Transport Layer

Table of Contents

- Reading
- Transport Layer Services
  - TCP/UDP Service Models
- Multiplexing and Demultiplexing
  - Multiplexing UDP
  - Multiplexing TCP
  - Establishing a TCP connection
  - Port scanning
  - Web servers and TCP
- UDP
- UDP Segment Structure
- Reliable Data Transfer

Reading

- ☒ K&R 3.1
- ☒ K&R 3.2
- ☒ K&R 3.3
- ☐ K&R 3.4
- ☐ K&R 3.5
- ☐ K&R 3.6

Transport Layer Services

- **logical communication** at transport-layer: service provided by transport-layer protocol that makes it appear from application perspective that hosts running the processes are directly connected, even though they may be on opposite sides of the world connected by various links
  - allows apps to send messages without worrying about physical infrastructure carrying these messages
- **segments** are transport-layer packets

- transport-layer protocols are implemented on hosts, not in routers, and handles messages from application process to network edge
  - doesn't have any say about routing in network core
- network layer provides logical communication between *hosts*
  - doesn't respond to information transport layer may have added to messages
- **User Datagram Protocol (UDP)**: unreliable, connectionless service
- **Transmission Control Protocol (TCP)**: reliable, connection-oriented service
- network layer protocol: **Internet Protocol (IP)**
  - best-effort delivery service: no guarantees
  - unreliable service
  - every host has an IP address

TCP/UDP Service Models

- TCP + UDP
  - **transport-layer de-/multi-plexing**: extend host-to-host delivery to process-to-process delivery
  - **integrity checking**
- TCP only
  - **reliable data transfer** (TCP only)
  - **congestion control**

Multiplexing and Demultiplexing

- e.g. downloading Web pages, while running FTP session and telnet sessions: 4 network application processes running
  - transport layer receives data from network layer and needs to direct to one of these four processes
- **sockets**: interfaces between process and network; each with a unique ID
- **demultiplexing**: deliver data from segment to correct socket
- **multiplexing**: combining data from different sockets in a segment and passing to network layer
- **source/destination port number:** listed in header field of segment
  - 16-bit: 0-65535

  - well-known port numbers: 0-1023; restricted/reserved for well known application protocols
  - application must be assigned a port number

Multiplexing UDP

- socket identified by two-tuple: (destination IP address, destination port number)
- segments with distinct source IP address and/or source port, but with the same destination IP address and port, will be directed to the same socket
- source IP address/port are used as a return address

Multiplexing TCP

- socket identified by 4-tuple: (source IP address, source port #, destination IP add., destination port #)
- when TCP segment arrives from network to host, all four values are used to demultiplex to the appropriate socket
- two segments with distinct source IP address and/or source port will be directed to two different sockets
  - exception: segment containing original connection-establishment request
- TCP server has “welcoming socket” on port 12000 that listens for connection-establishment requests from TCP clients
  - connection establishment request segment
    * destination port 12000
    * connection-establishment bit set in TCP header
    * source port number set by client
- server host may support simultaneous TCP connection sockets, each attached to a process, and each socket identified uniquely by its 4-tuple

Establishing a TCP connection

- TCP client creates a socket
  - sends connection-establishment request
- host OS on server receives connection-request segment

  - locates server process waiting to accept a connection,
  - server creates a new socket
  - transport layer keeps track of
    * source port number
    * IP address of source host
    * destination port number in segment
    * server IP address

Port scanning

- `nmap`
  - TCP: sequentially scans ports for those accepting TCP connections
  - UDP: sequentially scans ports for those UDP ports that respond to transmitted UDP segments
  - returns a list of open/closed/unreachable ports
  - can attempt to scan any target host anywhere in the world

