NWEN 243

Networked Applications

Layer 4 – TCP and UDP

About the second lecturer

- Aaron Chen
- Office: AM405
- Phone: 463 5114
- Email: aaron.chen@ecs.vuw.ac.nz

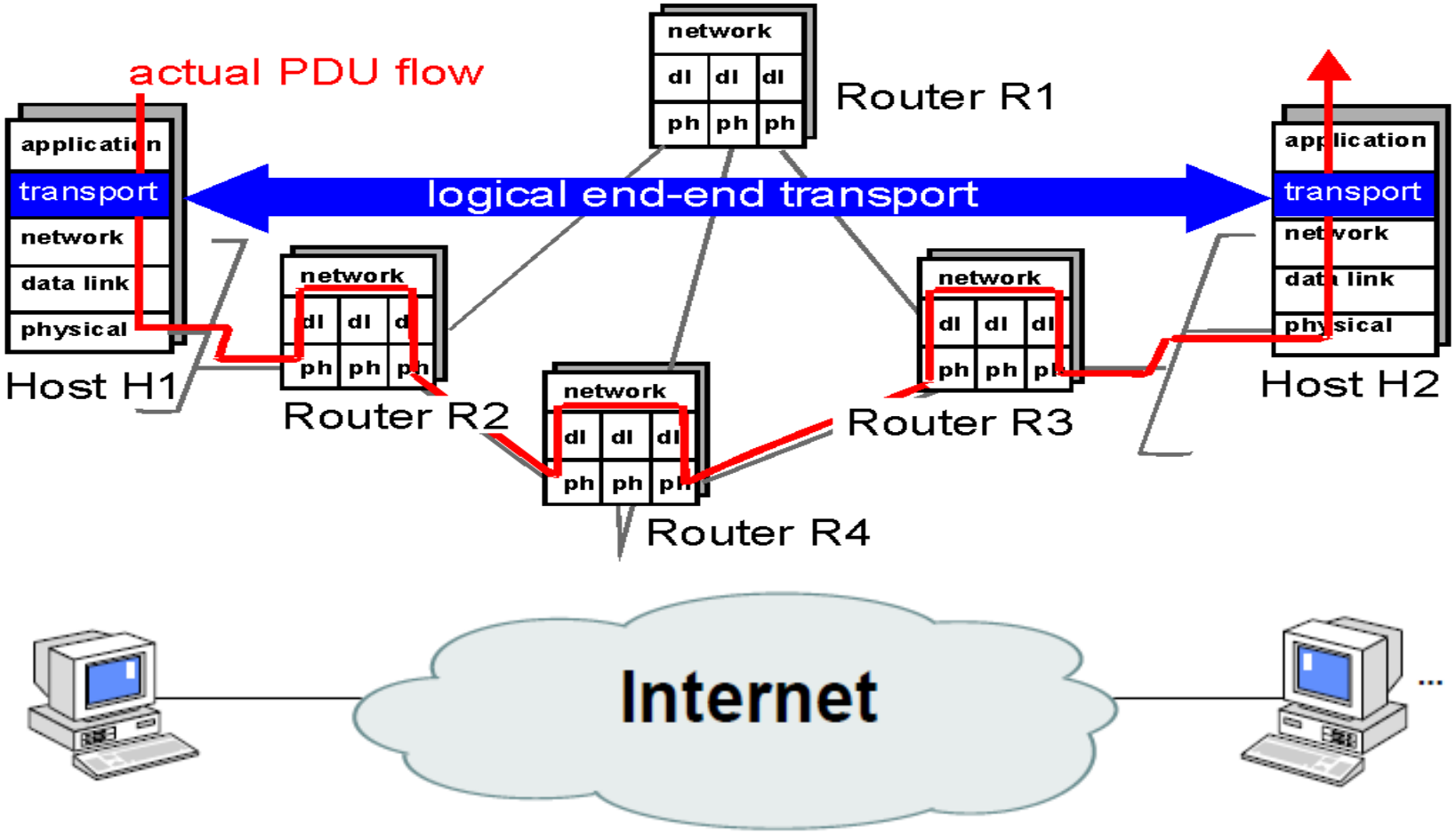
- Transport layer and application layer protocols

- Building mobile applications and Web Services

- Multimedia streaming and P2P (optional)

Transport layer protocols

- *Entire network seen as a pipe*



Transport layer port numbers

- How can ensure it is delivered to the right application on that machine?
- We need to be able to address the applications on a machine, just like we had to address the machines themselves.
- We use *port numb*ers.

IP: 192.168.0.1
Port: 3000

Layer 4 – the Transport Layer.

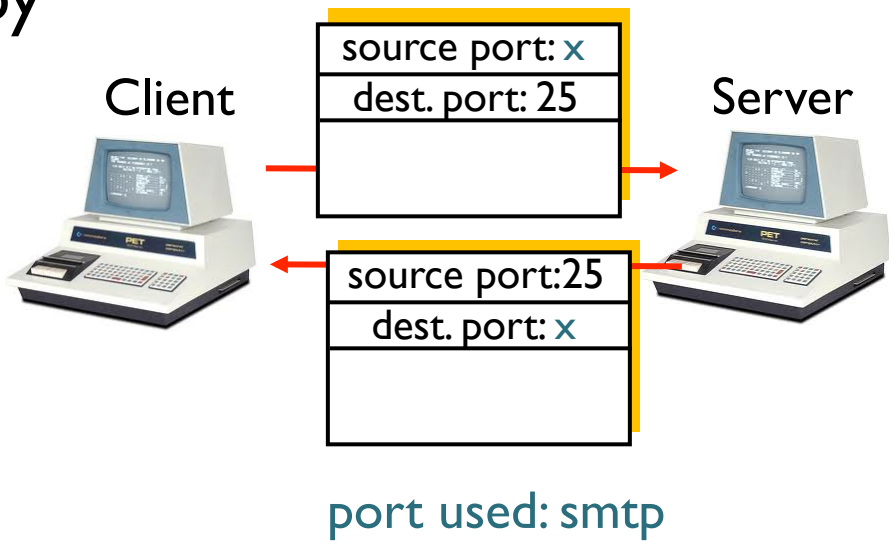
- The network layer performs machine to machine delivery of datagrams
- The transport layer performs application to application delivery.
- Ports are (16 bit) numbers (like house numbers) that form the address space of a *protocol*.
  - i.e. you can have tcp/53 and udp/53.
- A socket is a software structure associated with a port.
- An application must 'bind' (associate) a socket to a port before it can be used.
- The Socket API is the interface for applications to gain access to the network.

Socket API

- Client/server paradigm
- Two levels of service in the socket API:
  - User Datagram Protocol (UDP)
    - Best effort protocol, transmits *datagrams*.
  - Transmission Control Protocol (TCP)
    - reliable, byte stream-oriented, with capacity control, transmits *segments*.

