The Internet Protocol (IP)

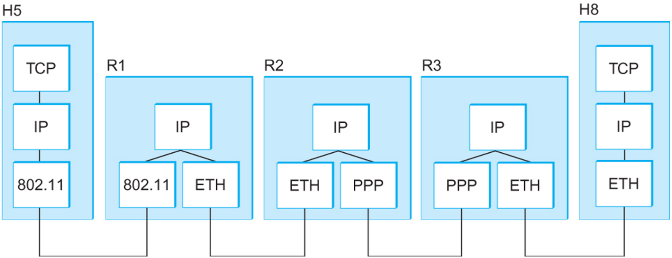
Module Goals

At the conclusion of this module, students will be able to

- describe the IP addressing scheme
- describe how IP addresses are mapped to hardware addresses
- explain how IP addresses are dynamically assigned

The Internet Protocol

- the key network protocol is simply the Internet Protocol (IP)
- key tool used today to build scalable, heterogeneous networks
- it runs on all nodes in a collection of networks and defines the infrastructure that allows these nodes and networks to function as a single, logical internetwork



IP Service Model

- a "low expectations", least common denominator kind of service

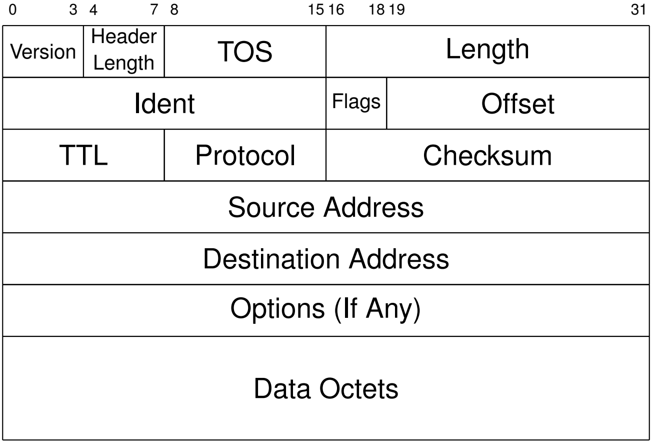
  (runs over anything)

- packet delivery model:
  - connectionless model for data delivery
  - best-effort delivery (unreliable service)
    - packets are lost
    - packets are delivered out of order
    - duplicate copies of a packet are delivered
    - packets can be delayed for a long time

- global addressing scheme

  (we'll talk about this later)

Packet Format



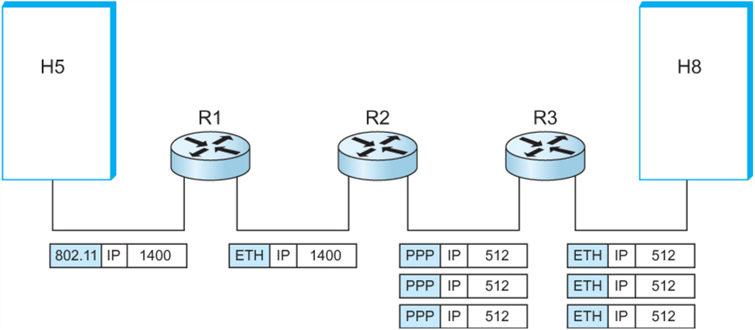
Packet Format

- Version (4 bits): currently 4
- Header Length (4 bits): length of the header in 32-bit words
- TOS (8 bits): type of service, now the differentiated services field
- Length (16 bits): number of bytes in this datagram
- Ident/Flags/Offset (32 bits): used by fragmentation
- TTL (8 bits): number of hops this datagram has traveled
- Protocol (8 bits): demux key; TCP is 6, UDP is 17
- Checksum (16 bits): header checksum only
- Destination Address (32 bits): destination address
- Source Address (32 bits): source address

Fragmentation and Reassembly

- each link layer has some maximum transmission unit (MTU)
  (Ethernet: 1500 bytes; FDDI: 4500 bytes)
- if IP is going to run on any of these link layers (and has to be contained in the payload of link layer frames, we have to figure out how to size packets
  - option 1: make them sufficiently small so that we hope they fit in anything
  - option 2: allow packets to get **fragmented** between multiple frames and reassembled at the destination
    - all fragments carry the same identifier in the Ident field
    - fragments are self-contained datagrams
    - if one fragment gets lost, we discard all fragments
    - we do not try to recover anything from missing fragments

Example Fragmentation



Host H5 wants to send a datagram to Host H8 through the network.

