CS4700/CS5700
Fundamentals of Computer Networks

Lecture 10: Internet Protocol (IP)

Slides used with permissions from Edward W. Knightly,
T. S. Eugene Ng, Ion Stoica, Hui Zhang
Recap

- Cannot build a global network such as the Internet using Ethernet bridges
- Problem 1: Addressing
- Problem 2: Routing

- Additionally, a global network should allow heterogeneous technologies (e.g. ATM, circuit-switched networks, Ethernet, etc)
New Word: Internetwork

- Multiple incompatible LANs can be physically connected by specialized computers called routers.
- The connected networks are called an internetwork.
  - The “Internet” is one (very big & successful) example of an internetwork

LAN 1 and LAN 2 might be completely different, totally incompatible LANs (e.g., Ethernet, Wi-Fi, ATM, Circuit-switched)
Logical Structure of Internet

- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router connects one network to another
  - Different packets may take different routes
Logical Structure of Internet

- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router connects one network to another
  - Different packets may take different routes
Logical Structure of Internet

- Ad hoc interconnection of networks
  - No particular topology
  - Vastly different router & link capacities
- Send packets from source to destination by hopping through networks
  - Router connects one network to another
  - Different packets may take different routes
Adding an Internetwork Layer (IP) for Interoperability
Issues in Designing an Internetwork

• How do I designate a distant host?
  – Addressing / naming

• How do I send information to a distant host?
  – Underlying service model
    • What gets sent?
    • How fast will it go?
    • What happens if it doesn’t get there?
  – Routing

• Challenges
  – Heterogeneity
    • Assembly from variety of different networks
  – Scalability
    • Ensure ability to grow to worldwide scale
Possible Addressing Schemes

• **Flat**
  – e.g., every host identified by its 48-bit MAC address
  – Router would need entry for every host in the world
    • Too big (although technology can help this)
    • Too hard to maintain as hosts come & go

• **Hierarchy**
  – Address broken into segments of increasing specificity
    • 713 (Houston) – 348 (Rice area) – 2000 (Particular phone)
  – Route to general region and then work toward specific destination
  – As people and organizations shift, only update affected routing tables
An Example of a Binary Hierarchy

Destination address: 101
An Example of a Binary Hierarchy

Destination address: 1 0 1
An Example of a Binary Hierarchy

Destination address: 1 0 1

Datagram
An Example of a Binary Hierarchy

Destination address: 101

Datagram
IP Addressing

- IPv4: 32-bit addresses
  - Typically, write in dotted decimal format
    - E.g., 128.42.198.135
    - Each number is decimal representation of byte
  - Big-Endian Order

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Hexadecimal</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>128</td>
<td>80</td>
<td>0100 0000</td>
</tr>
<tr>
<td>42</td>
<td>2a</td>
<td>0010 1010</td>
</tr>
<tr>
<td>198</td>
<td>c6</td>
<td>1100 0110</td>
</tr>
<tr>
<td>135</td>
<td>87</td>
<td>1000 0111</td>
</tr>
</tbody>
</table>
IP Addressing and Forwarding

• Routing Table Requirement
  – For every possible destination IP address, give next hop
  – Nearly $2^{32} \times (4.3 \times 10^9)$ possibilities!

• Hierarchical Addressing Scheme

  x

  y

  pfx  network  host

  – Address split into network ID and host ID
## IP Address Classes

- **Class A**
  - First octet: 1–126
  - Example: mit.edu: 18.7.22.69

- **Class B**
  - First octet: 128–191
  - Example: rice.edu: 128.42.129.23

- **Class C**
  - First octet: 192–223
  - Example: adsl-216-63-78-18.dsl.hstntx.swbell.net: 216.63.78.18

- **Classes D, E, F**
  - Not commonly used
Two Level Hierarchy of Basic IP addressing

IP address Q

Demultiplex with network id

Network A

Demultiplex with host id

Host B

Size of subtree determined by Class of network id

x

y

pfx

network

host
### IP Address Classes

<table>
<thead>
<tr>
<th>Class</th>
<th>Count</th>
<th>Hosts</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>$2^7-2 = 126$ (0 &amp; 127 reserved)</td>
<td>$2^{24}-2 = 16,777,214$ (all 0s, all 1s reserved)</td>
</tr>
<tr>
<td>B</td>
<td>$2^{14} = 16,398$</td>
<td>$2^{16}-2 = 65,534$ (all 0s, all 1s reserved)</td>
</tr>
<tr>
<td>C</td>
<td>$2^{21} = 2,097,512$</td>
<td>$2^8-2 = 254$ (all 0s, all 1s reserved)</td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>$2,114,036$</td>
</tr>
</tbody>
</table>