Simple Example

- Client is assigned some random unused port x by its host.
- The SMTP server runs on well known port 25.
- Reply is to destination port x, with src port 25



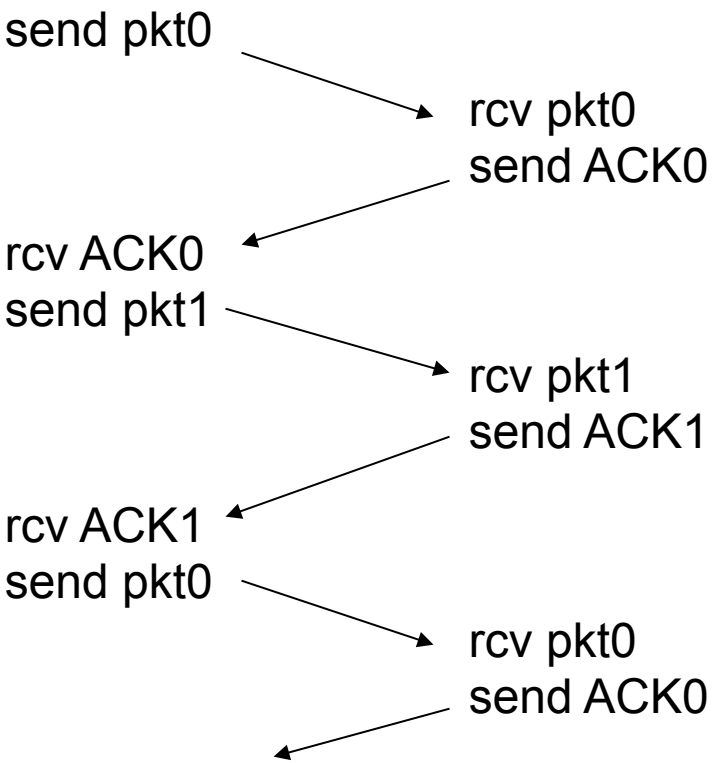
port used: smtp

Delivery Without Guarantees

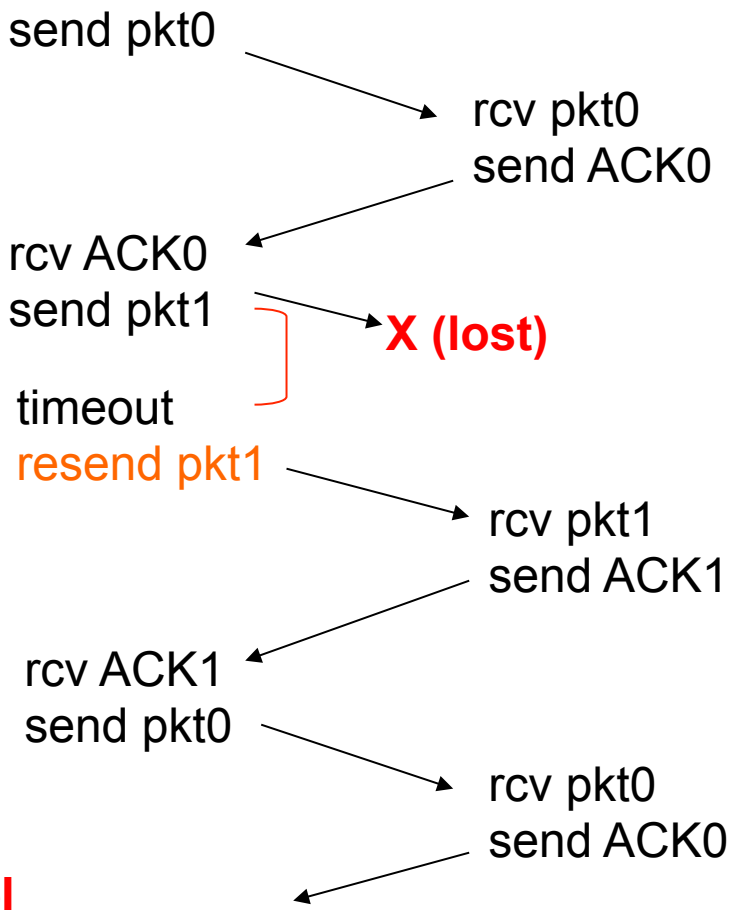
- Remember, IP Datagrams may be:
  - Duplicated,
  - Delivered out of order,
  - Lost/Discarded
  - Corrupted
- By the network layer.
- The TCP protocol will need to resolve these, while the UDP protocol ignores them.

Stop And Wait

No loss:



Lost packet:



reliable communication and **flow control**

Stop and Wait Performance

Wellington

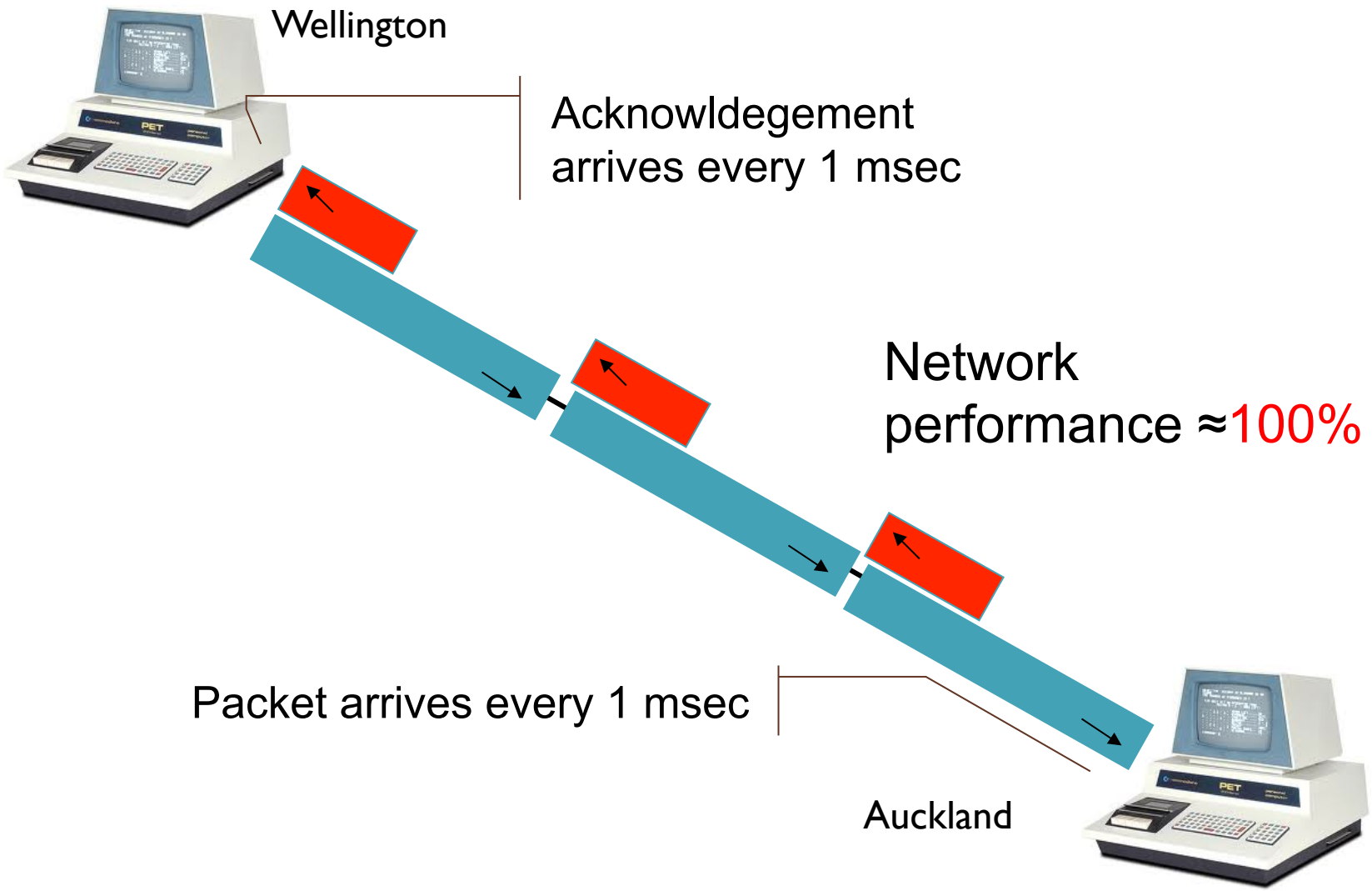
Round trip ≈ 14-30+ msec
Network performance <10%

Transmit 10kb at
10Mbps = 1 msec.

