IT441: Network Servers & Infrastructure

CLASS 4 : 26 Sep 2005
13:30 - 16:15

Last Time:

- servers
  - addresses, of interfaces
  - sockets and ports, sample programming basics
This Time

- IPv4 options
- ICMP
- bridges & routers
- IPv4 addressing: classful, CIDR
- NAT
- IP: a delivery service
- UDP: a transport service

IPv4 Header

```
<table>
<thead>
<tr>
<th>VERS</th>
<th>HLEN</th>
<th>SVC TYP</th>
<th>LEN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IDENT</td>
<td>FLAGS</td>
<td>FRAG OFFSET</td>
<td></td>
</tr>
<tr>
<td>TTL</td>
<td>PROTO</td>
<td>HDR CHKSUM</td>
<td></td>
</tr>
<tr>
<td>SRC ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEST ADDR</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

options [var]: IP address of packet's intended destination

padding: fill to 32-bit size
Options

- may or may not be present in header
- if present, must always be aligned to 32-bit size
  - add padding to end of list as needed
- all start with “options type” byte:

```
0 7
C CL OP TYPE
```

- Copy Flag
  - 0 do not copy
  - 1 do copy

- Class
  - 00 control
  - 01 RFU
  - 10 debug/measurement
  - 11 RFU

- Operation Type [S]
  - 0 end of list
  - 1 NOP
  - 2 security
  - 3 loose source routing
  - 4 timestamp
  - 7 record route
  - 8 stream ID
  - 9 strict source routing

Need for a Protocol

- packets are relayed via sequence of nodes from sender to receiver
- packet may be undeliverable/fail to arrive
- how notify sender of failure?
  - need a protocol for error msgs
Internet Control Message Protocol

- ICMP is protocol used by nodes to send ‘error’ messages to each other (RFC792)
- basic format of ICMP message:

<table>
<thead>
<tr>
<th>TYPE</th>
<th>CODE</th>
<th>CHECKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARMS</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OTHER DATA</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- code values depend on type
- the parameters and other data depend on the type of ICMP message

Internet Control Message Protocol

- for TTL reaching 0,
  - type is 11 ["time exceeded"]
  - codes are:
    - 0 TTL reached 0 during transit
    - 1 TTL reached 0 during re-assembly
- for need–to–but–can’t frag,
  - type is 3 ["destination unreachable"]
  - codes are:
    - 4 need to frag but DF bit set
    - (16 codes are defined for this type)
Errors in Errors

- when should an ICMP message **not** be generated? in response to:
  - an ICMP **error** msg
  - fragment other than first
  - a datagram whose src addr does not identify a single host
  - other situations we'll see later

ICMP

- not all ICMP messages are errors
- some are request/reply msgs, e.g., **echo**
  - sender sends some data in an echo-request
  - dest must return that data to sender in an echo-reply
  - type 8 is request, 0 is reply

<table>
<thead>
<tr>
<th>8 or 0</th>
<th>0</th>
<th>CHECKSUM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>IDENTIFIER</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SEQ NUM</td>
</tr>
<tr>
<td></td>
<td></td>
<td>OPTIONAL DATA</td>
</tr>
</tbody>
</table>
Service Using ICMP Echo Request

PING csl.gmu.edu (129.174.87.240) from 192.120.8.146 : 56(84) bytes
64 bytes from csl.gmu.edu: icmp_seq=0 ttl=235 time=29.361 msec
64 bytes from csl.gmu.edu: icmp_seq=1 ttl=235 time=23.023 msec
64 bytes from csl.gmu.edu: icmp_seq=2 ttl=235 time=21.618 msec
64 bytes from csl.gmu.edu: icmp_seq=3 ttl=235 time=22.383 msec
64 bytes from csl.gmu.edu: icmp_seq=4 ttl=235 time=21.442 msec

--- csl.gmu.edu ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/mdev = 21.442/23.565/29.361/2.935 ms

Service Using ICMP Echo Request

- sender generates request message
  - puts sequence number into msg
  - may put identifier value into ident
    - e.g., UNIX puts PID value there (why?)
- ICMP request msg encapsulated in IP datagram
- sent to dest
- dest causes message to be returned
Tracking Packets

- one of the IPv4 header ‘options’ was “record route"
- causes each router handling a datagram to add its address to the datagram’s header
- how many addresses can we put there?