- **Partitioning too Coarse**
  - No local organization needs 16.7 million hosts
    - Large organization likely to be geographically distributed
  - Many organizations must make do with multiple class C’s
- **Too many different Network IDs**
  - Routing tables must still have 2.1 million entries
Within Organization: Subnetting

• Add Another Layer to Hierarchy
  
  x y
  z

  pfx | network | subnet | host
  
  – From the outside, appears as one monolithic network
    • Single entry in routing table
  – Within network, manage as multiple subnetworks
    • Internal routers must route according to subnet ID

• Subnet Mask
  
  1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

  pfx | network | subnet | host
  
  – Similar masks used in many contexts
Subnetting

- IP address Q
- Demultiplex with network id
- Network A
- Demultiplex with subnet id
- Subnet S
- Demultiplex with host id
- Host B

Size of subtree determined by Length of subnet mask
## Routing Table

<table>
<thead>
<tr>
<th>Address Pattern</th>
<th>Subnet Mask</th>
<th>Next Hop</th>
</tr>
</thead>
<tbody>
<tr>
<td>128.42.222.0</td>
<td>255.255.255.0</td>
<td>R1</td>
</tr>
<tr>
<td>128.42.128.0</td>
<td>255.255.128.0</td>
<td>R2</td>
</tr>
<tr>
<td>18.0.0.0</td>
<td>255.0.0.0</td>
<td>R3</td>
</tr>
<tr>
<td>0.0.0.0</td>
<td>0.0.0.0</td>
<td>R4</td>
</tr>
<tr>
<td>128.42.0.0</td>
<td>255.255.0.0</td>
<td>R5</td>
</tr>
</tbody>
</table>

- Address 128.42.222.198 matches 4 entries
- Longest Prefix Match
  - Select entry with longest sequence of 1’s in mask
  - Most specific case
Improving the Hierarchy

• Basic Idea of Hierarchy is Good
  – Organizations of different sizes can be assigned different numbers of IP addresses

• Shortcomings of Class-Based Addressing
  – Class A too coarse; Class C too fine; not enough Class B’s
  – When fully deployed would have too many entries in routing table (2.1 million)

• Solution
  – Hierarchy with finer gradation of network/host ID split
Subnetting