Travel 600km in 3 msec

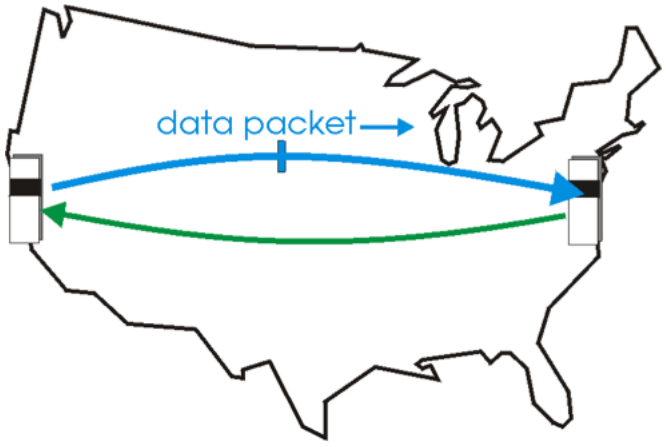
Auckland

Pipelining

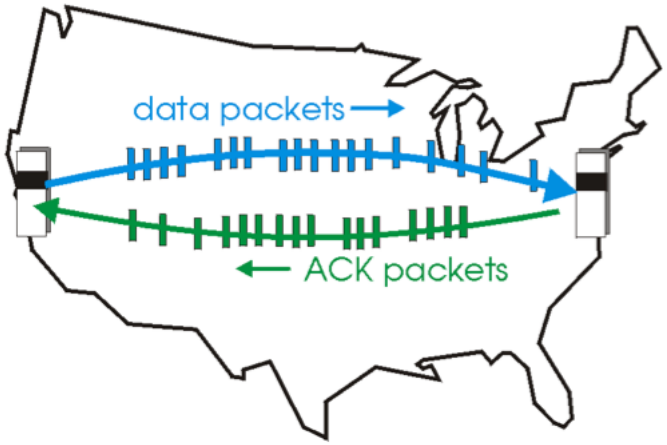

Wellington
Acknowldegement
arrives every 1 msec
Network
performance ≈100%
Packet arrives every 1 msec
Auckland

Pipelined Protocols

Pipelining: sender allows multiple, "in-flight", yet-to-be-acknowledged packets.



(a) a stop-and-wait protocol in operation

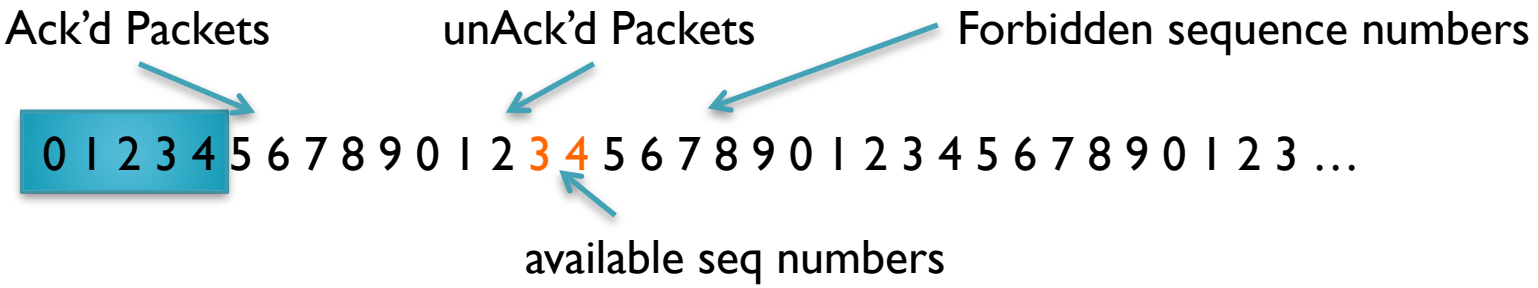


(b) a pipelined protocol in operation

- Two generic forms of pipelined protocols: *go-Back-N, selective repeat*

How do we manage all these inflight packets?

- We can use a software construct called a sliding window.
  - We have a fixed size window (i.e. n packets)
    - This means we can have up to n packets on the 'wire'.
    - When we successfully send a packet (i.e. its been ack'd by the receiver), we can move the window on by 1.
    - We need buffering at sender and/or receiver



Quick exercise

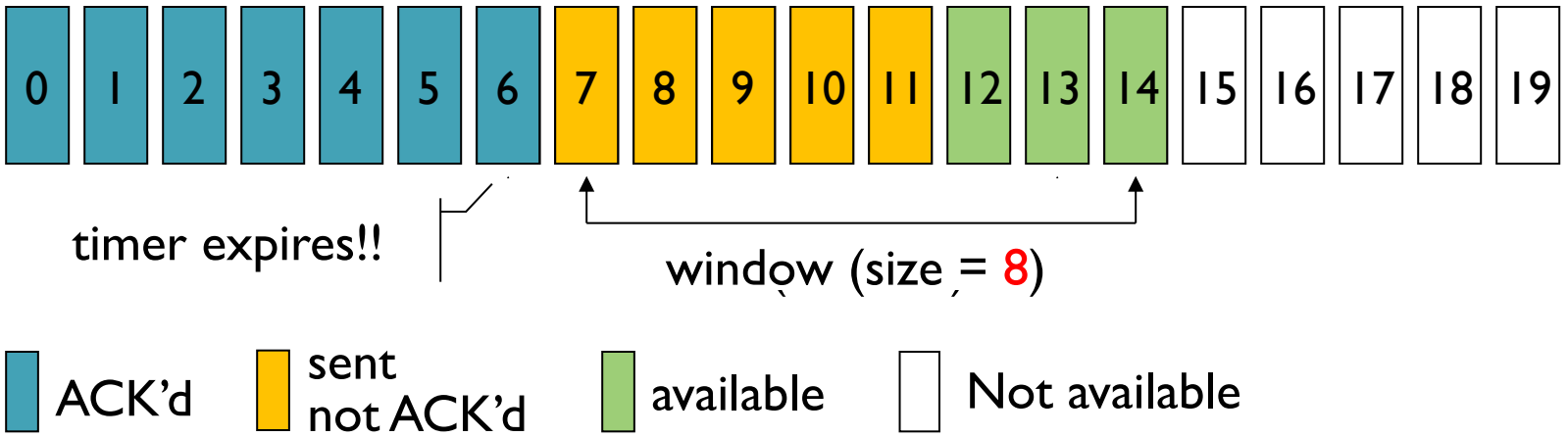
- When using the sliding-window protocol, will the utilization be improved because of the following?
  - A. Reduce propagation delay
  - B. Reduce sliding window size
  - C. Increase sliding window size
  - D. Reduce packet size

