1 # EECS 122, Lecture 18

Today's Topics:

Review of Where We Are

Introduction to Transport Layer

UDP: The User Datagram Protocol

Introduction to Reliability

Kevin Fall, kfall@cs.berkeley.edu

2 # Where We Are So Far…

- Networking concepts
  - remote access to resources
  - controlled sharing
    - multiplexing: TDM, Stat Mux
  - protocols and layering
    - ISO reference model, encapsulation
    - service model, error detection
    - end-to-end argument
    - soft state

3 # Where We Are So Far…

- Development of the Internet
  - interconnection of heterogeneous networks
  - simple best-effort service model
  - fully-connected graph of hosts (routing)
- Internet scaling issues
  - use of hierarchies in routing, addresses, DNS
  - use of caching in DNS

4 # Where We Are So Far…

- Direct-link networks

– signals, modulation, error detection
– best-effort delivery between attached stations
– possible error correction using codes
– MAC protocols, Ethernet

5 Where We Are So Far...

- The Internet Protocol
  - IP service model
    - best-effort datagram model
    - error detection in header only
    - consistent, abstract packet, addressing
    - routing
    - signaling (ICMP)
    - multicasting, IGMP, multicast routing
    - IP futures with IPv6

6 What We Are Missing...

- Access to process-level information
  - currently, can only send traffic from one computer to another
  - no way to indicate which process or service should receive it
- Reliable transport
  - no way to know whether data received was correct
  - no way to correct for delivery errors

7 Problem Set #3

- Peterson & Davie:
  - Ch 3: 11, 12, 13, 15
  - Ch 6: 2, 8, 10
  - Ch 8: 2, 5, 15, 17
  - (problem on web page)
- Due April 13

8 The Transport Layer

- provide application-to-application communication (end-to-end)
- properties to expect:
  - guaranteed message delivery, correct ordering, duplicate elimination, large messages (streams), end-to-end synchronization, flow control, multiple applications [clients/servers]
- what is lacking: security, format conversion

9 Internet Transport Layers

- Two main ones: UDP and TCP
- UDP (User Datagram Protocol)
  - datagram abstraction
  - error detection
- TCP (Transmission Control Protocol)
  - stream abstraction
  - error detection and correction
  - flow control
  - congestion control

10 Identifying Processes/Services

- How to identify a service/process
  - process ID?
  - process memory address?
  - these are OS specific, and may be transient
- Mailboxes (ports)
  - abstract way of reaching a process/service
  - does not correspond to physical entity
  - usually some fixed number per computer

11 Port Numbers

- How to completely identify a remote application/service on the Internet?
- [IP Address, port number, protocol]
  - expect to find a process listening for incoming requests on IP address, port number, using transport layer protocol
  - doesn't tell which application it is!
  - (or which app-layer protocol to employ)

12 ## Picking Port Numbers

- Port numbers are in range [0..64K-1]
- Ports below 1023 are known as "reserved" or "well-known" ports, and are managed by IANA
- Ports in range 1024-65535 may be "registered" with IANA but aren't enforced by them
- RFC1700 - Assigned Numbers RFC

13 ## Why Does This Matter?

- To what port should a client send in order to reach a server?
- To what port should a server starting off *bind* to?
- For standard services, well-known port provides an answer
- Some well-known ports:
  - echo (7), discard (9), DNS (53), snmp (161)

14 ## Ephemeral Ports

- Typically, servers will bind to a particular port they are assigned (e.g. well-known)
- Clients use a temporary, OS-assigned port (an *ephemeral* port)
- Servers are capable of detecting the client's port

number, enabling responses to be sent to a particular client process

- Ephemeral ports are returned to the OS to give out later after process completes

15 UDP: User Datagram Protocol

- UDP provides a datagram service model
- Provides error detection, not correction
- Basically is IP with an end-to-end checksum and with port numbers
- UDP Header (8 bytes):

- (NOTE: book is WRONG!)

