allocation



| Prefix | Use | |
|---|---|---|
| 0000 0000 | Reserved | |
| 0000 0001 | Unassigned | |
| 0000 001 | Reserved for NSAP allocation | |
| 0000 010 | Reserved for IPX allocation | |
| 0000 011, 0000 1, 0001 | Unassigned | |
| 001 | Aggregatable Global Unicast | **Classes A, B, and C** |
| 010, 011, 100, 101, 110, 1110, 1111 0, 1111 10, 1111 110, | Unassigned | |
| 1111 1110 10 | Link local use | May not be unique |
| 1111 1110 11 | Site local use | May not be unique |
| 1111 1111 | Multicast | **Class D** |

# Address Notation



**X:X:X:X:X:X:X:X where X is a 16-bit value**

- When there are many consecutive 0s, omit them:

  47CD:0000:0000:0000:0000:0000:A456:0124 becomes

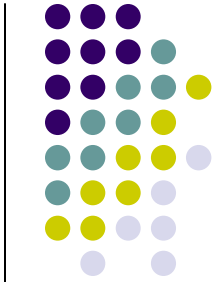
  47CD::A456:0124 (double colon means a group of 0s)

- Two types of IPv6 address can contain embedded IPv4 addresses. For example, an IPv4 host address 128.96.33.81 becomes

  ::FFFF:128.96.33.81 (the last 32 bits are an IPv4 address)

This notation facilitates the extraction of an IPv4 address from an IPv6 address.

# Aggregatable Global Unicast Addresses



**001 prefix:** How are these addresses assigned to ISPs, autonomous systems, networks, hosts, and routers?

**Subscriber:** non-transit AS (stub and multihomed)

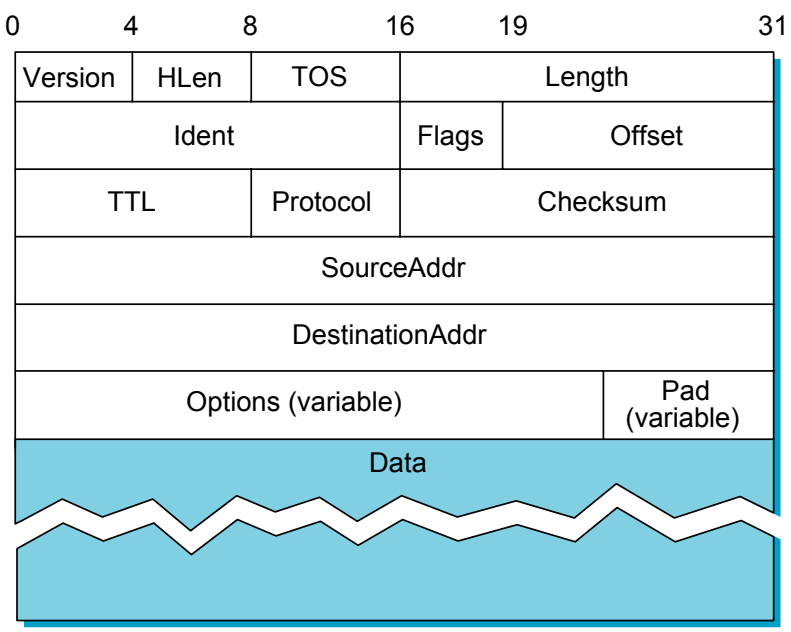
**Provider:** transit AS

**Direct:** connected directly to subscribers

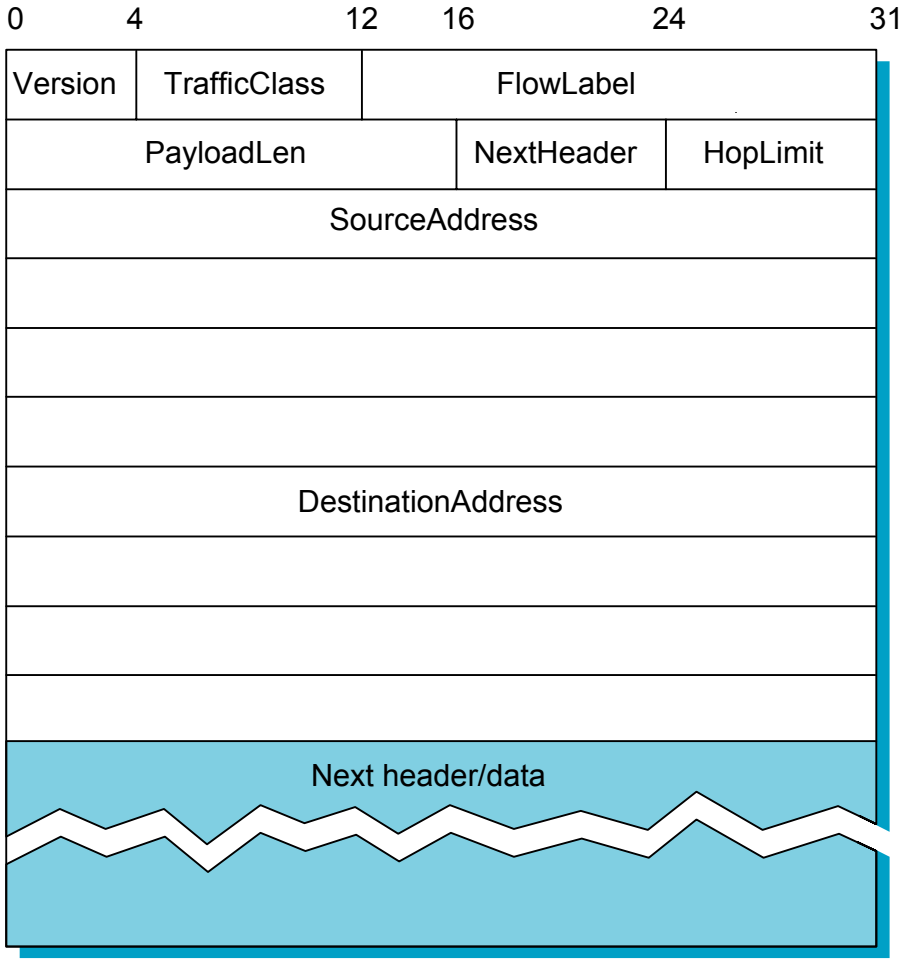
**Indirect** or **Backbone network**: connected to other providers

**Plan:** Aggregate routing information to reduce the burden on intradomain routers. Assign a prefix to a direct provider; within the provider assign longer prefixes that reach its subscribers.

# Packet Format




IPv4




IPv6