Protocol summary

- The important features of a sliding window protocol are:
  - Sequence number assignment
  - The resend protocol
  - Window management

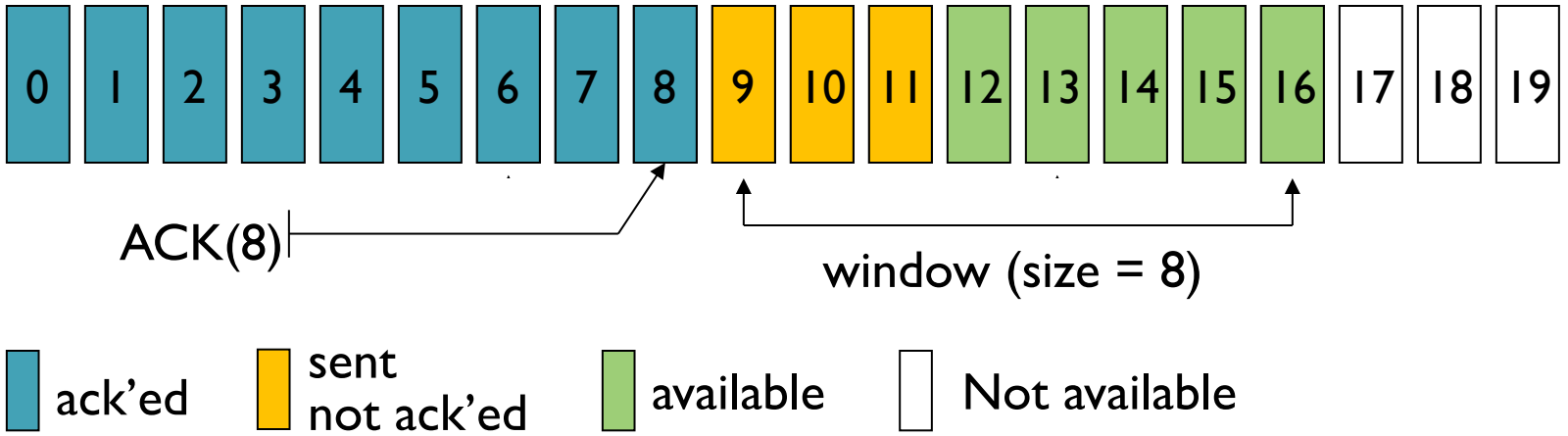
- We're going to look at GoBackN followed by Selective repeat.

Go Back N



- Very simple receiver, only accepts packets that arrive in-order and discards others.
- Send each packet in window in turn, window moves on when ***first*** packet in window is acknowledged.
- Timeout on first packet, then all *UNACKNOWLEDGED* packets resent

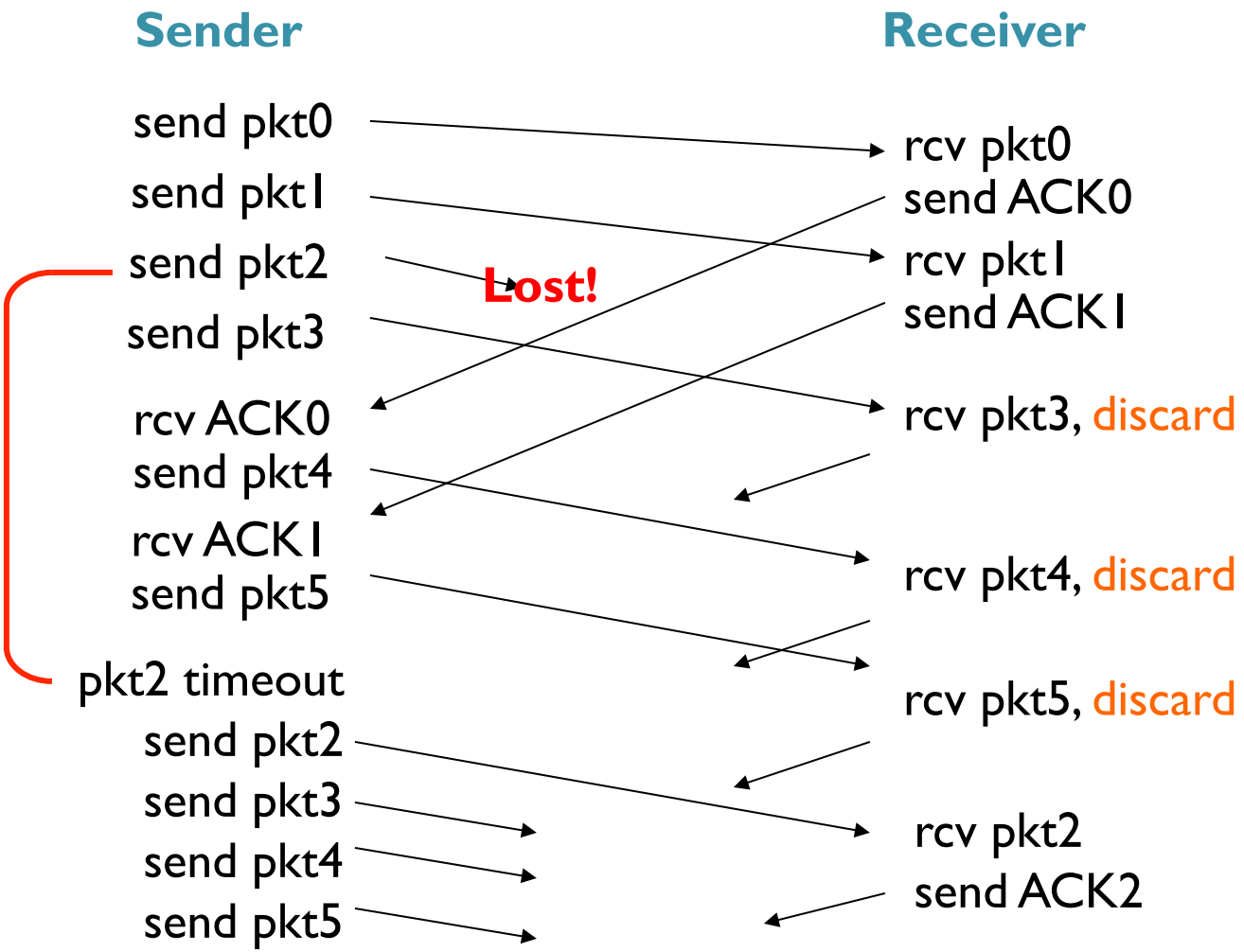
Go Back N (Cumulative ACK)



- Should a later packet be ACKed, consider ***all prior*** packets in the window to also be ACKed, so in other words:
  - ACK is cumulative, so an ACK for a later sequence number effectively ACKs all preceding packets.

- Lost ACK(6) and ACK(7)
- Get ACK(8)

Go Back N in action



Sender's sliding window size is 4

Quick exercise

- In a Go-Back-*N* protocol, if the window size is 63, what is the minimum range of sequence numbers?
  - A. 0-62
  - B. 0-63
  - C. 0-64
  - D. 1-63

An example

- sequence number range is **0-2** and the window size is **3**.