IP Fragmentation and Reassembly

- the original datagram is in (a)
- when it reaches router R2, it gets fragemented into 3 datagrams (b)
  - Ident is set to some value x
  - Flags “more bit” is set
  - Offset is set for the number as the starting point in terms of 8 byte chunks (fragmentation in IP is always on 8 byte boundaries)

(a)

Start of Header				
x			0	0
Rest of Header				
1400 Data Bytes				

(b)

Start of Header				
x			1	0
Rest of Header				
512 Data Bytes				

Start of Header				
x			1	64
Rest of Header				
512 Data Bytes				

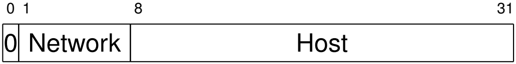
Start of Header				
x			0	128
Rest of Header				
376 Data Bytes				

Global Addresses

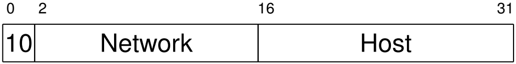
- what makes a good addressing scheme?
- step 1: globally unique addresses
  - TCP had ports, but these weren't unique in the world
  - we'll discuss hardware addresses for protocols like Ethernet which do have globally unique addresses, but...
  - these addresses don't provide any meaningful information on how to actually find these devices
- what if the addresses provided more information?

Hierarchical Addresses

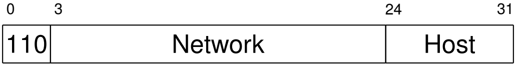
- hierarchical approach: network + host
- $\approx$ 4 billion addresses (32 bits)
  - Class A ($^1/_2$)



  - Class B ($^1/_4$)



  - Class C ($^1/_8$)



- specified using dot notation:
  10.3.2.4, 128.96.33.81, 192.12.69.77
- most addresses are now classless because the original class scheme proved to be too inflexible

A Flawed Idea

- it quickly became clear that the network/host division of IP addresses wasn't sufficient
- a network with 6 hosts would need a class C address, which supports up to 255 hosts (wasting 249! addresses)
- if it was a class B address we would be wasting 64,0000 addresses
- in a nutshell, assigning whole blocks of network addresses to a real, physical network is inefficient

Working Towards a Solution

- one solution: **subnets**
  (but these only fix part of the problem)
  - subnetting lets us take a classful address among multiple smaller networks
  - example: a given organization might not need a class B address (for 64K hosts) and gets 16 class C addresses instead (4096 hosts)
  - now we have a different problem: the 16 different address ranges need 16 different forwarding entries in every router
    (tradeoff between forwarding entries and address waste)
- a better solution: **Classless InterDomain Routing** (CIDR)

Classless Addressing

- CIDR tries to balance the desire to minimize the number of routes that a router needs to know against the need to hand out addresses efficiently
- CIDR uses **aggregate routes**
  - use a single entry in the forwarding table to tell the router how to reach a lot of different networks
  - breaks the rigid boundaries between address classes
  - essentially lets any number of bits in the address represent the “network” portion of the address

Classless Addressing

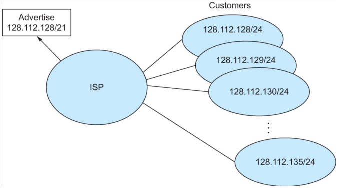
- consider that hypothetical org needing those 16 class C network ranges
- instead of handing out 16 addresses at random, hand out a block of contiguous class C addresses
- suppose we assign the class C network numbers from 192.6.16 through 192.4.31
  - if we blow those values up into binary, we see the top 20 bits of all the addresses in this range are the same: `11000000 00000100 0001`
  - in essence, we just created a 20 bit network number, which is in between a class B network number and class C number
- of course, this means somebody needs to hand out blocks of class C addresses to organizations such that they share a

CIDR Notation

- this calls for a slightly different way of representing the address
- the convention is to place a `/X` after the prefix where `X` is the prefix length in bits
- for example, the 20 bit prefix for all the networks 192.4.16 through 192.4.31 is represented as 192.4.16/20
- by contrast, if we wanted to represent a single class C network number, which is 24 bits long, we would write it as 192.4.16/24
- Michigan Tech owns 141.219/16

Classless Addressing

- this can be done hierarchically too
- the ISP can be allocated the IPs and then hand them out to customers in smaller blocks
- the greater internet sees 128.112.128/21
- For each of the clients, the ISP knows 128.112.xxx/yy
  (yy doesn't need to be the same for all the customers serviced by an ISP)



Multiple Matches

- as we shared, Michigan Tech owns 141.219/16
- however, RIPE Network Coordination Center owns 141/8
- should 141.219.10.20 be routed to RIPE or MTU?
  - principle of **longest specific match**
  - the datagram should be routed to MTU
  - you can look up routing info using `route` on Unix systems

