Transport Services and Protocols

Andy Carpenter

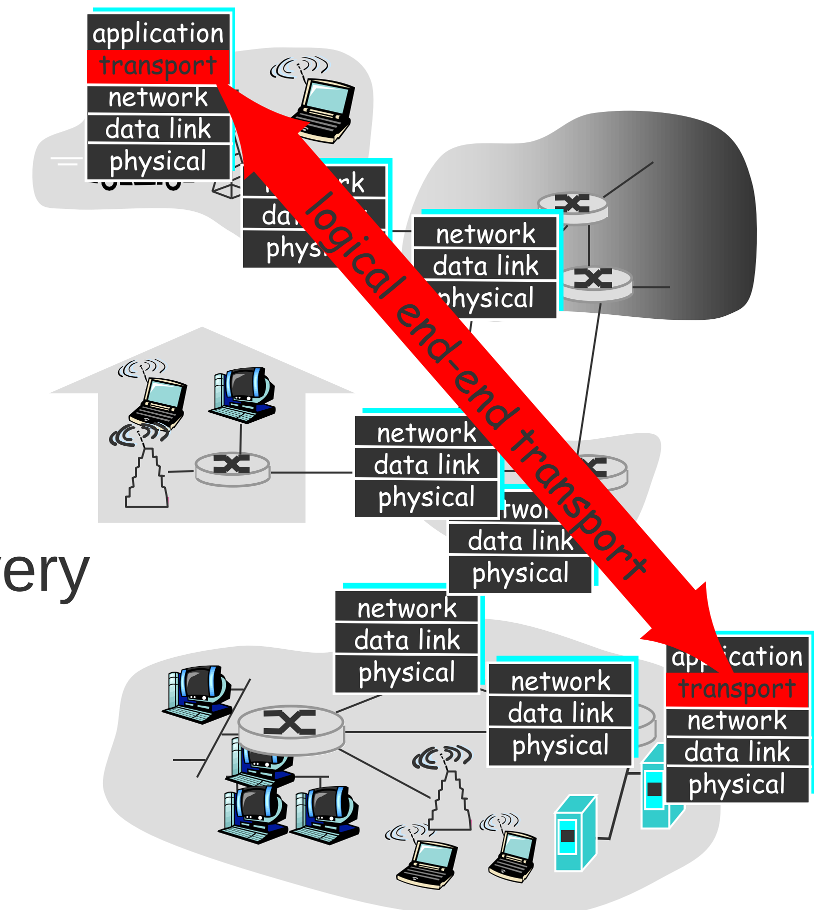
(Andy.Carpenter@manchester.ac.uk)

Transport Services and Protocols

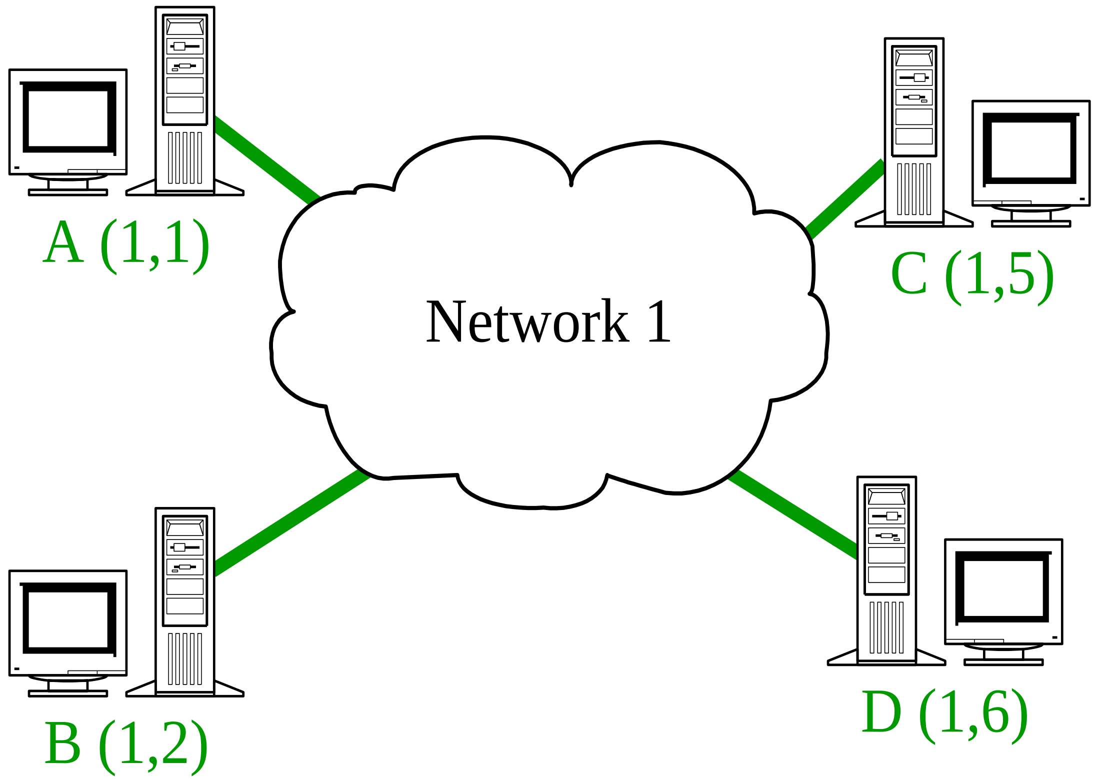
- Moves data between elements of applications
  - process-to-process
- Must identify source and destination processes
- Defines service to application (next higher) layer
  - connection-oriented or connectionless
  - Quality of Service (QoS) parameters
- Isolates higher layers from:
  - technology, design and imperfections of network
- For connection oriented, must address:
  - error control, reliability, sequencing, flow control
- Examine: UDP, TCP

