- **Packet** and **network services**
  - Packet in the Internet is a generic term, which refers to the content or message and a header associated with that. Each layer adds its own header. In the internet, a transport packet is referred as a **segment**, an IP packet (network layer packet) as a **datagram**, a link layer packet such as Ethernet packet as a **frame**. Noticing when a transport layer receives a packet from an application layer through a socket, it already contains the application layer header. Similarly, when a transport protocol hands a segment to IP layer, it already contains the transport header, and IP adds its own header (IP header we will discuss in Chapter 4); similarly Ethernet will add Ethernet header (Chapter 5)
  - Transport protocols run on top of network layer protocols. In the Internet, the network layer protocol is IP, which delivers IP datagram from a source machine (source IP address) to a destination machine (destination IP address) **unreliably**. This is referred as "**best effort**" service.

- **Transport services**
  - A logical communication (channel) between two processes on two different hosts. Similar to application protocols, it does not concern or involve with how a packet travels through the Internet (route) and how it is delivered to another host (discussed in Chapter 4). So a transport protocol is referred as an **end-to-end protocol**.
  - At a sender side, it usually breaks message(s) into multiple segments (transport packets); at a receiver side, it assembles segments before passing to upper layer application protocols.
  - A transport protocol relies on the services from a network layer, which is supposed to deliver a packet from one host to the other, i.e., communications between hosts. In the Internet, this communication is unreliable, that means the packets are subject to both errors and losses.

- **Multiplexing** and **Demultiplexing**
  - The minimum function that a transport layer protocol has to provide is **multiplexing** and **demultiplexing**, i.e., multiplexing at a sender and demultiplexing at a receiver. Demuplexing implies that a packet received by a host needs to be demuplexed and then delivered to the corresponding process (socket) identified by port number(s) and IP address(es).
  - In connectionless multiplexing and demultiplexing such as in UDP, a UDP socket is identified by a **two-tuple**, *destination IP address* and *destination port number*. So one UDP socket receives and handles segments from different machines (different source IP address) and from different processes of a same machine (different source port number). See an example on Slide 3-10/11.
  - In connection-oriented multiplexing and demultiplexing such as in TCP, a TCP socket is identified by a **four-tuple**, *source IP address, source port*

number, destination IP address and *destination port number*. So when a TCP segment arrives at a host, the host uses these four values to direct (demultiplex) a segment to the appropriate TCP socket (Slide 3-12/14).

- **UDP**
  - UDP exactly provides the minimum function by adding source port and destination port in the UDP header.
  - The checksum in the UDP is for error detection. You will see in Chapter 5 that many link layer protocols also provide error detection as well.
  - The length field specifies the UDP segment length in term of byes including the 8 bytes header. Since there are only 2 bytes (16 bits) in the length field, so the maximum UDP segment size is 64K bytes (including UDP header).

- **On Finite State Machine (FSM)**
  - *State*：the state that a sender or a receiver is in
  - *Event*: that might cause a state transition
  - *Action*: operation if any when an event occurs during possible state transition
  - Slide 3-26 rdt1.0 is a simple example of this, in which the sender and the receiver each has only one state.

- **rdt2.X**
  - This assumes no packet loss, i.e., packets always arrive either correctly or corruptly
  - **Stop and Wait protocol:** a sender sends out one packet until it receives ACK from a receiver before the sender can send out another packet. This results in poor throughput or link utilization.
  - rdt2.0 introduces ACK and NAK, only data packet can be corrupted, ACK and NAK are never corrupted (slides 27-31)
  - rdt2.1 handles the case when the ACK or NAK can be corrupted. If ACK and NAK are corrupted, a sender has no way of knowing whether it is an ACK or NAK, thus the sender can only *retransmit* the same packet. This can be a possible duplication at the receiver if the corrupted acknowledgment is ACK, a sequence number is introduced in the packet (0 and 1) (slides 32-35)
  - Rdt2.2 is essentially the same as rdt2.1 functionally, except it adds a sequence number in the ACK, i.e., ACK(0) or ACK(1), to remove the NAK (slides 36-37). Here, ACK(0) implies that the receiver has received packet 0 and it also implies that it has not received packet 1 (as NAK).

- **rdt3.0**
  - Packets can be corrupted and lost; this is still a *stop and wait protocol.*
  - A timer is added to handle a packet or an ACK loss. When time out, a sender retransmits the packet.
  - Functionally, when a sender receives corrupted ACK or ACK that it is not

expected (same effect as the NAK), the sender can initiate the retransmission and restarts the timer. This has the potential of introducing more duplicated packets. Please check the right example on slide 40-41 and think this over!

- rdt3.0 might **not** always work if one of the ACKs is delayed until two packets later with the same sequence, or it will fail if ACK(1) flips the sequence bit to ACK(0), such cases occur with extremely small probability, this protocol ignores those cases. In reality, seen from GBN and SR protocols, this will not occur with a large sequence number.