Delivering a Datagram

- so every device has an IP address
- a host on some network sending a datagram to a host on another network can get that datagram to the network
- if it's not on my network, I send it to the default router—the **gateway**
- then it's up to other devices to get it to the gateway for the destination network
- but how does it get to the actual host?
  - an Ethernet switch doesn't know about IP addresses (switches are data link layer devices!)
  - somehow we have to translate an IP address into an address that "makes sense" on the network the datagram is traveling on

Address Translation

- idea 1: encode the physical address (e.g., Ethernet device address) into the IP address
  - question: how do you encode the 48 bit Ethernet address into a 32 bit IP address?
  - answer: you don't; come up with a better ideaa
- idea 2: address translation table
  - a host maintains a table mapping IP addresses to physical addresses
  - question: how does the table get populated?
  - answer: either a human does it, or the hosts dynamically learn the network
  - the dynamic version is the **Address Resolution Protocol** (ARP)

Address Resolution Protocol

- imagine: Host A (10.0.1.4) wants to send a datagram to Host B (10.0.1.8)
- it looks in the table and has no match for that IP
- it sends out a broadcast message on the network: “Who is 10.0.1.8?”
- reply from Host B: “10.0.1.8” is at address `68:A8:6D:54:D8:CA`
  - Host A adds that information to its ARP table
  - Host B also just got a frame from Host A with its IP and physical address, which B adds to its own ARP table
    (after all, it’s pretty likely there’s future communication in store)
- what about the other hosts watching the ARPing?
  (they generally ignore it)

Packet Format

<table>
<tr><td>0</td><td>7 8</td><td>15 16</td><td>31</td></tr>
<tr><td colspan="2">Hardware Type</td><td colspan="2">Protocol Type</td></tr>
<tr><td>H. Length</td><td>P. Length</td><td colspan="2">Operation</td></tr>
<tr><td colspan="4">Source Hardware Address</td></tr>
<tr><td colspan="2">Source Hardware Address</td><td colspan="2">Source Protocol Address</td></tr>
<tr><td colspan="2">Source Protocol Address</td><td colspan="2">Target Hardware Address</td></tr>
<tr><td colspan="4">Target Hardware Address</td></tr>
<tr><td colspan="4">Target Protocol Address</td></tr>
</table>

Packet Format

- Hardwaare Type (16 bits): type of physical network (e.g., Ethernet = 1)
- Protocol Type (16 bits): type of higher layer protocoal (e.g., IP = `0x0800`)
- H. Length: length of hardware addresses, in bytes (e.g., Ethernet = 6)
- P. Length: length of protocol addresses, in bytes (e.g., IPv4 = 4)
- Operation: request (1), reply (2)
- Sender Hardware Address: media address of the sender
- Sender Protocol Address: internetwork address of the sender
- Target Hardware Address: media address of the target
- Target Protocol Address: internetwork address of the target

ARP Security

- does ARP seem secure?
- there are definitely issues
- what if a malicious host responds, “sure, I’ve got IP x.x.x.x”, even when it doesn’t?
- there is also **gratuitous ARP** that is sent by a device first joining the network
- what if a malicious host sends a **gratuitous ARP** with the IP of the gateway?

Getting IP Addresses

- how do devices get IP addresses anyways?
- MAC addresses are given to devices by the manufacturer
  (could IP addresses be given out this way?)
- two main ways this is handled: static and dynamic IP addresses

Dynamic Host Configuration Protocol (DHCP)

- the DHCP server is responsible for providing configuration information to hosts
- there is at least one DHCP server for an administrative domain
- DHCP server maintains a pool of available addresses

DHCP Requests

- newly booted or attached hosts send a `DHCPDISCOVER` message to a special IP address (255.255.255.255)
- routers refuse to forward frames containing this IP address to external networks
- DHCP relay agent unicasts the message to the DHCP server and waits for the response
- why a relay?
- we have a bunch of small networks hooked together via bridges, and who wants a DHCP server for every little network?
- addresses are leased for a certain amount of time, though a host can renew its lease

DHCP Responses

The DHCP Server tells the host:

- what IP address to use
- how long we can use the IP address for
- local DNS servers
- domain name information
- subnet mask
- default gateway

DHCP Security

- does DHCP seem secure?
- we could have an unauthorized DHCP server sending out information ranging from useless to malicious
- with a little work, we could exhaust the IP pool by sending loads of DHCP requests

Summary

- the network layer might have to fragment packets to fit within a link layer transmission
- the Internet Protocol requires globally unique addresses
- addresses can be classful vs. classless
- address discovery can be done with ARP
- address allocation can be done with DHCP
- if everybody can see your traffic, and if you rely on other devices for correctness, you have to show a lot of trust