Internet Transport Layer Protocols

- 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:
  - delay guarantees
  - bandwidth guarantees

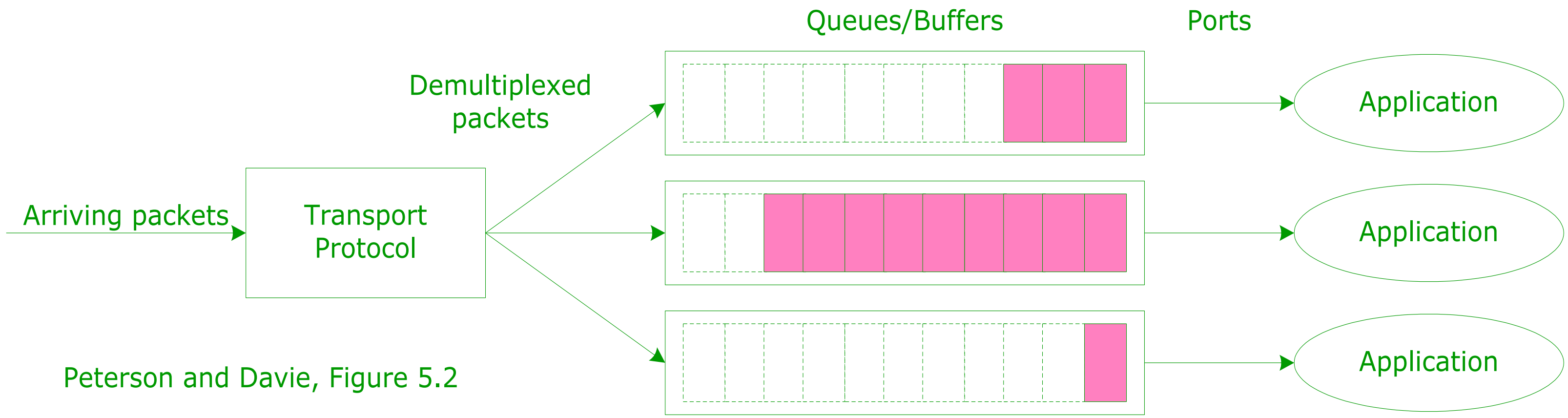


Ports: Connecting to a Service



- Packets are sent to ports with attached applications
- Data to other ports will generate an error message

Ports: Implementation



- Port implementation is operating specific specific
- Typically a message queue or data buffer
- When application wants data, fetches from queue/buffer
- If queue/buffer is empty, blocks awaiting data
- What happens when queue/buffer is full?

User Datagram Protocol (UDP) [RFC 768]

- "no frills" Internet transport protocol
- Often used for streaming multimedia applications
  - loss tolerant
  - rate sensitive
- Other uses:
  - DNS, SNMP
  - NFS
  - o/s applications

Why is there a UDP?