- **Go-Back-N (GBN) protocol**
  - Window size (N) for now can be considered as a constant, which is the maximum number of packets that a sender can send without receiving any acknowledgment. If you recall the calculations (slides 3-42/45), this determines the link utilization
  - **The receiver** only accepts a packet that it is expecting, i.e., the sequence number of the packet is *expectedseqnum*; it will drop all other out of order arrived packets. This is very simple in that the receiver has no buffer
  - **The sender:** 1) sends packets with the sequence number in the range of [send_base, send_base+N-1], and the parameter *nextseqnum* is updated; 2) keeps only one timer for the earliest sent packets that has yet been acknowledged, so a timeout will trigger the sender to resend all those packets yet acknowledged; 3) when ACK(n) is received, which is **cumulative**, i.e., indicating all packets up to n has been received by the receiver. The parameter send_base is updated, and a new timer is started if there are still outstanding packets transmitted but unacknowledged.

- **Selective Repeat (SR)Protocol**
  - Key features: individual packet acknowledgment as supposed to the cumulative acknowledgement in GBN; receiver buffers all out of order arrived packets up to a window size N
  - **The receiver**: 1) packet with the sequence number n in the range of [rcv_base, rcv_base+N-1] is correctly received, send ACK(n), deliver the packet to upper layer protocol and adjust parameter rcv_base if necessary (if this is an in-order arrival); 2) packet arrival with sequence number n [rcv_base – N, rcv_base – 1], send ACK(n). This occurs if ACK is lost; 3) ignore
  - **The sender**: 1) packet from upper layer with sequence number n [send_base, send_base + N-1], send the packet and start the timer for this packet; 2) timeout, resend the packet; 3) ACK received, mark the buffer as received and update the parameter if necessary.
  - The SR protocol is much more complex that GBN protocol, and only when packet(s) with correct order are acknowledged or received, it can move the send_base or rcv_base at either side.

- **Window size**
  - Window size specifies the maximum number of packets that a sender can send without waiting for the acknowledgement. This determines the amount of traffic that a sender can inject into the network.
  - This is bounded by the *RTTxbandwidth*. Since by RTT time, the first acknowledgment is supposed to come back to the sender if we ignore the transmission time of the ACK. In a high speed network, this can be very large, i.e., potentially excessive amount of traffic injected into the network by a single node.
  - If you recall the calculation on Slide 3-45, this yields the link utilization

- **Sequence number and window size**
  - The window size (W) is the maximum number of packets that a sender can send out without receiving any acknowledgment.
  - Sequence number, N = 2^n, where n is the number of bits in the sequence number. Suppose n=4, the sequence number is 16, i.e., 0-15.
  - Slide 3-55 (SRprotocol) illustrates the fact that in the worse case, if a full window of packets are received by a receiver, and all ACKs are lost. The range of the sequence number has to be at least twice more than the window size. In another word, N or (2^n) > 2W.

- **Summary of the basic mechanisms in reliable data transfer**
  - **Checksum**: detect errors in packets
  - **Timer:** to retransmit packets
  - **Sequence number**: for receiver to detect out of order arrivals or lost of packets
  - **ACK**: ack the reception of packets individually or cumulatively
  - **Window** and **pipelining**: allow a sender to send multiple packets without waiting for the ACK to increase the link utilization

- **TCP Basics**
  - T**CP Header**: source and destination port numbers (multiplexing and de-multiplexing), checksum, 32-bit sequence number and ACK number for TCP reliable data transfer, 16-bit receive window for TCP flow control, 4-bit header length specifies the TCP header (can be a variable, standard 20 bytes, up to $4*2^4$ = 64 bytes), option field, and flag fields.
  - TCP treats the data as an unstructured, but ordered, stream of bytes. **Sequence number in TCP** is the byte-stream number of the first byte in the TCP segment. ACK is **piggybacked** (included in the date transferred in the other direction). See Slide 3-60

- **Round-Trip Time Estimation and Timeout**
  - Exponential weighted moving average (EWMA) shown on Slide 3-62. The

algorithm puts more weight on recent samples than on old samples, and this helps to smooth the instantaneous randomness of the samples.

  - *DevRTT* measures the variability of the sample RTT, i.e., the EWMA of the difference between the SampleRTT and EstimateRTT (Slide 3-63).
  - The timeout interval (TimeoutInterval) is set as a function of Estimate RTT and DevRTT (Slide 3-63). The intuitions are: 1) the TimeoutInterval should be larger than EstimateRTT, 2) it cannot be too large than EstimateRTT otherwise TCP would not quickly retransmit the lost segments; 3) DevRTT measures the fluctuation in the SampleRTT, higher the fluctuation, the larger margin that TimeoutInterval should be over the EstimateRTT.