16 UDP Header Structure

- Source Port: sender's port number
- Dest Port: destination's port number
- Length: data plus header length (minimum value is 8)
- Checksum: [optional] 16-bit 1's complement sum of a *pseudoheader* of information from the IP header, UDP header, and data, padded with zero if necessary to be a multiple of 2 bytes

17 The UDP Checksum

- End-to-end checksum
- Pseudoheader is a logical collection of fields over which the checksum is computed; not sent directly as data

18 Pseudoheader

- Why use such a thing?
  - Including IP header info provides an end-to-end check on src/dst IP addresses and IP protocol info
  - assures the correct recipient
  - required in IPv6 (recall no hdr checksum)
- A layer violation
  - transport layer needs to "peek inside" network layer
  - hard to run UDP on other than IP net layer

19 Implications

- modifications to the IP address or protocol info is detected by the transport layer
- systems that intentionally modify IP addresses [e.g. NAT devices] must also modify UDP-layer checksum

20 Sending a UDP Datagram

- Application acquires dest IP address, port number to send (e.g. use DNS)
- Application chooses message size, requests send using API (e.g. sockets)
- API allocates OS-level buffer, leaving room for some headers, copies data from user-level buffer to OS-level buffer, gives to UDP

21 Sending a UDP Datagram

- UDP Module receives user buffer, prepends IP and UDP headers
- fills in IP header info [proto, len, src, dst]
- fills in UDP header [sport, dport, len]
- computes pseudoheader cksum if enabled and fills it in
- sets TTL and TOS (system defined)

- sends UDP/IP packet to IP

22 Sending a UDP Datagram

- IP Module receives packet
- insert options if enabled
- set IP vers, IHL, offset, ID fields
- determine a interface/MTU to use
- if multicast, look for special TTL, info
- fragment if needed and send to link layer

23 Receiving a UDP Datagram

- Network adapter receives frame, interrupts processor
- Device driver determines frame contains IP type data, strips header, gives to IP
- IP checks header, processes options
- IP checks for good address (unicast, one of our multicasts, broadcasts)
- IP reassembles if necessary, gives whole pkt to UDP based on protocol field

24 Receiving a UDP Datagram

- UDP receives IP/UDP packet
- checks length and checksum
- if multicast, give to all listeners on port
- locate OS PCB based on dest port, providing receiving process 'ID; generate ICMP unreachable if nobody there
- copy to receiving process 'buffer
- make receiving process runnable

25 What a UDP/IP Packet Looks Like

- UDP/IP Packet on Ethernet, no frag:

- UDP/IP Packet on Ethernet, frag'd:

26 Why Use UDP?

- Downsides:
  - no error correction
  - no flow control
  - no congestion control
  - app picks packet size
- Upsides:
  - no connection establishment or state
  - broadcast/multicast more straightforward
  - app picks packet size

27 Intro to Reliability

- So, with UDP we basically have IP with port numbers and error detection
- Would like a way to provide reliable delivery to applications
- Must deal with:
  - packet drops, duplicates, and damage
  - flow control (overrun at receiver)
  - congestion control (overrun in network)

28 Repairing Errors

- We have already seen error correcting codes. These are rarely use to repair whole-packet errors (drops)
- Instead, typical strategy is to re-send data which was lost during transit (lost includes damaged beyond repair)
- Example of ARQ (Automatic Repeat Request)

29 Simple ARQ: Stop & Wait

- Agree that a receiver will send an *acknowledgement* (ACK) to the sender for every packet it receives correctly (e.g. validating checksum)
- When sender sends packet, also sets a timer
- If no ACK received before timer expires, sender *retransmits* the packet

30 Stop and Wait Event Plot

31 Stop and Wait Performance

- Stop and Wait doesn t perform very well
- How much work is done?
  - one packet every send/ACK cycle
  - so, about 1 packet every round-trip time (RTT)
  - overall throughput is ~ to (1/RTT)
  - degrades significantly as RTT goes up (distance from sender to receiver grows)
- Next time, will see how to improve this...