- no connection establishment (which can add delay)
- simple: no connection state at sender, receiver
- small segment header
- no congestion control: UDP can blast away as fast as desired

UDP: Service Model

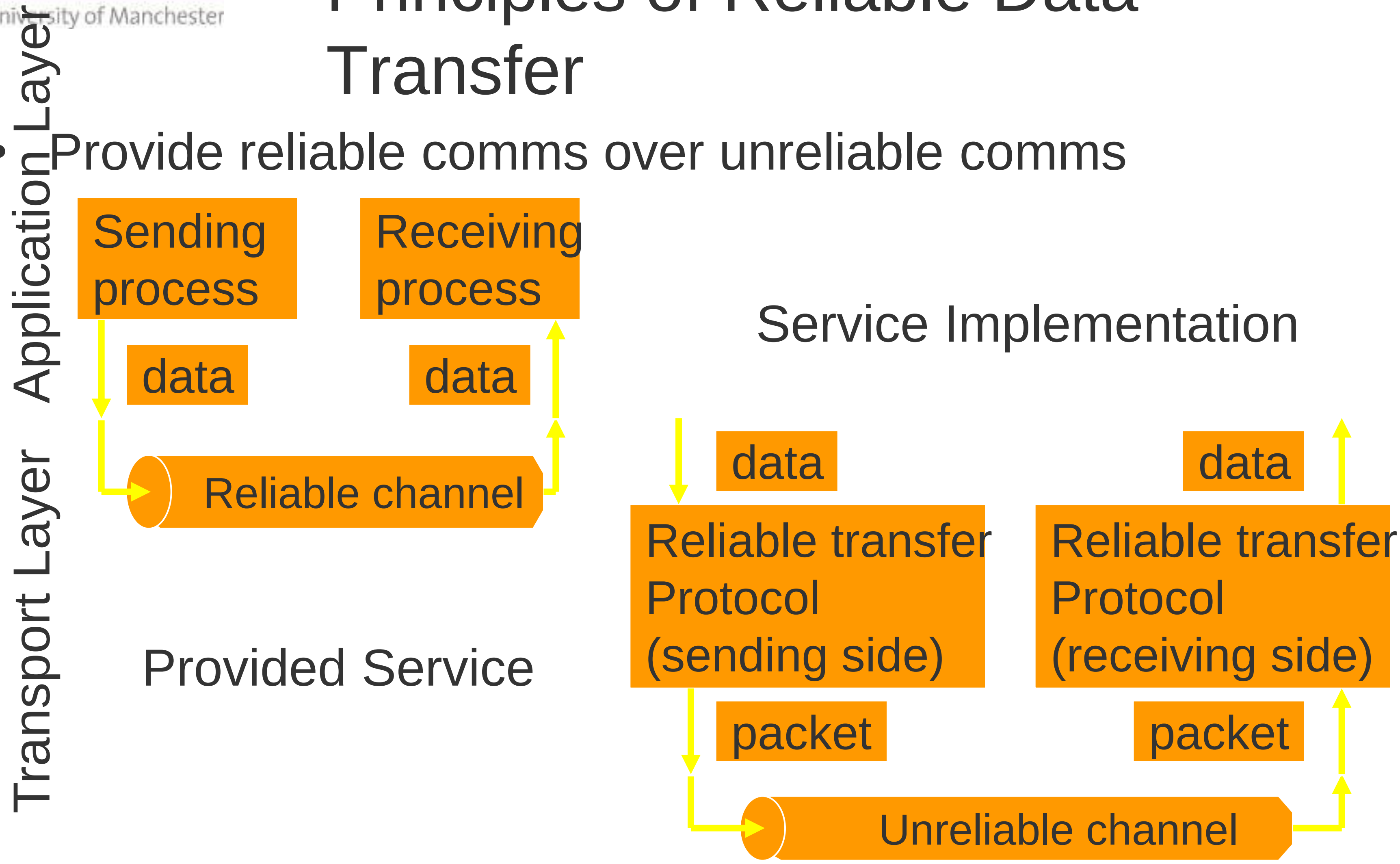
- Inherits IP “best effort” service; e.g.:
  - connectionless, unreliable, unordered
- UDP datagrams are:
  - independent; unsequenced
  - not acknowledged, not flow controlled
- Applications must implement:
  - error detection and recovery, flow control
  - application-specific error recovery!
- Effectively just adds support for multiple applications to underlying network service model

UDP: Error detection

- Uses checksum
- Sender:
  – segment contents treated as sequence of 16-bit integers
  – checksum: 1's complement addition of contents
  – sends checksum in UDP datagram
- Receiver:
  – computes checksum of received segment
  – compares received and computed checksums:
    - different - error detected
    - no error detected – *nonetheless, maybe errors*