Tracking Packets

- maximum header length is 60 bytes
  - why?
- 20 of those bytes are already taken
- of 40 left, RR consumes 3 for overhead
  - leaves 37: enough for 9 addresses
- so an ICMP echo request message with RR option set would capture routers the packet visits
  - we’d know how it was routed
**Tracking Packets**

- the traceroute command is intended to show the route a packet followed between sender and dest
- useful those RR option is, it is **not** used by traceroute
  - why not?
  - what is used instead? how does it work?

**Bridges**

- a bridge joins 2 or more segments
  - each is connected through a separate interface
- bridge reads and stores every received frame on each interface and makes this decision:

```java
if (frame for dest on segment different from segment it arrived on) {
    send frame out interface for that (dest) segment
} else {
    discard the frame
}
```
Bridges

- discard frame whose target is on same segment as frame arrived on

- forward a frame out a different interface if dest on different segment
Routers and Bridges

- a bridge:
  1. receives a frame on an interface
  2. discards frame if target is on same interface frame arrived on
  3. forwards frame directly to interface on which next segment is connected/reachable from bridge
     - determined from routing table
  4. if forwarding, only concerned about next segment, not resolving individual hosts
- how similar is router's job?

Routers and Bridges

- a bridge
  1. receives a frame on an interface
- a router
  1. receives some protocol unit (PU) over an interface, e.g., an Ethernet frame
     - depends on kind of network PU arrived on:
       - different PUs have different characteristics:
         - addresses
         - size
         - data format
Routers and Bridges

- a bridge
  2. discards frame if target is on same interface frame arrived on

- a router
  2. delivers to directly connected network

Routers and Bridges

- a bridge
  3. forwards frame directly to interface on which next segment is connected/reachable from bridge
    - determined from routing table

- a router
  3. cannot forward frame directly
    - next network may have totally different requirements
Protocol Units

- to make possible universal service, Internet uses its own form of ‘frame’: the **datagram**
- IP datagram carried by whatever PU physical network uses, e.g., Ethernet frame:

![Diagram](image)

encapsulation

Routers & Packets

- so router must:
  - unpack data frame received
  - pack new frame in target network’s PU format
  - may need to fragment
    - how will it know?
    - forward resulting PU(s)
- do frags get fragmented?
- who does re-assembly?
Routers and Bridges

- a bridge
  4. if forwarding, only concerned about next segment, not resolving individual hosts
- a router:
  4. if forwarding, only concerned about next router

From Here to There

- routers forward packets
- bridges joining LAN segments needed addresses
  - of form <segment ID> <host ID>
- only cared about forwarding to next LAN
  - not interested in individual host
  - unless on directly connected segment
Addressing in Internet

- Internet addresses have two parts:
  - network ID
  - host ID
- routers forward to next network along path toward destination
  - how know this?
  - unless is on directly connected network
- addresses must be unique

addresses must be unique

- network id part must be unique across network
  - requires global coordination
- host id part need only be unique within a network
  - managed locally by administrator of network
**IP Address Format**

- all are 32 bit addresses
  - provides for 4,294,967,296 distinct possibilities
  - earth had estimated population of 6,556,097,802 people
  - $3.4 \times 10^{38}$ for short, or, $= 6 \times 10^{22}$ addresses/ft² earth surface

**IP Addresses, cont’d**

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>B</td>
<td>10</td>
<td>21</td>
<td>8</td>
</tr>
<tr>
<td>C</td>
<td>110</td>
<td>28</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>1110</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- 0.0.0.0 – 127.255.255.255 128 nets, 16M hosts each
- 128.0.0.0 – 191.255.255.255 16K nets, 64K hosts each
- 192.0.0.0 – 223.255.255.255 2M nets, 256 hosts each
- 224.0.0.0 – 239.25.255.255 256M groups
IP: Dots in Addresses and Names

- each decimal digit in dotted decimal notation refers to 8 bits in address
  - *e.g.*, 11000000 01111111 11111010 00000001
    192 . 127 . 253 . 1
- humans can use hostnames which are mapped to IP addresses by the domain name system,
  - *e.g.*, cs.gmu.edu 129.174.87.2
  - site.gmu.edu 129.174.40.83
  - netlab.gmu.edu 129.174.65.1
  - cne.gmu.edu 129.174.120.40
  - osf1.gmu.edu 129.174.1.13

Network Numbers

- number identifying a network in Internet address must be unique
- usually get your network number from an Internet Service Provider (ISP)
- control/allocation of network numbers mediated by Internet Assigned Number Authority (IANA)
  - see http://www.iana.org
- for internet not connected to Internet, addresses and network numbers may be ‘anything’
  - but must be unique within that internet (RFC 1918)
Hierarchical Addresses

- Internet address structure:
  - network number host number
  - i.e., division in 32–bits between network number and host number
- only network number is “imposed”
- net admin could apply hierarchy idea to host numbers within network...
