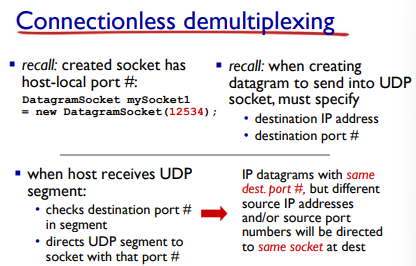
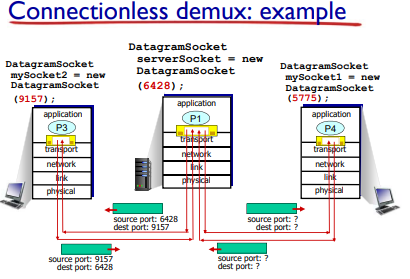
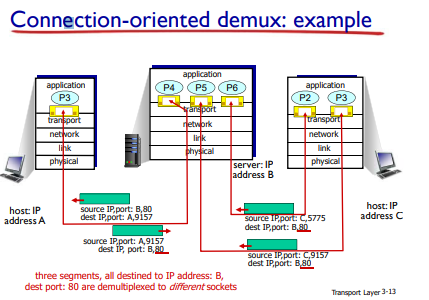
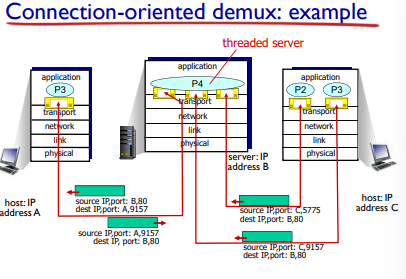
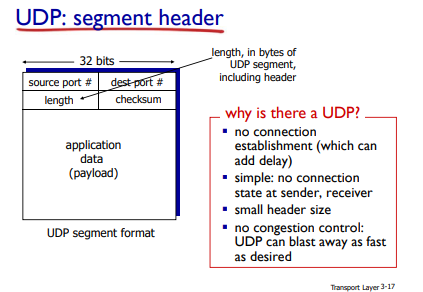
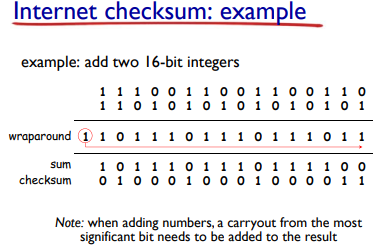
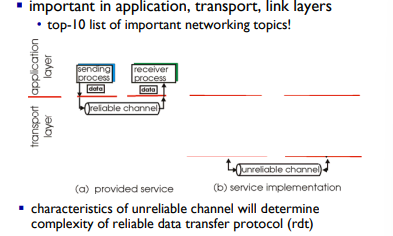
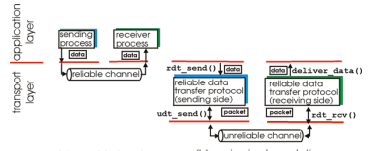
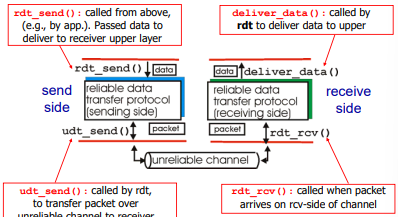
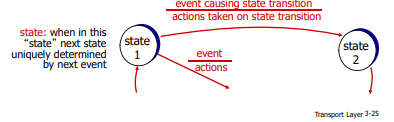
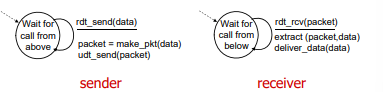
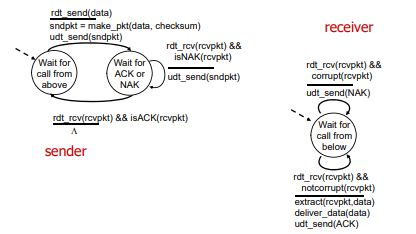
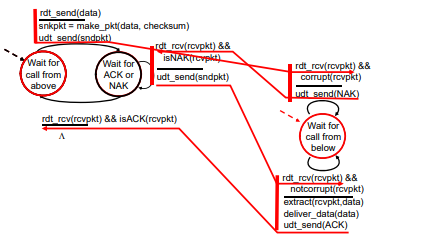
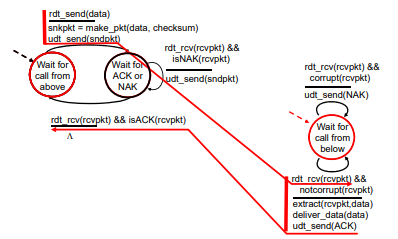
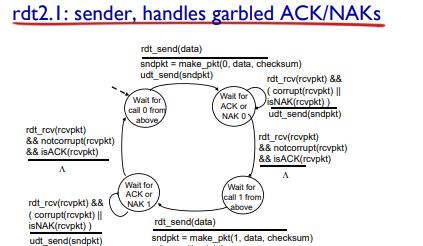
* Transport Layer
  + Provide logical communication between app processes and running on different hosts
  + Transport protocols run in end systems
    - Send side: breaks app messages intos egments, passes to network layer,
    - Rcv side: reassembles segments into messages, passes to app layer
  + More than one transport protocol available to apps
    - Internet: TCP and UDP
  + Network layer: logical communication between hosts
  + Transport layer: logical communication between processes
    - Relis on enhances, network layer services
  + Reliable, in order deliver(TCP)
    - Congestion control
    - Flow control
    - Connection setup
    - Unreliable, undordered deliver: UDP
    - Services not available:
      * Delay guarantees
      * Bandwidth guarantees
* Multiplexing/demultiplexing
  + Multi at sender - handle data from multiple sockets,a dd transport header (later used for demultiplexing)
  + Demulti at receiver - use header info to deliver received segments to correct socket
  + How Demultiplexing works:
    - Host receives IP datagrams
      * Each datagram has source IP address, destination IP address
      * Each datagram carries on transport-layer segment
      * Each segment ahs source destination port number
    - Host uses IP address & port numbers to direct segment to appropriate socket.
  + Connectionless Demultiplexing (UDP)
  + 
  + Connection-orientated demux (TCP)
  + 
  + 
* UDP: User Datagram Protocol
  + “No frills’. “Bare bones” internet transport protocol
  + “Best effort” service, UDP segments maybe:
    - Lost
    - Delivered out of order to app
  + Connectionless
    - No handshaking between UDP sender,receiver
    - Each UDP segment handled independently of others
  + UDP Use:
    - Streaming multimedia apps(loss tolerant, rate sensitive)
    - DNS
    - SNMP