Principles of Reliable Data Transfer

- Provide reliable comms over unreliable comms

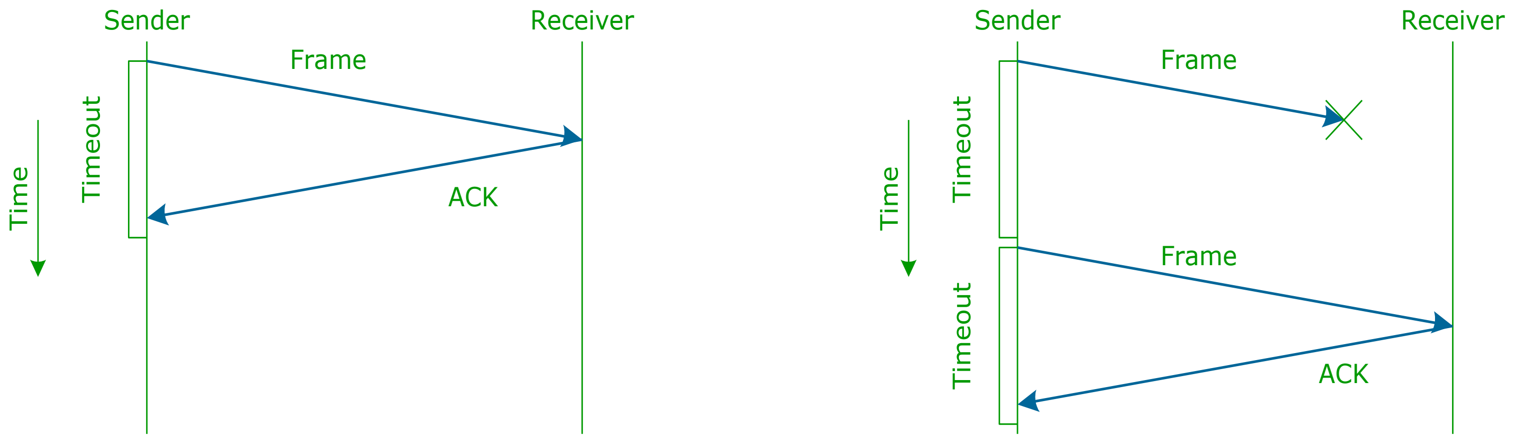


- Complexity depends on characteristics of unreliable channel

Recovering from Errors

- Recovery uses two mechanisms:
  - acknowledgements and timeouts
- Acknowledgement (**ACK**) is control packet from
  - receiver to transmitter of data packet being ACKed
- Receipt of ACK confirms delivery of data
- If ACK not receiver within **timeout**:
  - transmitter of data retransmits data; needs copy
- Process called automated repeat request (ARQ)
- ARQ mechanisms: **stop-and-wait**, **sliding window**

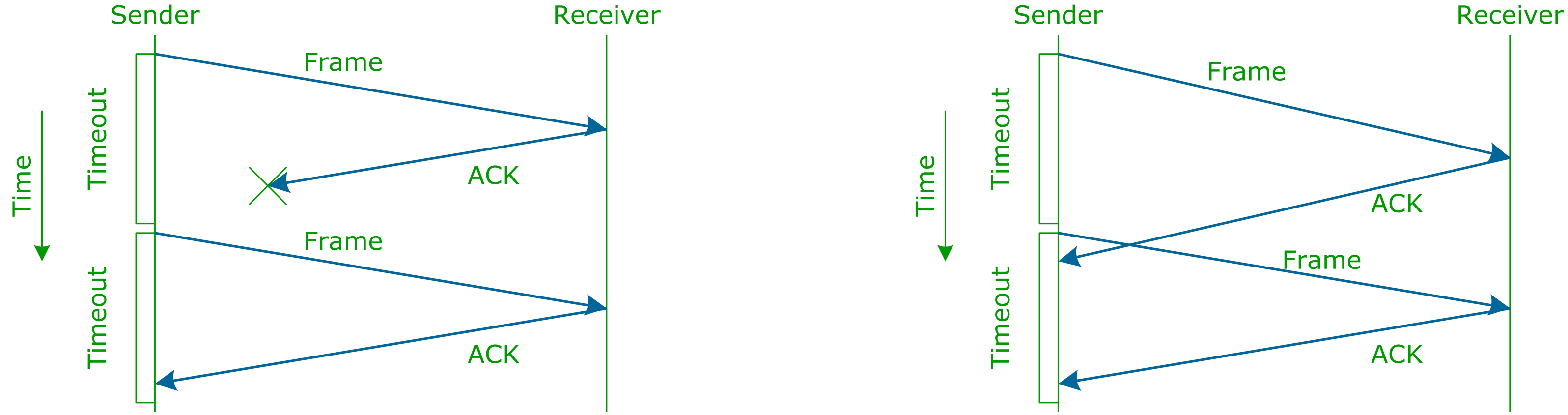
Reliability: Stop-and-Wait



Peterson and Davie, Figure 2.19

- Wait for acknowledgement before sending next packet
- Normal operation, ACK received before timeout expires
- If not, data packet is retransmitted and, hopefully, ACKed