A -> 0 -> B

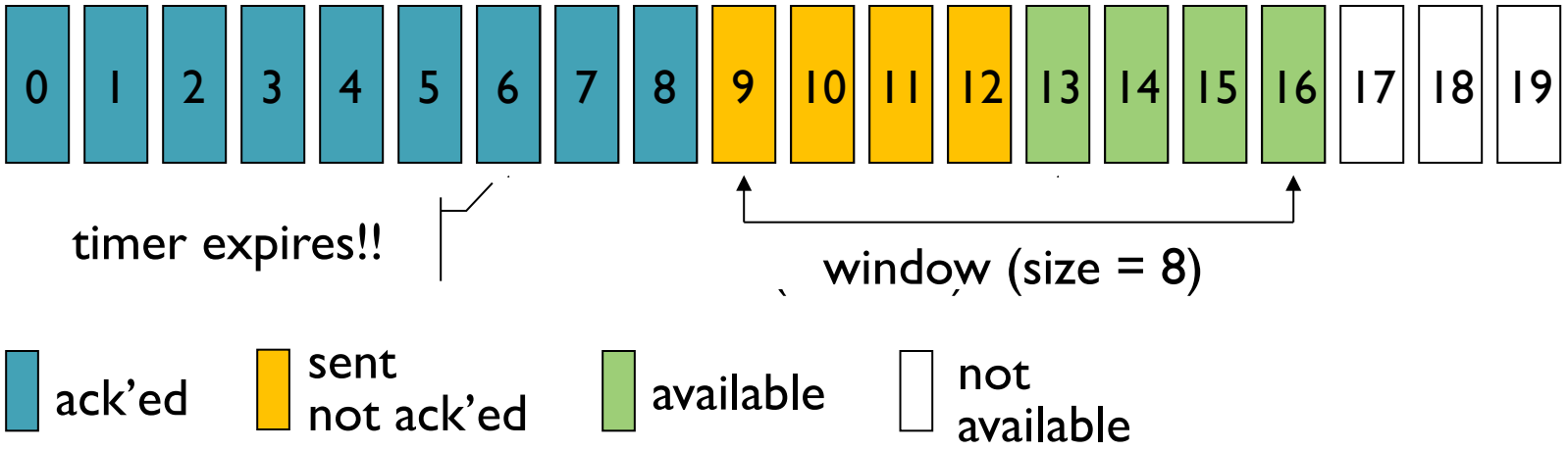
A -> 1 -> B

A -> 2 -> B

(ACKs from B to A all lost)

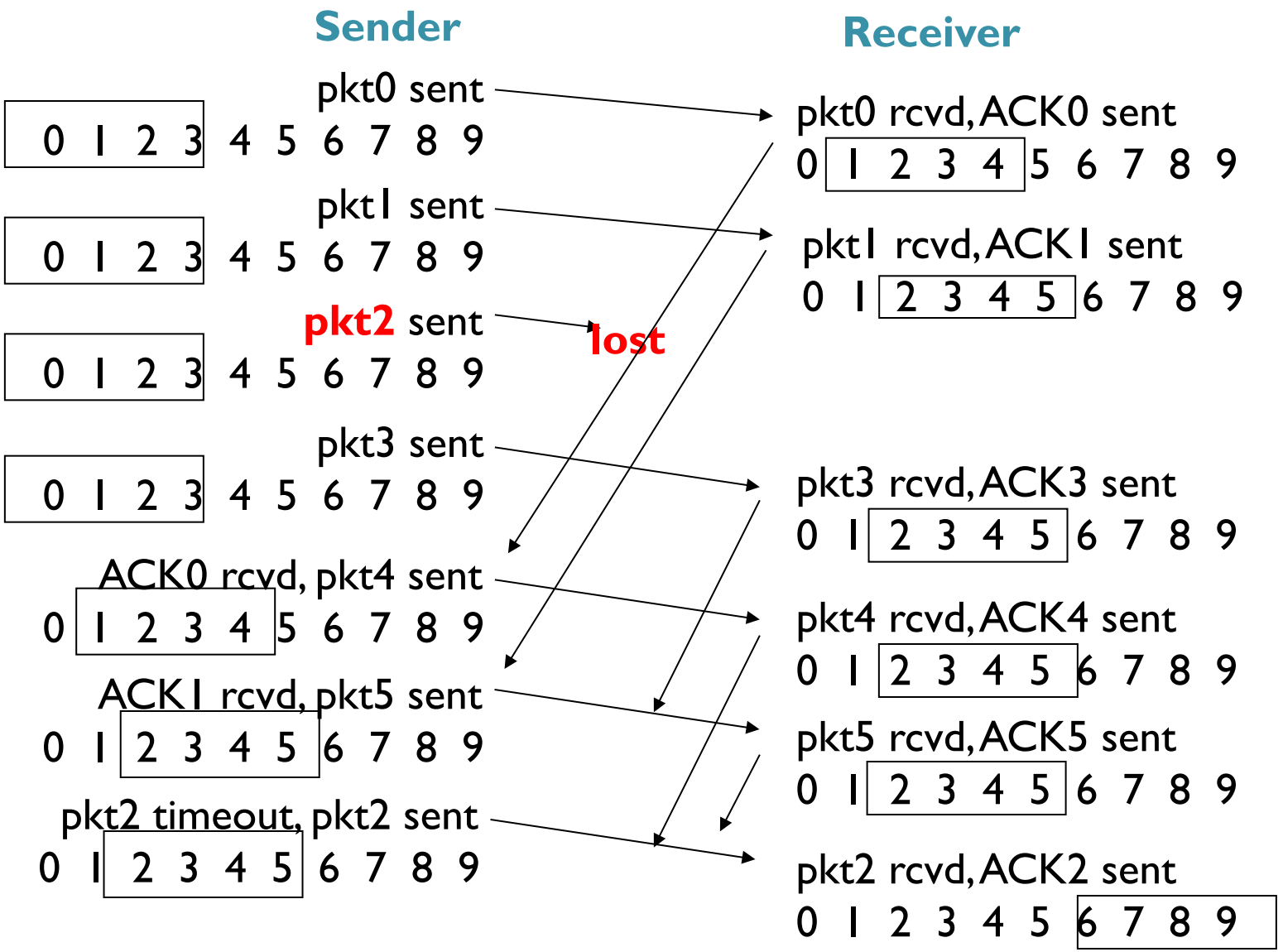
B is expecting to receive a new packet 0

Selective Repeat



- Can send fewer packets at the cost of making the protocol more complicated.
- Packets are individually acknowledged.
- Only one packet resent on timer expiry.

Selective Repeat in Action



Quick exercise

- In Selective Repeat, if 5 is the number of bits for the sequence number, then the maximum size of the sliding window must be _____

  - A. 15
  - B. 16
  - C. 31
  - D. 1

An example

- sequence number range is **0-3** and the window size is **3**.