  - introduce a subnet + host number
    - network number subnet host number

Using Subnets

- Diagram showing subnet allocation and examples of subnetted addresses:
  - 129.174/16
  - 255.255.0.0

- Examples of subnetted addresses:
  - cs.gmu.edu 129.174.87.2
  - site.gmu.edu 129.174.40.83
  - netlab.gmu.edu 129.174.65.1
  - cne.gmu.edu 129.174.120.40
  - osfl.gmu.edu 129.174.1.13
Masking

- use **subnet mask** to identify part of address that is network number
- e.g., 129.174.1.13 is:

<table>
<thead>
<tr>
<th>IP addr</th>
<th>Subnet Mask</th>
<th>Network Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000 0001 1010 1110 0000 0001 0000 1101</td>
<td>1111 1111 1111 1111 0000 0000 0000 0000</td>
<td>129 174 0 0</td>
</tr>
</tbody>
</table>

- subnet mask allows fast isolation of network number from address
  - in this example, mask was 255.255.0.0

Notation

- denote mask showing 1’s where network number bits are, 0’s where host number bits are
  - e.g., 255.255.0.0
- denote using `/k` notation: `k` is number of bits in network number, e.g.,
  - GMU class B is 129.174/16
    - 16 bits left for 'internal use' (delivery)
  - NETLAB subnet is 129.174.65/24
    - 8 bits left
One Size Fits All?

- use of subnets adds much flexibility to local sites
  - like universities
- generalize on the idea?
  - split 32-bit address at *arbitrary* point to distinguish network from 'host' numbers
  - need mask to distinguish network number from host number
- if split at arbitrary positions, what happens to address classes?

CIDR

- Classless InterDomain Routing: CIDR
- must have mask specified so as to know how to interpret addr
- advantage?
  - economize on addresses
  - economize on routing table size
CIDR and Address Blocks

- ISP has 10 customers wanting Internet access
  - none has more than 10 computers
  - what class? B or C works best (small # hosts)
- with classful routing, need 10 separate network numbers
- with CIDR, can use, e.g., 4 bits for hosts, so:
  - allocate 128.211.0.16/28 to first
    - provides 128.211.0.16.0 to 128.211.0.16.15
  - allocate 128.211.0.17/28 to second... and so on

CIDR and Address Blocks

- with CIDR, can use, e.g., 4 bits for hosts, so allocate 128.211.0.x/28
  - consecutive values of x for separate customers
- so now have 10 addresses with common 24-bit prefix
  - all traffic to any of those addresses comes to this ISP
- reduce routing table size by keeping only the prefix in tables
  - as 128.211.0.0/24
- whole block reachable via one router
E Pluribus Unum

- basic IP address rule: one IP address = 1 real interface
- so every node on Internet must have unique IP address
- uses up address space quickly
- if could have one IP address used by many different nodes, could save addresses
  - but how?

Network Address Translation

- another way to economize on IPv4 addresses
- a work-around in common use today is network address translation (NAT):
  - use one IPv4 address to 'front' for many hosts
  - assign hosts addresses from 'private' address space
    - e.g., class A net 10 (ARPANET) set aside for testing
  - run a process that uses TCP port multiplexing to identify 'back side' host address
NAT Operation

Port Mapping Table:
-port:real_src:real_port:dest_addr:dest_port

outside world, i.e., the Internet, only sees IP addr of NAT gateway

NAT Operation, cont’d

- port map table entries are allocated on-demand for requests originating ‘inside’ NAT space
- table entries have time-outs
  - reset each time they are used
  - typically 2 - 3 minutes
- packets arriving with dest_port not found in table are dropped (firewall)
  - so how can you run a server here?