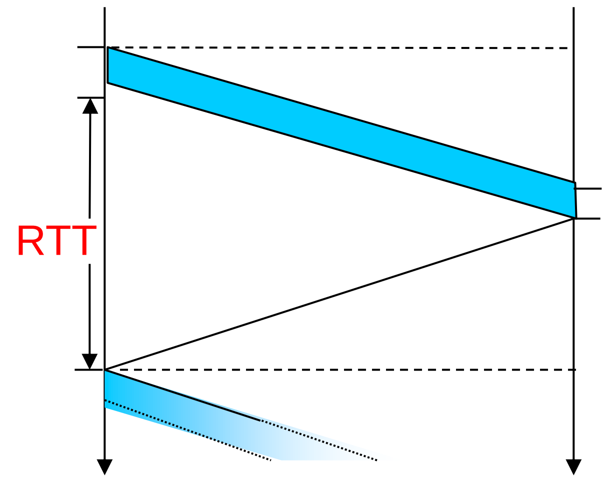
Reliability: Stop-and-Wait



Peterson and Davie, Figure 2.19

- Loss of ACK, or late arrival, causes retransmission
- Duplicate packet is received, but
- receiver believes that it is receiving a new packet
- Duplicates can be detected using **sequence numbers**
- One-bit sequence numbers used; 0 or 1

Reliability: Stop-and-Wait - Utilisation

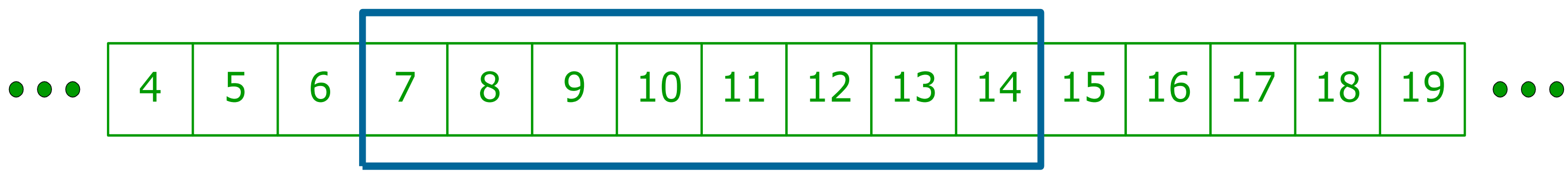


- For example, 1Gbps link with 15ms propagation delay and 8Kb packet
  - transmit delay (d_{trans}) = L/R = 8K/1G = 8μsec
  - utilisation = d_{trans} / (RTT + d_{trans})

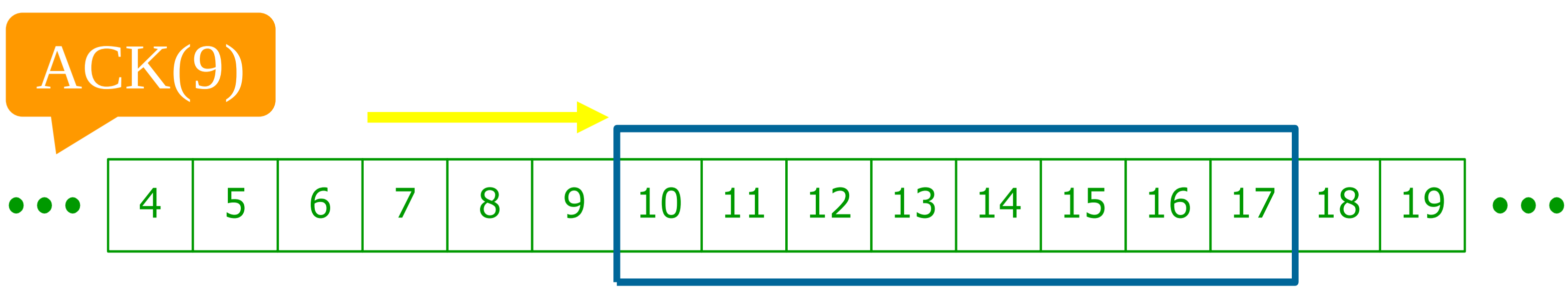
    = 8μ/(2 * 15m + 8μ) = 0.00027 = 0.027%
- Network protocols limit use of physical resources

Reliability: Sliding Windows

- Allows multiple packets to be unacknowledged
- Window size is number of unacked packets allowed
- Duplicate packets detected using sequence number
- For example, received ACK for 6 and window is 8
  - can send packets 7-14