A -> 0 -> B

A -> 1 -> B

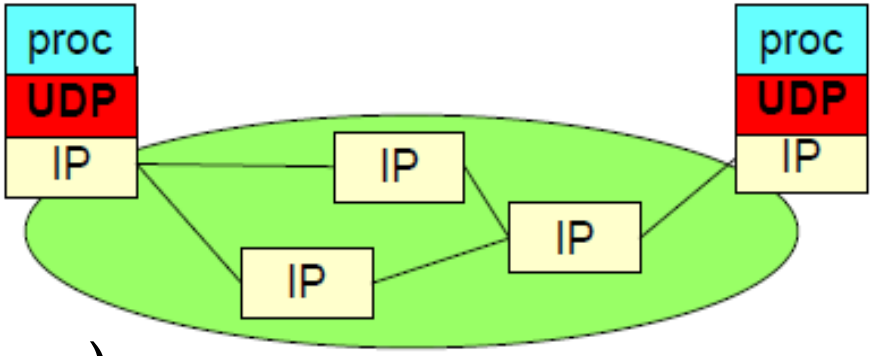
A -> 2 -> B

B's sliding window={3, 0, 1}

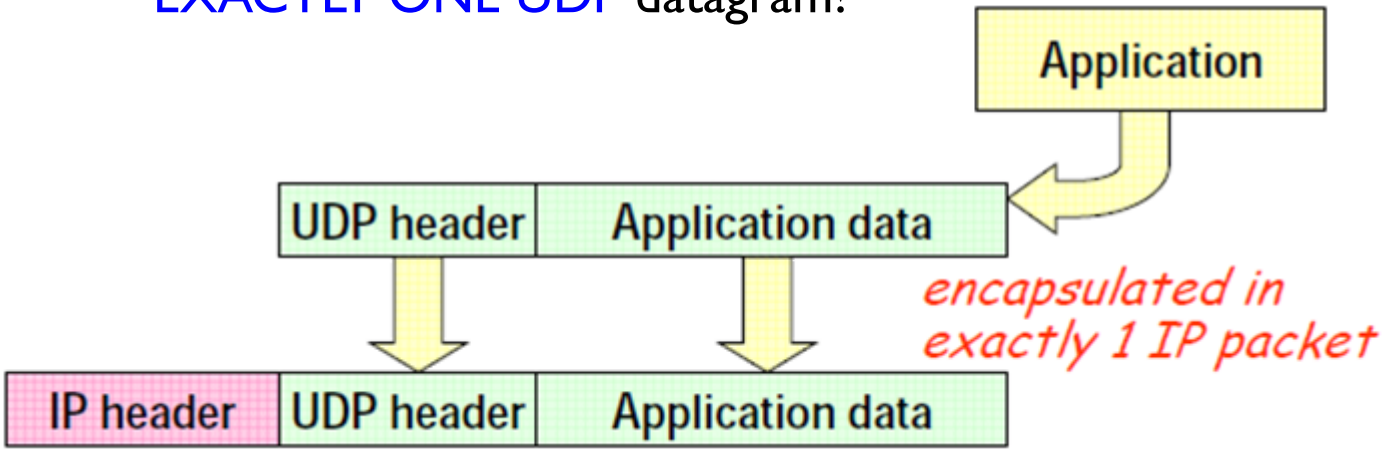
(ACKs from B to A all lost)

User datagram protocol

- Connection-Less
  - (no handshaking)



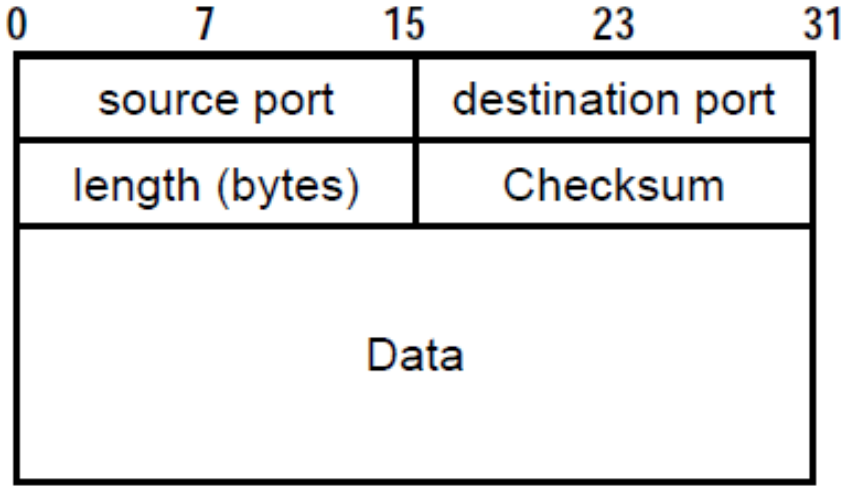
- UDP packets (Datagrams)
  - Each application interacts with UDP transport sw to produce EXACTLY ONE UDP datagram!



This is why, improperly, we use the term UDP packets

UDP datagram format

8 bytes header + variable payload

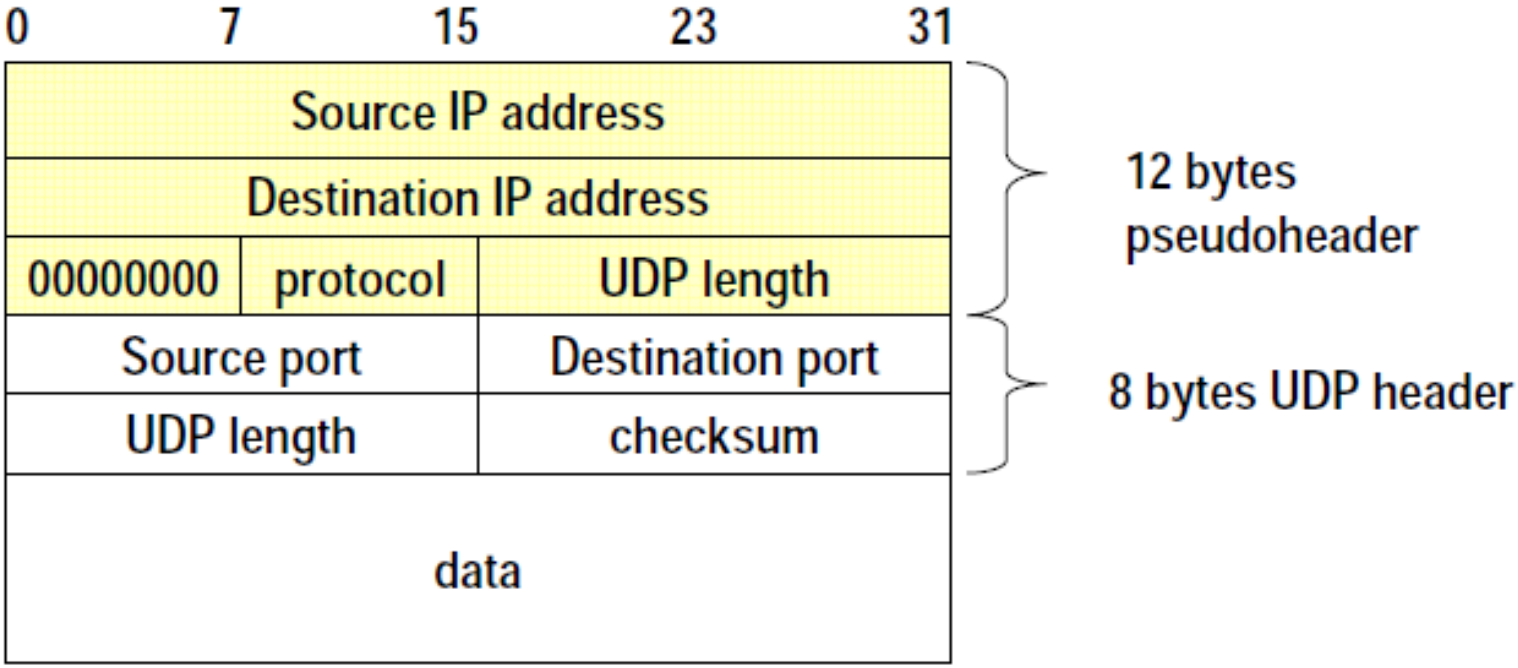


- UDP length field
  - all UDP datagram
  - (header + payload)
- payload sizes allowed:
  - Empty (0)
  - 65527 bytes (65535-8)