- **TCP Reliable data Transfer**
  - A hybrid of GBN (single timer and cumulative ACK) and SR (receiver can buffer out of order arrivals)
  - **Receiver**: 1) in-order arrival with no other in-order arrival earlier, delay ACK; 2) in order arrival with other earlier in-order arrival(s), send ACK; deliver packets to upper layer, release buffers and receiver window moves; 3) out-of-order arrival with gap (with lower sequence number packet not arrived), send duplicated ACK (recall the ACK is cumulative); 4) out-of-order arrival filling in gap (with all lower sequence number packets arrived), send ACK; deliver packets to upper layer, release buffers and receiver window updates.
  - **Sender**: 1) receive packet from upper layer application, if within window range, send the packet; if this is the first unacked packet, start the timer; 2) time out, resend the packet (only one packet, as the timer is set for the not-yet-ack packet with the smallest sequence number), restart the timer; 3) ACK, duplicated ACK, do nothing, if not duplicated ACK (sequence number extracted from the packet y > SendBase), move the sliding window (SendBase = y, that also release the buffer at the sender side), restart the timer if there are still unacked packets.

- **Fast Retransmit**
  - A sender often sends a large number of segments back to back, if one segment is lost, there will be likely many back-to-back duplicate ACKs.
  - If a sender receives three duplicate ACKs, it takes as an indication that the segment following the ack'ed segment is lost, so to retransmit the segment without waiting for the timeout (Slide 3-71 – Slide 3-72).

- **TCP Flow Control**
  - To prevent a sender from transmitting too many packets to overflow a receiver
  - RcvWindow = RcvBuffer – [LastByteRvcd – LastByteRead] is simply the amount of free buffers available at the receiver
  - As long as a sender does not transmit more than RcvWindow (in the ACK

from receiver to sender), the receiver will not be overflowed.

- **TCP Connection Management**
  - Connection establishment: the three-way handshaking protocol.
  - The example given in TCP connection closure is for a client to initiate the end connection. Unlike the connection establishment, the server can also initiate the end connection or client and server can do this simultaneously. The important aspect in connection termination is that each side has to close the connection separately, by sending FIN and receiving ACK.

- **The Causes of Congestion**
  - **Scenario I**: two streams share the link with infinite buffer, where each can send up to C/2 rate (Slide 3-86). The **first cost of the congestion** in this case is *the large queuing delays experienced by packets when the arrival rates near the link capacity.*
  - **Scenario II**: (Slide 3-87 - Slide 3-92) two streams share one link with finite buffer. (1) In the ideal case, when a sender only sends packets when the buffer is free (magically), each can send up to C/2 rate. (2) In a bit more realistic case, a sender retransmits a packet when it knows the packet is lost for certain, in this case the real rate is about C/3. The **second cost of congestion** is *that the sender must perform retransmission in order to compensate for lost packets due to buffer overflow*. In the realistic case, a sender may time out, thus retransmits a packet that has been delayed in the queue but not yet lost. The real rate is about C/4. The **third cost of congestion** is *the unneeded retransmission by the sender in the face of large delays may cause a router to use its link bandwidth to forward unneeded copies of a packet.*
  - **Scenario III:** (Slide 3-83 - Slide 3-94) The heavy traffic from node A to this link might cause no transmission for the traffic generate by node B in this example, so the link B -> A can be wasted. *So* **the fourth cost of congestion** *is related to dropping a packet due to congestion, i.e., when a packet is dropped along a path, the transmission capacity that was used at each of the upstream links to forward that packet to this point at which it is dropped ends up having been wasted.* One solution to this could be for a router to give priority to packets that have already traversed certain number of upstream routers.

- **The basic principles of congestion control**
  - Congestion control at the essence has to be done at the sender, which upon noticing or suspecting congestion has to slow down the transmission.
  - One of the difficulties is how to detect congestion. There are two ways in doing so; one is a router-assisted approach or network-assisted approach, in which routers send explicit congestion indication bit. The other more common way (used in the Internet) is for a sender to probe and “detect”.

- The congestion detection in the Internet is done at a sender, either receiving duplicated acknowledgement or experiencing a timeout. In the actual implementation, the Internet congestion control treats these two slightly different; as it argues that a timeout is more likely a packet loss caused by the network congestion.

- **TCP congestion mechanism**
  - The rational of AIMD mechanism is to explore the maximum transmission rate by the network while reacting to possible congestion.
  - Slow start simply speed up the initial process of reaching the maximal transmission rate
  - Refinement shown on Slides 3-103 is one of the examples as there are many small variations of the TCP congestion control (see additional reading on TCP Reno, TCP Tahoe, TCP Vegas). In this refinement, it makes a distinction in treating the three duplicated acknowledgment and timeout. Noticing in both cases, the Threshold is set to the half of the current CongWin, but with timeout, CongWin is set back to 1 MSS; with duplication, CongWin is set to the Threshold.