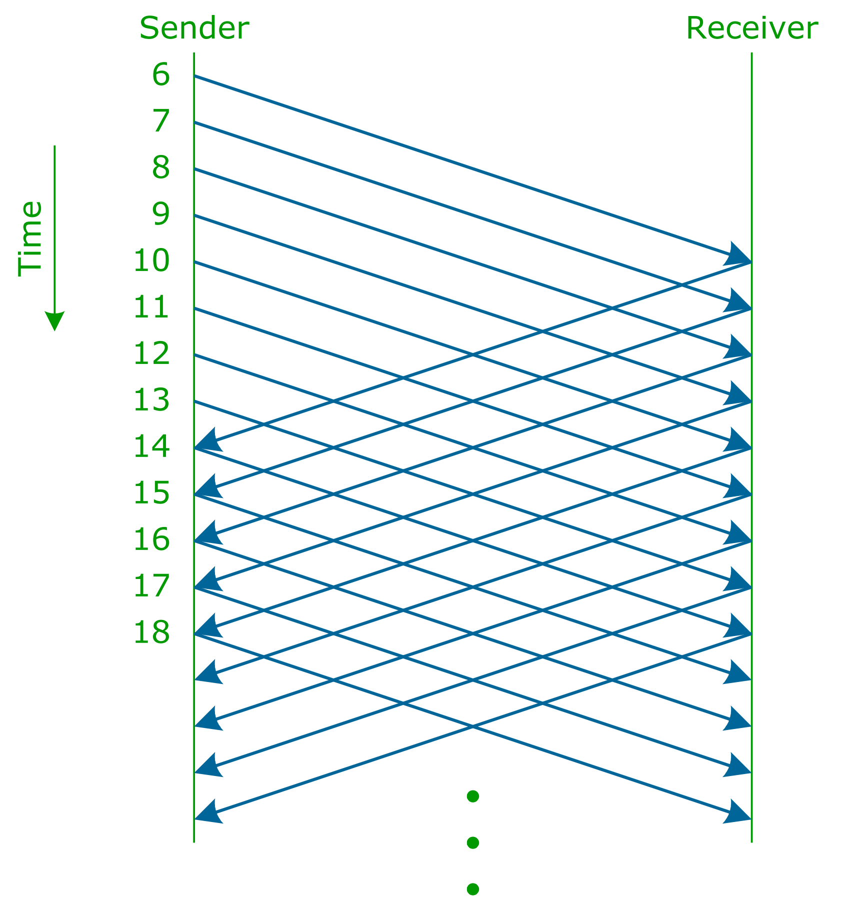


- When new ACK is received, window moves



Reliability: Sliding Windows - Utilisation

- Example scenario:
  - 1.5Mbps link
  - 45ms RTT
  - 1KB frames
  - received ACK for 5
  - windows size 8
- Can send packets 6-13
- ACK for 6 arrives as
  - want to send packet 14
- Utilisation 100%