Why NAT is ‘evil’…

- involves re-writing of IP headers
  - hence also re-computation of checksums
- some applications are ‘NAT sensitive’
  - use of NAT breaks them, e.g., ftp
  - propose use of ‘application level gateways’ to rewrite parts of messages as well as headers
- threatens:
  - end-to-end security
  - end-to-end functions
- see http://www.ietf.org/html.charters/OLD/nat-charter.html

Doing NAT

- software implementations suitable for speeds up to \( \approx 10 \) Mbps
  - can run on not-so-fancy hardware, e.g., old PC
  - software typically free
    - masquerade for linux
    - Internet Connection Service for Windows
- firmware in consumer products
  - like SOHO routers (NAPT + hub or sw in a box)
- hardware for high performance needs
Special Addresses

- not all address values can be assigned to hosts
  - numbers 0 to $2^k - 1$ for k-bit host numbers
  - not all valid for identifying hosts
- host number 0: reserved as network address
  - not legal to assign to a host
  - should never appear in pkt as dest addr
  - e.g., 128.211.0.0/16 is address of net with prefix 128.211

Special Addresses

- host number $2^k - 1$ (i.e., all 1 bits) reserved as network’s directed broadcast address
  - single pkt arrives at target network where it is propagated to every host in that network
  - can use hardware broadcast if hardware supports it (e.g., Ethernet)
  - do in software by individually replicating pkt otherwise
Special Addresses

- IP address with all 1 bits (255.255.255.255) is **limited broadcast address**
  - broadcasts pkt to everyone on local net only
  - typically used at boot time by computers that don’t know their network number

Special Addresses

- IP network 127/8 is **local loopback**
  - used to test programs that use Internet
  - can be used by apps to contact other apps on same node avoiding ‘special-casing’ local access
  - no packets leave node
  - loopback addr never appears in pkt on network
**Traffic To/From Routers**

- how does a packet get to a router?

**Traffic To/From Routers**

- every router has $\geq 2$ interfaces
Multi-Homed Hosts

- node C on 2 networks: is it a router?

More Than One Destination

- all routing so far has been unicast
  - one sender → one destination
- what if want to send to > 1 destination?
  - broadcast: send to ‘everybody’
  - multicast: send to everyone in a set (group)
**Multicast**

- receivers of a multicast are nodes in a group
  - may join group at any time; start receiving pkts
  - may leave group at any time; stop receiving pkts
- applications on nodes join groups
  - node running app tells nearby router
  - if > 1 app on node joining, router only sends node 1 pkt, and node does replication
- group only defines set of receivers
  - any app on any node can send to any group at any time

**IGMP**

- Internet Group Multicast Protocol [RFC 3376]
- used on network between host and its first router
- lets host declare itself as member of a group
  - host, not an app on the host
IP Multicast

- send packets to a group of addresses in a network
- IPmc: packets sent to “groups” addresses:
  - delivered in parallel
  - class D address 224.0.0.0 and greater

<table>
<thead>
<tr>
<th>D</th>
<th>1110</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>28</td>
</tr>
</tbody>
</table>

224.0.0.0 – 239.25.255.255 256M groups

- LANs have inherent broadcast/multicast capability
  - how come?

IPmc, cont’d

- for WANs:
  - set of all nodes in multicast group forms a tree
  - rooted at node starting the group
  - routers at forks in tree must **replicate** packets
    - commercially available routers do this
  - requires a different routing protocol than unicast
IPmc Tree

A B C D E F G H J

IP Multicast Directionality

- IPmc uses a many:many model of packet delivery
  - equivalent to fdx for multicast
  - any participating host can send to group
  - good for collaborative-style applications
  - very appealing model but increases complexity of network layer considerably
SSM 1:many multicast

IP Multicast Directionality

- IETF recently developed source specific multicast
  - uses 1:many model
  - equivalent to hdx for multicast
  - only 1 host can send to group
  - good for presentation-style applications
IP Multicast Routing

- Internet Group Management Protocol (IGMP)
  - runs over IP
  - distributes info between router and hosts on LAN
  - does not have router-to-router role (as does ICMP)
- IPmc routers use IP ‘tunnels’ to span sections of Internet that do not route IPmc
  - IPmc packets encapsulated in IP unicast

Tunnelling

- tunnels used to cross networks that don’t support X.
- e.g.,
  - car = packet
  - roadways = networks
  - waterway = network that doesn’t support car traffic on surface
- use tunnel (chunnel) to carry cars ‘encapsulated’ (in train)
Types of Service

- **connectionless**:  
  - fast, easy, low overhead  
  - each pkt sent as independent event  
    - may or may not arrive; may arrive more than once...  