- IP address Q
- Network A
- Subnet S
- Host B

Demultiplex with network id
Demultiplex with subnet id
Demultiplex with host id

Size of subtree determined by Length of subnet mask

2.1 million possibilities!
Classless Interdomain Routing

- CIDR, pronounced “cider”
- Arbitrary Split Between Network & Host IDs
  - Specify either by mask or prefix length

```
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

- E.g., Rice can be specified as
  - 128.42.0.0 with netmask 255.255.0.0
  - 128.42.0.0/16
Aggregation with CIDR

- Original Use: Aggregate Class C Addresses
- One organization assigned contiguous range of class C’s
  - e.g., Microsoft given all addresses 207.46.192.X -- 207.46.255.X
  - Specify as CIDR address 207.46.192.0/18

  0 8 16 24 31
  
  207 46 192 0
  
  cf 2e c0 00

  1100 1111 0010 1110 11xx xxxx xxxx xxxx

  Upper 18 bits frozen  Lower 14 bits arbitrary

  • Represents $2^6 = 64$ class C networks
Routing Table Entry Examples

- Snapshot From MAE-West Routing Table
  - Probably out of date

<table>
<thead>
<tr>
<th>Address</th>
<th>Prefix Length</th>
<th>Third Byte</th>
<th>Byte Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>207.46.0.0</td>
<td>19</td>
<td>000xxxxxx2</td>
<td>0 – 31</td>
</tr>
<tr>
<td>207.46.32.0</td>
<td>19</td>
<td>001xxxxxx2</td>
<td>32 – 63</td>
</tr>
<tr>
<td>207.46.64.0</td>
<td>19</td>
<td>010xxxxxx2</td>
<td>64 – 95</td>
</tr>
<tr>
<td>207.46.128.0</td>
<td>18</td>
<td>10xxxxxx2</td>
<td>128 – 191</td>
</tr>
<tr>
<td>207.46.192.0</td>
<td>18</td>
<td>11xxxxxx2</td>
<td>192 – 255</td>
</tr>
</tbody>
</table>

microsoft.com: 207.46.245.214 & 207.46.245.222
- Note hole in table: Nothing covers bytes 96 – 127
Splitting with CIDR

- Expose subnetting structure to external routers
- Example
  - Class A address 12.X.X.X has 413 entries in routing table
  - Prefix lengths 8--24
  - attbi.com
    - Backbone services of AT&T
  - Geographically distributed
    - Don’t want all packets to concentrate to single region
Size of Complete Routing Table

- Source: www.cidr-report.org
- Shows that CIDR has kept # table entries in check
  - Currently require 124,894 entries for a complete table
  - Only required by backbone routers
Important Concepts

- Hierarchical addressing critical for scalable system
  - Don’t require everyone to know everyone else
  - Reduces amount of updating when something changes
- Non-uniform hierarchy useful for heterogeneous networks
  - Class-based addressing too coarse
  - CIDR helps
  - Move to IPv6 due to limited number of 32-bit addresses
- Implementation Challenge
  - Longest prefix matching much more difficult than when no ambiguity
IP Service Model

• Datagram
  – Each packet self-contained
    • All information needed to get to destination
    • No advance setup or connection maintenance
  – Analogous to letter or telegram

IPv4 Packet Format

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>19</th>
<th>24</th>
<th>28</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>HLen</td>
<td>TOS</td>
<td>Length</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Ident</td>
<td>Flag</td>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Header

Source Address

Destination Address

Options (if any)

Data
IP Header Fields: Word 1

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>19</th>
<th>24</th>
<th>28</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>HLen</td>
<td>TOS</td>
<td>Length</td>
<td>Identifier</td>
<td>Flags</td>
<td>Offset</td>
<td>TTL</td>
<td>Protocol</td>
</tr>
<tr>
<td>Source Address</td>
<td>Destination Address</td>
<td>Options (if any)</td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **Version**: IP Version
  - 4 for IPv4
- **HLen**: Header Length
  - 32-bit words (typically 5)
- **TOS**: Type of Service
  - Priority information
- **Length**: Packet Length
  - Bytes (including header)

- Header format can change with versions
  - First byte identifies version
- Length field limits packets to 65,535 bytes
  - In practice, break into much smaller packets for network performance considerations
### IP Header Fields: Word 3

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>19</th>
<th>24</th>
<th>28</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>HLen</td>
<td>TOS</td>
<td>Length</td>
<td>Identifier</td>
<td>Flags</td>
<td>Offset</td>
<td>TTL</td>
<td>Protocol</td>
</tr>
<tr>
<td>Source Address</td>
<td>Destination Address</td>
<td>Options (if any)</td>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- **TTL**: time to live
  - Decrement by one at each intermediate router
  - Prevent looping forever
- **Protocol**
  - Protocol of next layer (in “data”)
  - E.g. TCP (6), UDP (17)
- **Checksum**
  - Of IP header

- Protocol field used for demultiplexing
- Checksum re-computed at each router
  - Why?
- TTL field used to implement traceroute
IP Header Fields: Words 4&5

- Source Address
  - 32-bit IP address of sender
- Destination Address
  - 32-bit IP address of destination

- Like the addresses on an envelope
- In principle, globally unique identification of sender & receiver
  - In practice, there are contexts where either source or destination are not the ultimate addressees
IP Fragmentation

MTU = 4000

MTU = 2000

MTU = 1500

host

router

host
IP Fragmentation

MTU = 4000

MTU = 2000

MTU = 1500
IP Fragmentation

- Every Network has Own Maximum Transmission Unit (MTU)
  - Largest IP datagram it can carry within its own packet frame
    - E.g., Ethernet is 1500 bytes
  - Don’t know MTUs of all intermediate networks in advance