Web servers and TCP

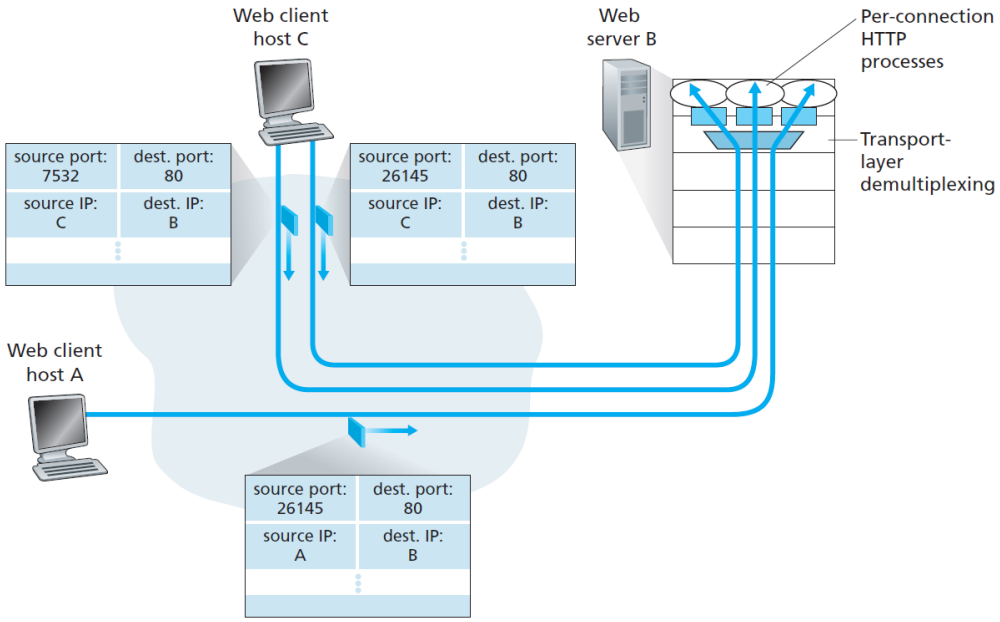

Figure 3.5 ♦ Two clients, using the same destination port number (80) to communicate with the same Web server application

Figure 1: tcp_clients

- consider host running Apache Web server on port 80:
  - when browsers send segments to the server, all segments will have destination port 80 (including connection-establishment segment and segment carrying HTTP request messages)
  - server distinguishes between them using source IP address and source port
  - Web server may spawn a new process for each connection, each with its own socket
  - high performance Web servers typically only use one process, and create new threads (lightweight processes) for each new client connection
  - persistent HTTP: for duration of connection, client and server exchange HTTP messages via the same server socket
  - non-persistent: new socket created/closed for every request/response

UDP

- does about as little as transport protocol can: de-/multiplexing + error checking

- e.g. DNS uses UDP
  - if application at querying host doesn't receive reply, it may resend the query, try sending query to another name server, inform invoking application that it cannot get a reply
- e.g. SNMP (Simple Network Management Protocol) uses UDP
  - must operate when network is in a stressed state, which is when reliable, congestion- controlled is difficult
- e.g. Internet phone, video-conferencing may use UDP
  - applications react poorly to TCP's congestion control
  - tolerate some packet loss
- why to choose UDP for an application
  - finer application-level control over what data is sent and when
    - UDP immediately packages data in segment and passes to network layer
    - TCP has congestion control which throttles sender
    - real-time apps typically require minimum sending rate and can tolerate data loss
  - no overhead associated with connection establishment
    - main reason for DNS to use UDP as it is much faster than would be with TCP
    - Quick UDP Internet connection (QUIC) protocol: used in Chrome; uses UDP as underlying transport protocol and implements reliability as application-layer protocol
  - no connection state
    - allows UDP to support many more active clients c.f. TCP
  - small packet header overhead
    - UDP: 8 bytes overhead per segment
    - TCP: 20 bytes overhead per segment
- use of UDP widely for multimedia applications will cause network congestion,
  - high rate of packet loss
  - slow down rates of TCP connections
  - research area: adaptive congestion control for UDP
- reliability of data transfer is possible, but responsibility of application layer