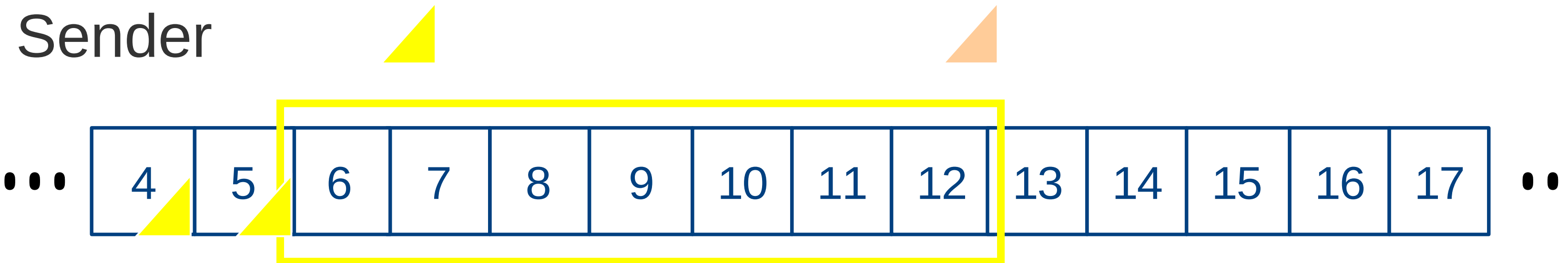


Reliability: Sliding Windows - Approaches

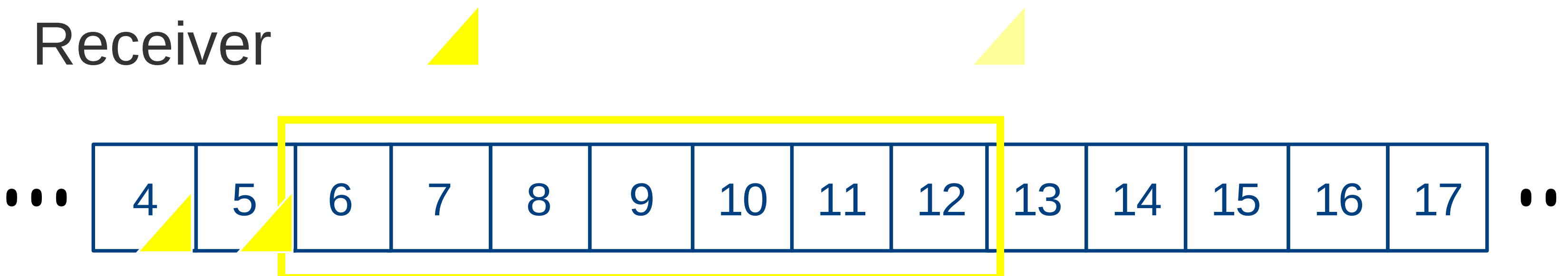
- What if expecting packet 8 and get packet 9?
- If packet 8 delayed, when arrives can send ACK(9)
- If lost, timeout will expire and will be resent
- Go-Back-N, send cumulative ACKs:
  - ACK(n) acknowledges all packets upto n
  - likely that will also get packets 9, 10, … resent
- Selective repeat:
  - explicitly acknowledges all packet
  - only unsuccessfully received packets are resent
- Could negatively acknowledge (NACK) packet 8,
  - requests retransmission without timeout expiring