- UDP functions limited to:
  - Error checking
    - which may even be **disabled for performance**
  - addressing
    - which is the only strictly necessary role of a transport protocol

IPv4 pseudo header

- The pseudo header is only used for the checksum computation



Quick exercise

- Which of the following is not true about disabling checksum?
  - A. Checksum in UDP is not always necessary since IP packet checksum includes packet payload.
  - B. UDP checksum is often disabled to speed up implementation.
  - C. Lack of UDP checksum is tolerable for communication in LANs.
  - D. Lack of UDP checksum is definitely dangerous in the Internet.

UDP: a lightweight protocol

- No connection establishment
  - no initial overhead due to handshaking
- No connection state
  - greater number of supported connections by a server!
- Small packet header overhead
  - 8 bytes only vs 20 in TCP

- originally intended for simple applications, oriented to short information exchange
  - DNS
  - management (e.g. SNMP)
  - Distributed file system support (e.g. NFS)
  - etc

Unregulated send rate in UDP

- No rate limitations
- No re-transmission
- Extremely important features for today multimedia applications
- *Be careful: UDP ok for multimedia because it does not provide anything at all (no features = no limits!).*

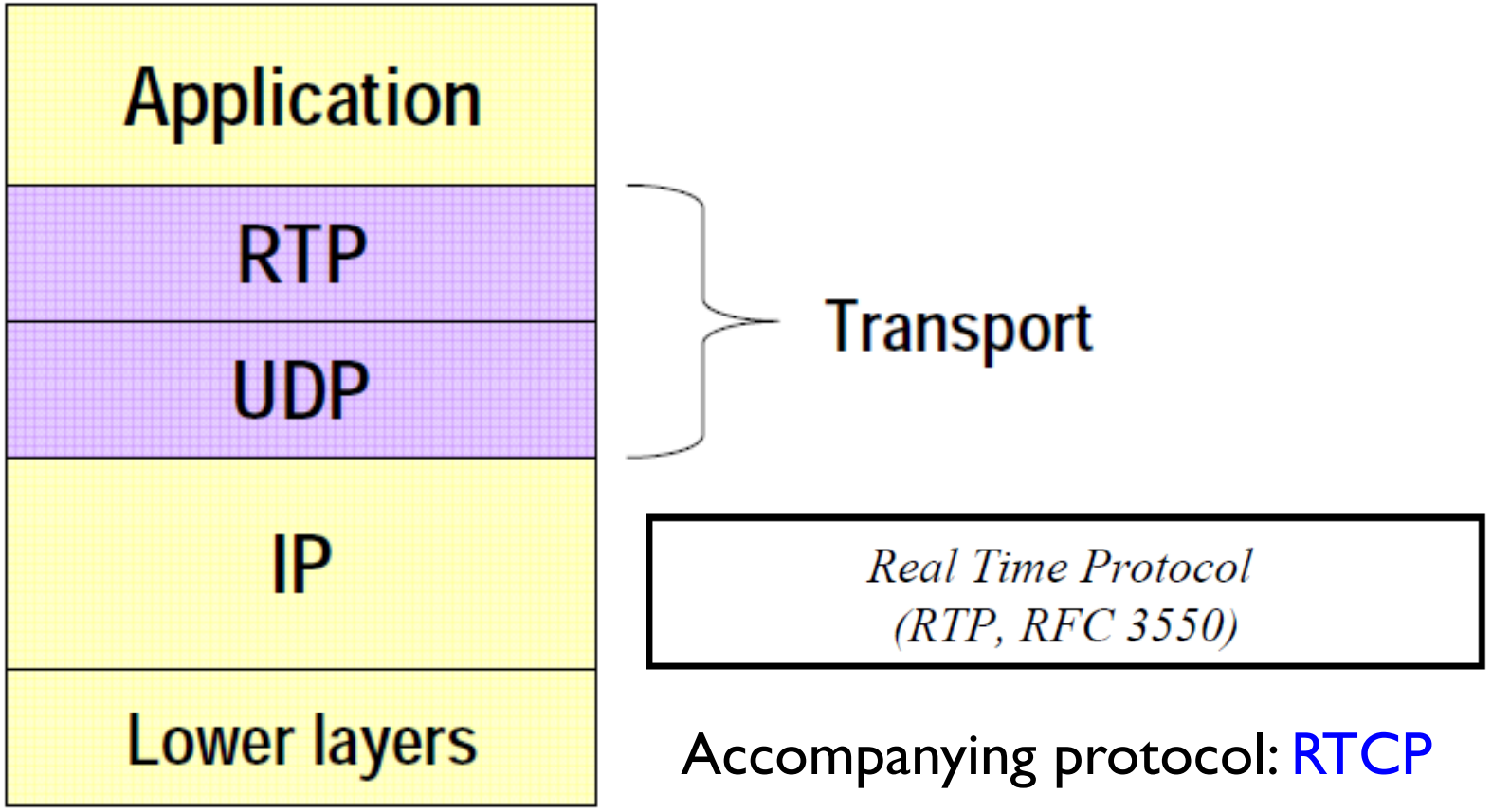
Quick exercise

- Which of the following about UDP is correct?
  - A. UDP supports a self-regulating "throttle" feature that prevents network saturation
  - B. UDP consumes fewer computer resources by not maintaining connection state
  - C. UDP guarantees that individual packets of a transmission will arrive "in order"
  - D. None of the above

Question to ponder

- Why would a TCP and UDP "phone call" likely have equivalent performance characteristics in practice?