UDP Segment Structure

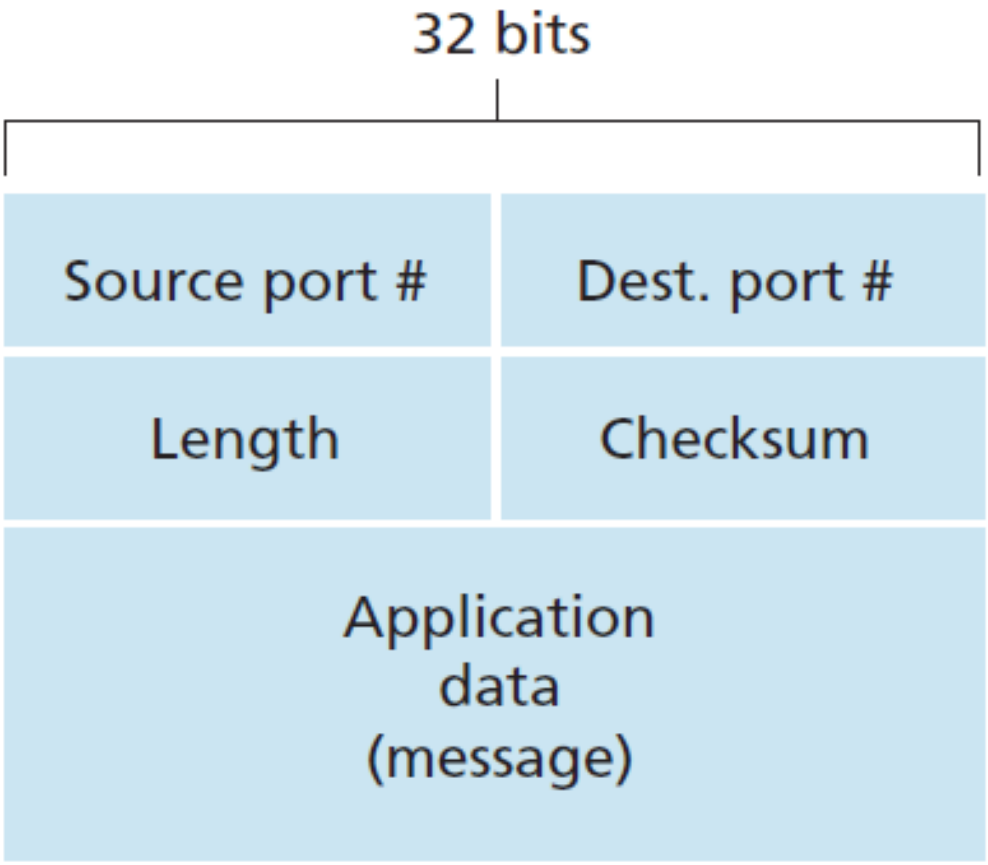

Figure 2: udp_segment

- UDP header: 4 fields, 2 bytes each
  - Souce port #
  - Destination port #
  - Length: number of bytes in segment (header + message)
- Checksum: error-check
  - determine if bits of UDP segment have been altered (noise in links, while stored in router, …) between source and destination
  - see RFC1071

  - sender side: 1s complement of sum of all 16-bit words, with overflow being wrapped
    * 1s complement: flip bits
  - receiver side: sum all 16-bit words, add to checksum
    * if no errors are introduced, should get 0xffff, i.e. if any of the bits is 0, there has been an error introduced
  - UDP provides checksum because no guarantee that link-layer protocol will provide error-checking, and it's possible an error will occur while segment is stored in router's memory

Reliable Data Transfer

- reliable data transfer is fundamentally important problem in networking and applies at transport layer, link layer, application layer
- service abstraction provided to upper layers is a reliable channel through which data can be transferred
  - reliable channel: no data bits corrupted, all data delivered in order sent/