Reliability: Go-Back-N

- Sender

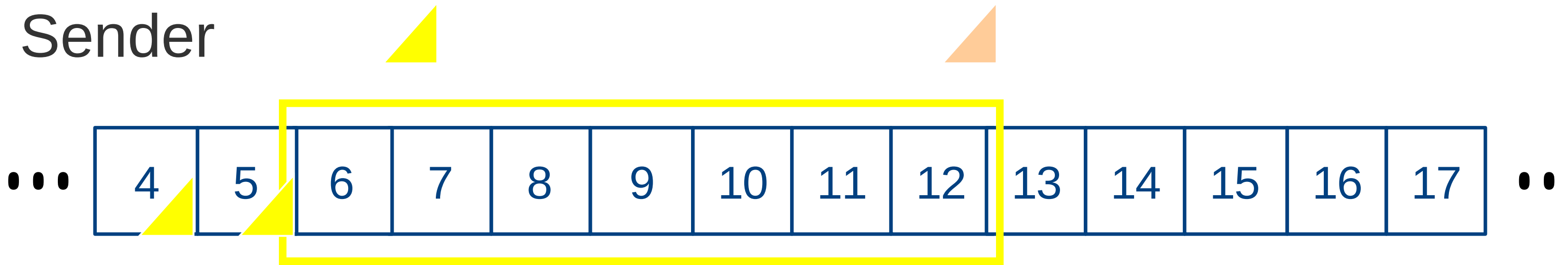


- Receiver

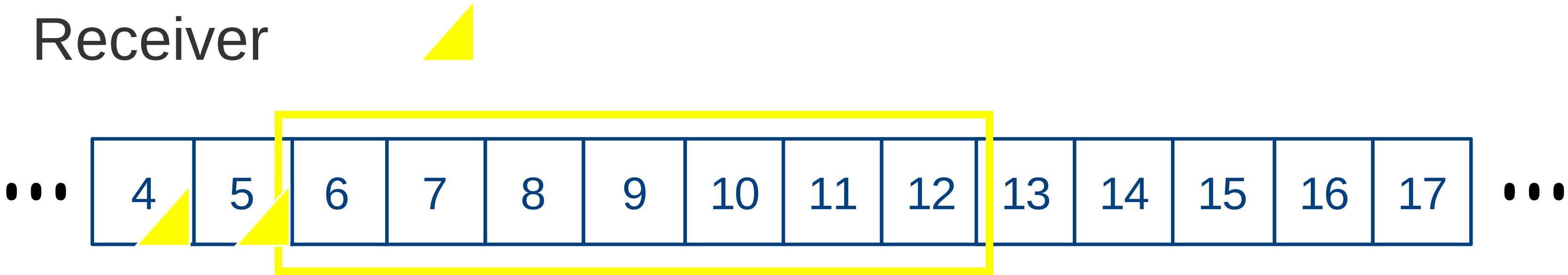


Reliability: Selective Repeat

- Sender

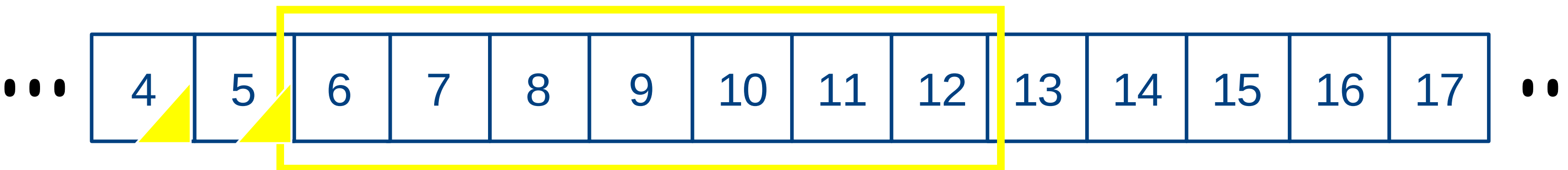


- Receiver

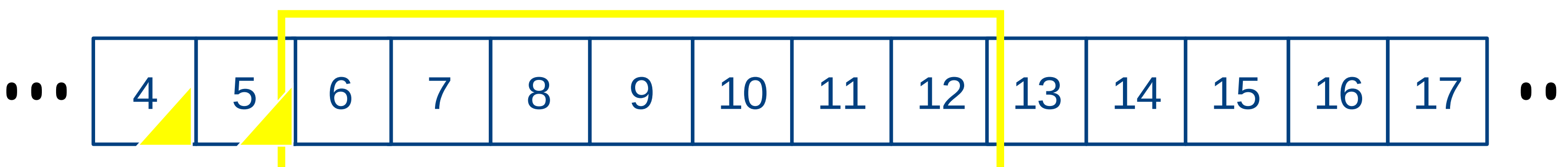


Reliability: Negative Acks

- Sender



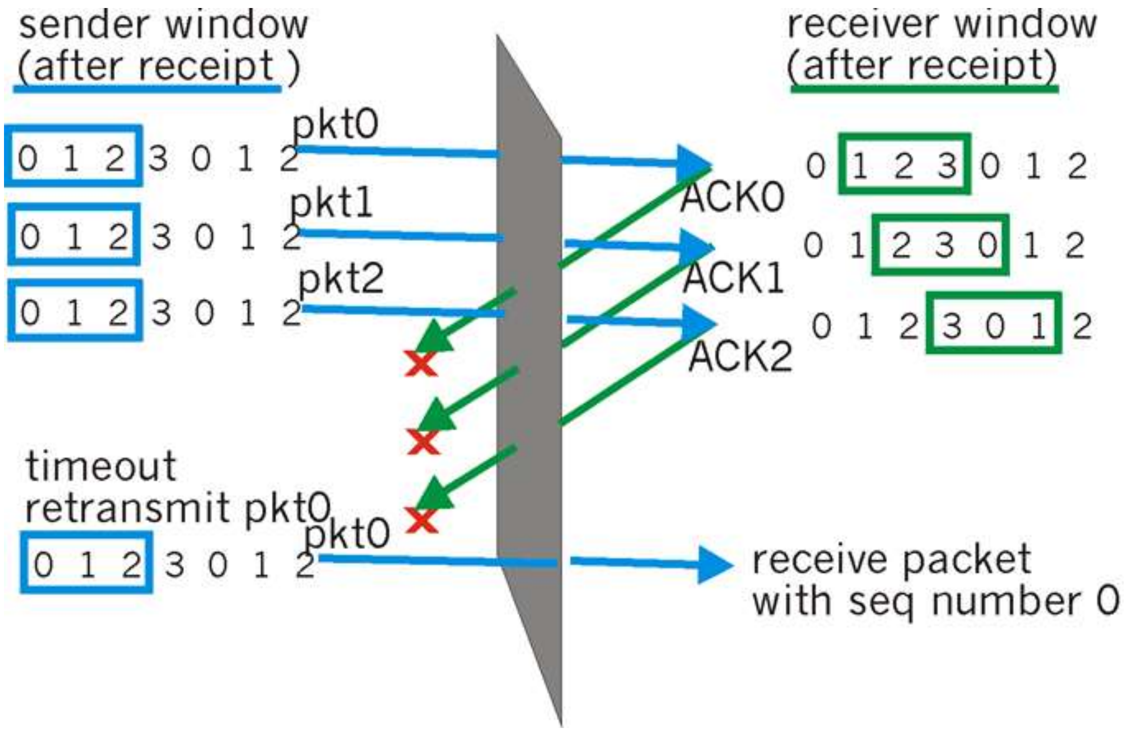
- Receiver



Sliding Windows – Finite Seq. Numbers

P&D: 2.5.2

- Sequence numbers wrap, must interpret correctly
- For example,



- Max window size = (max sequence number + 1)/2