IP Fragmentation

- Every Network has Own Maximum Transmission Unit (MTU)
  - Largest IP datagram it can carry within its own packet frame
    - E.g., Ethernet is 1500 bytes
    - Don’t know MTUs of all intermediate networks in advance
- IP Solution
  - When hit network with small MTU, fragment packets
    - Might get further fragmentation as proceed farther
  - Reassemble at the destination
    - If any fragment disappears, delete entire packet
### IP Header Fields: Word 2

<table>
<thead>
<tr>
<th>0</th>
<th>4</th>
<th>8</th>
<th>12</th>
<th>16</th>
<th>19</th>
<th>24</th>
<th>28</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>version</td>
<td>HLen</td>
<td>TOS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Identifier</td>
<td>Flag</td>
<td>Offset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>Protocol</td>
<td>Checksum</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Source Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Destination Address</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Options (if any)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Each fragment carries copy of IP header
  - All information required for delivery to destination
- All fragments comprising original datagram have same identifier
- Offsets indicate positions within datagram

- **Identifier**
  - Unique identifier for original datagram
    - Typically, source increments counter every time sends packet
- **Flags (3 bits)**
  - M flag: This is not the last fragment
- **Offset**
  - Byte position of first byte in fragment \( \div 8 \)
  - Byte position must be multiple of 8
IP Fragmentation Example #1

MTU = 4000

Length = 3820, M=0

IP Header

IP Data
IP Fragmentation Example #2

MTU = 2000

Length = 3820, M=0

3800 bytes

Offset must be a multiple of 8, but ignored in these examples for simplicity
IP Fragmentation Example #2

Length = 3820, M=0

Length = 2000, M=1, Offset = 0

IP Header

IP Data

3800 bytes

IP Header

IP Data

1980 bytes

Offset must be a multiple of 8, but ignored in these examples for simplicity
IP Fragmentation Example #2

MTU = 2000

Length = 3820, M=0

Length = 2000, M=1, Offset = 0

Length = 1840, M=0, Offset = 1980

Offset must be a multiple of 8, but ignored in these examples for simplicity
IP Fragmentation Example #3

Length = 2000, M=1, Offset = 0

- IP Header
- IP Data

1980 bytes

Length = 1840, M=0, Offset = 1980

- IP Header
- IP Data

1820 bytes

MTU = 1500
IP Fragmentation Example #3

Length = 2000, M=1, Offset = 0

1980 bytes

Length = 1840, M=0, Offset = 1980

1820 bytes

MTU = 1500
IP Fragmentation Example #3

Length = 2000, M=1, Offset = 0

Length = 1500, M=1, Offset = 0

Length = 1840, M=0, Offset = 1980

Length = 520, M=1, Offset = 1480

MTU = 1500

1980 bytes

1480 bytes

500 bytes

1820 bytes
IP Fragmentation Example #3

MTU = 1500

Length = 2000, M=1, Offset = 0

Length = 1840, M=0, Offset = 1980

Length = 1500, M=1, Offset = 1980

Length = 1500, M=1, Offset = 1980

1820 bytes

1980 bytes

1480 bytes

1480 bytes

500 bytes

1820 bytes

1980 bytes

1480 bytes

1480 bytes

1480 bytes

1480 bytes

1480 bytes
IP Fragmentation Example #3

Length = 1500, M=1, Offset = 0

Length = 2000, M=1, Offset = 0

IP Header
IP Data

Length = 1840, M=0, Offset = 1980

IP Header
IP Data

Length = 1500, M=1, Offset = 1980

IP Header
IP Data

Length = 520, M=1, Offset = 1480

IP Header
IP Data

Length = 360, M=0, Offset = 3460

IP Header
IP Data

MTU = 1500

1480 bytes

1980 bytes

1980 bytes

1820 bytes

1480 bytes

340 bytes

500 bytes
IP Reassembly

Length = 1500, M=1, Offset = 0

Length = 520, M=1, Offset = 1480

Length = 1500, M=1, Offset = 1980

Length = 360, M=0, Offset = 3460
IP Reassembly

Length = 1500, M=1, Offset = 0

Length = 520, M=1, Offset = 1480

Length = 1500, M=1, Offset = 1980

Length = 360, M=0, Offset = 3460
IP Reassembly

- Performed at final destination
- Fragment with M=0 determines overall length
  - \((360-20) + 3460\)
IP Reassembly

- Performed at final destination
- Fragment with M=0 determines overall length
  - \((360-20)+3460\)

**Challenges**
- Fragments might arrive out-of-order
  - Don’t know how much memory required until receive final fragment
- Some fragments may be duplicated
  - Keep only one copy
- Some fragments may never arrive
  - After a while, give up entire process
- Significant memory management issues
Frag. & Reassembly Concepts

- Demonstrates Many Internet Concepts

- Decentralized
  - Every network can choose MTU

- Connectionless Datagram Protocol
  - Each (fragment of) packet contains full routing information
  - Fragments can proceed independently and along different routes

- Fail by Dropping Packet
  - Destination can give up on reassembly
  - No need to signal sender that failure occurred

- Keep Most Work at Endpoints
  - Reassembly
Frag. & Reassembly Reality

• Reassembly Fairly Expensive
  – Copying, memory allocation
  – Want to avoid

• MTU Discovery Protocol
  – Protocol to determine MTU along route
    • Send packets with “don’t fragment” flag set
    • Keep decreasing message lengths until packets get through
    • May get a “can’t fragment error” message from router which contains
      the correct MTU
  – Assumes every packet will follow same route
    • Routes tend to change slowly over time

• Common Theme in System Design
  – Fragmentation is handled as a special case by slower general
    processor in router
  – Assure correctness by implementing complete protocol
  – Optimize common cases to avoid full complexity