  - each pkt may follow different route to dest  
  - no assurances of reliability, correct ordering, ...

- **connection-oriented**:  
  - higher overhead: e.g., setup and tear-down  
  - assures reliable, correctly ordered delivery of pkts  
  - uses *virtual-circuit* connection

Delivery Services

1. best effort  
2. semi-reliable/unordered    
   - sender notified re. receipt of pkts  
3. reliable  
   - delivers correct pkts in correct order to next layer
IP: Internet Protocol

- delivery protocol:
  - packet src → dest
  - over internet or Internet (i.e., possibly crossing multiple networks)
- simple packet format: header + data
- no checksum coverage of payload
- used by protocols we’ve already seen, including:
  - ICMP

IP: Internet Protocol

- best-effort delivery: makes best try to deliver, but doesn’t handle problems of datagrams being:
  - duplicated
  - delayed or arriving out-of-order (sequence)
  - corrupted
  - lost entirely
- solving these problems is left to higher-level protocols
Need More Than IP?

- IP is a **delivery** mechanism
  - identifies source and destination by IP address
- to be useful, must have software on dest listening for pkts to arrive
  - i.e., someone to deliver the pkt to
- how identify which of possibly many listeners pkt should go to?
  - no provision in IP dgram for recipient identification beyond host IP address

Transport Protocol

- need a **transport** protocol:
  - can identify individual end-point (process)
  - both source process and dest process
  - a.k.a. “end-to-end” protocol
- implemented as a separate layer in IP:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPLICATION</td>
<td>5</td>
</tr>
<tr>
<td>TRANSPORT</td>
<td>4</td>
</tr>
<tr>
<td>INTERNET</td>
<td>3</td>
</tr>
<tr>
<td>NETWORK INTERFACE</td>
<td>2</td>
</tr>
<tr>
<td>PHYSICAL</td>
<td>1</td>
</tr>
</tbody>
</table>
UDP: User Datagram Protocol

- simple, cheap, fast
- connectionless
- best-effort
- message-oriented
- no control msgs
- \( n:m \) interaction; \( n,m \geq 1 \)

UDP: User Datagram Protocol

- message oriented:
  - targeted for sending individual msgs, one per pkt
- UDP does no fragmenting nor reassembly:
  - each msg must fit within one UDP dgram
  - penalize processes sending small amounts of data
  - can penalize processes sending large messages
- uses IP for delivery
  - hence max size as determined by IP
**UDP: User Datagram Protocol**

- interaction model allows for IP
  - unicast: one sender → one receiver
  - multicast: one sender → many receivers
  - broadcast: one sender → everyone

---

**Who’s Listening?**

- pkts are intended for a process at dest
  - having originated from a process at src
- different platforms have different ways to identify their processes
  - no one standard OS process identification method
- define ‘our own’ method of identifying listening processes: by **port number**
- port numbers identify a service
  - whose implementation is platform specific
  - but that meets protocol definition of the service
UDP Ports

- port numbers are 16-bit values (0 to 65535)
- some port numbers are assigned:
  - 0 – 1023 are “well-known” port numbers
  - originally listed in RFC 1700
  - now found online at:
    http://www.iana.org/assignments/port-numbers
- see /etc/services on UNIX platforms for list, e.g.,
  - daytime 13/udp
  - http 80/udp

UDP Datagram

- has port numbers for source and dest
  - identifies endpoints of the dgram
- checksum covers UDP header and data
  - and is optional
    - if checksum is wrong, pkt is silently discarded
- what don’t you see in the datagram?
  - why aren’t they there?
### UDP Datagram Pseudo-Header

- **UDP pseudo-header**
- **UDP header**

- pseudo header used only in checksum calculation

### UDP Checksums

- if not being provided—**bad idea** — then report as all 0-bits
  - receiver understands this to mean no checksum was calculated
  - what if actual checksum value was 0?
- bad because:
  - no one else checks integrity of header info
  - some link layer protocols have no checksums of their own
A Better Model…

- applications want
  - to focus on the nature of the app
  - not to deal with handling communications reliability problems
- need a transport protocol able to deliver end-to-end reliable transport:
  - no duplicate packets
  - no out of order packets
  - no missing packets
  - no errors in packets