    - Reliable transfer over UDP:
      * Ad reliability at application layer
      * Application-specific error recovery
  + 
  + UDP Checksum
    - Goal: detect errors (eg flipped bits) in transmitted segment
    - Sender:
      * Treat segment contents, including header fields as sequence of 16-bit integers
      * Checksum: addition (one’s complement sum) of segment contents
      * Sender puts checksum value into UDP checksum field
    - Receiver:
      * Compute checksum of received segment
      * Check if computed checksum equals checksum field value:
        + No - error
        + Yes - no error
      * 
* Principles of reliable data transfer:
  + IMport in application, transport, link layers,
  + Characteristics of unreliable channel will determine complexity of reliable data transfer protocol (rdt)
* 
* 
* Reliable data transfer: getting started
  + 
* Incrementally develop sender, receiver sides of reliable data transfer protocol(rdt)
* Consider only unidrectional data transfer
  + But control info will flow in both directions
* Use finite state machinesFSM) to specific sender,receiver
* 
* Rdt1.0: reliable transfer over a reliable channel
* Underlying channel perfectly reliable
  + No bit errors
  + No loss packets
* Separate FSMs for sender,receiver
  + Sender sends data into underlying channel
  + Receiver reads data from underlying channel
  + 
* Rdt2.0: channel with bit errors
  + Underlying channel ma flip bits in packet
    - Checksum to detect bit errors
  + The question: how to recover from errors?
  + Underlying channel may flip bits in packet
    - Checksum to detect bit errors
    - The question: how to recover from errors
      * Acknowledgements (ACKs): reecevier explicitly tells sender that pkt received OK.
      * Negative acknowledgments(NAKs): receiver explicitly tells send that pkt had errors.
    - New mechanisms in rdt2.0 (beyond 1.0)
      * Error detection
      * Feedback: control msgs ACK,NAK from receiver to sener
    - 
* Rdt2.0: operation with no errors
* 
* Rdt2.0 Fatal Flaw
  + ACK/NAK corrupted:
    - Sender doesn’t know what happened at receiver
    - Can’t just retransmit: possible duplicate
  + Handling duplicates:
    - Sender retransmits current pkc if ACK/NAK corrupted
    - Sender adds **sequence number** to each pkt
    - Receiver discards(doesn’t deliver up) duplicate pkt.
  + Stop and Wait:  
     Sender sned son epacket, then waits for receiver response.
  + 
  + 