3. Transport Layer

3.1 Transport-layer services

* Transport services and protocols provide logical communication between app processes running on different hosts
* Network layer provide logical communication between hosts
* TCP: reliable, in-order delivery, congestion control, flow control, connection setup
* UDP: unreliable, unordered delivery, no-frills extension of ‘best-effort IP’
* Services not available by transport layer: delay guarantees, bandwidth guarantees
* Extending host-to-host delivery to process-to-process delivery is called transport-layer multiplexing and demultiplexing
* UDP provides only two services: process-to-process data delivery and error checking

3.2 Multiplexing and Demultiplexing

* Host uses IP address and port numbers to direct segment to appropriate socket
* TCP socket identified by 4-tuple: source IP address, source port number, dest IP address, dest port number
* Each port number is a 16-bit number, ranging from 0 to 65535
* The port numbers ranging from 0 to 1024 are called well-known port numbers
* HTTP – port 80, FTP – port 21
* UDP socket identified by 2-tuple: destination IP address, destination port number
* A connection-establishment request is nothing more than a TCP segment with destination port number 12000 and a special connection-establishment bit set in the TCP header

3.3 Connectionless Transport: UDP

* Connectionless: no handshaking between UDP sender, receiver; Each UDP segment handled independently of others
* UDP min length 8 bytes
* Why UDP: no connection establishment (which can add delay); simple: no connection state at sender, receiver; small header size; no congestion control: UDP can blast away as fast as desired
* UDP checksum goal: detect “error” (flipped bits) in transmitted segment
* UDP checksum: addition of segment contents (one’s complement sum)
* Sender puts checksum value into UDP checksum field
* Receiver: computer checksum of received segment
* DNS uses UDP over TCP because UDP has no connection establishment
* UDP > TCP : Finer application-level control over what data is sent, and when; No connection establishment; No connection state; Small packet header overhead
* TCP header 20 bytes
* UDP header 8 bytes: source port, dest port, length, checksum
* It is possible for application to have reliable data transfer when using UDP. This can be done if reliability is built into the application itself for example adding acknowledgment and retransmission mechanisms
* UDP length field specifies the number of bytes in the UDP segment (Header plus data)
* UDP sender: 16-bit segments added and then one’s complement, this is checksum
* UDP receiver: At the receiver, all four 16-bit words are added, including checksum. If no error result is 1\*16
* UDP does not do anything to recover from an error

3.4 Principles of Reliable Data Transfer

* Acknowledgements (ACKs): receiver explicitly tells sender that pkt received OK
* Rdt 2.0 has fatal flaw: ACK, NAK packet could be corrupted. A simple solution is to add a new field to the data packet and have sender number its data packets by putting a sequence number into this field
* Rdt 3.0 – alternating-bit protocol
* Rdt 2.0 – stop and wait
* Rdt too slow because of stop and wait, answer is pipelining
* Two basic approaches toward pipelined error recovery can be identified: Go-Back-N and selective repeat
* In a Go-Back-N protocol, the sender is allowed to transmit multiple packets (when available) without waiting for an acknowledgment, but is constrained to have no more than some maximum allowable number, N, of unacknowledged packets in the pipeline
* TCP has a 32-bit sequence number
* When the window size and bandwidth-delay product are both large, many packets can be in the pipeline. A single packet error can thus cause GBN to retransmit a large number of packets, many unnecessarily.
* Selective Repeat(SR):
* Window size must be less than sequence number

3.5 Connection-Oriented Transport: TCP