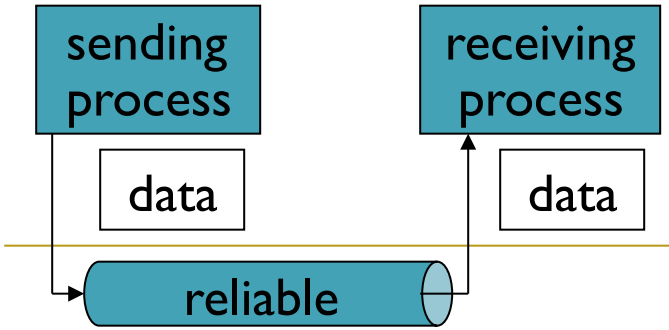
RTP: sublayer of transport



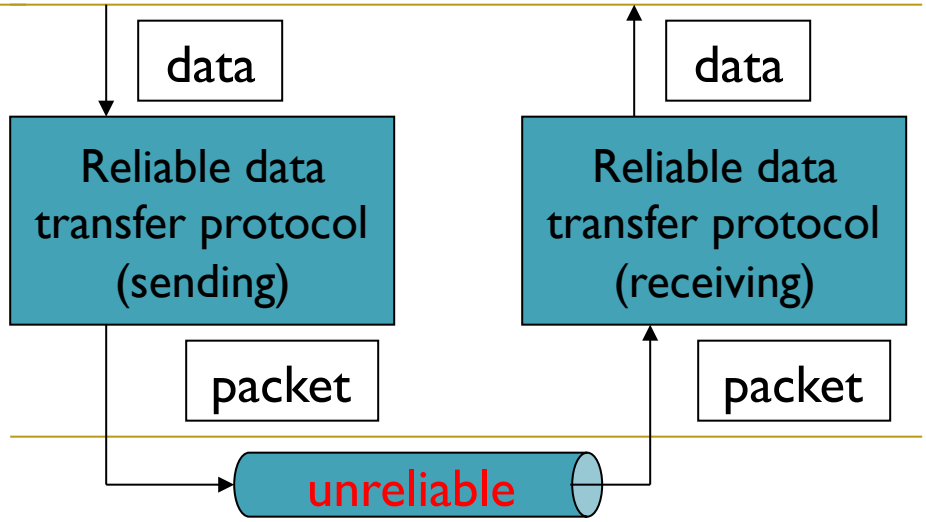
Accompanying protocol: RTCP

Connection Oriented Transport

- Process to process
- Sequenced, error free
- Flow and congestion control

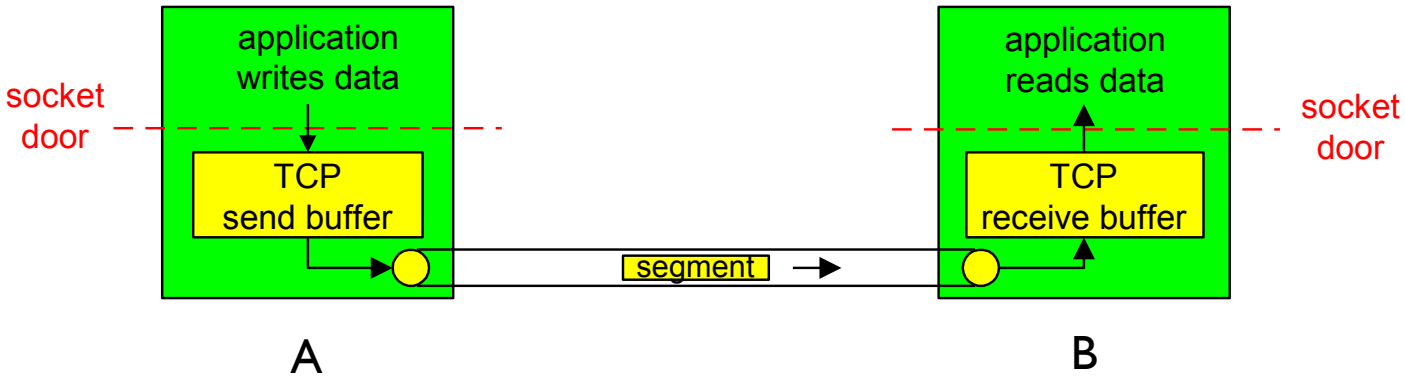


- Bit errors
- Losses
- Delay
- Reordering
- Fragmentation



TCP

- point-to-point:
  - one sender, one receiver
  - no multicast
- reliable, in-order *byte steam:*
  - no "**message boundaries**"
- full duplex data:
  - bi-directional data flow in same connection
- connection-oriented:
  - handshaking (exchange of control msgs) init's sender, receiver state before data exchange



Quick exercise

- TCP does not support virtual circuit, why?
  - A. TCP relies on IP and IP does not support virtual circuit
  - B. TCP maintains connection information at two communicating end systems only
  - C. Virtual circuit can only be established at the network layer
  - D. None of the above

The TCP Header

- Byte stream sent as sequence of segments
  - Segment may be 0 to 64k bytes

Offsets	Octet	0								1								2								3							
Octet	Bit	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	0	Source port																Destination port															
4	32	Sequence number																															
8	64	Acknowledgment number (if ACK set)																															
12	96	Data offset				Reserved 0 0 0				NS	CWR	ECE	URG	ACK	PSH	RST	SYN	FIN	Window Size														
16	128	Checksum																Urgent pointer (if URG set)															
20 ...	160 ...	Options (if Data Offset > 5, padded at the end with "0" bytes if necessary) ...																															

TCP seq numbers and ACKs

Seq. #'s:

- byte stream
- number of first byte in segment's data

ACKs:

- seq # of *next* byte expected from other side
- cumulative ACK

Host A

Host B

User types 'C'

Seq=42, ACK=79, data = 'C'

host ACKs receipt of 'C', echoes back 'C'

Seq=79, ACK=43, data = 'C'

host ACKs receipt of echoed 'C'

Seq=43, ACK=80, data= 'D'

time

PiggyBacking

- There is one important way to increase efficiency.
  - Assume that any connection is duplex,
  - with data flowing evenly back and forth.
- Now, we don't need to send a separate acknowledgment, just attach to a returning data segment.

Quick exercise

- Empty ACK segments consume _____ sequence number(s) and are not acknowledged.
  - A. no
  - B. one
  - C. two
  - D. three

44

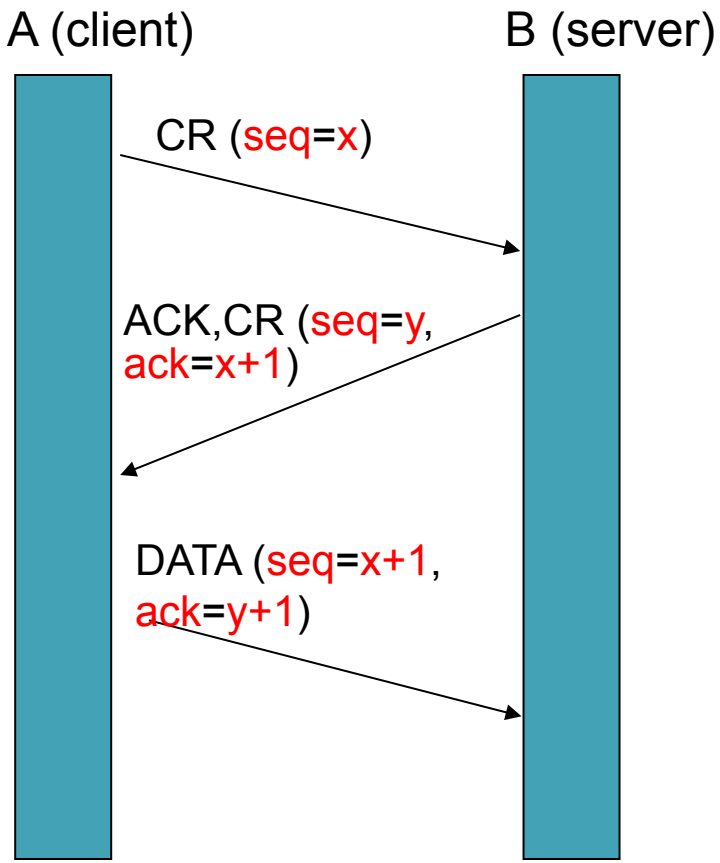
TCP Connection Establishment

TCP uses a **three-way handshake** to open a connection:

1. **ACTIVE OPEN:** Client sends a segment with
   - SYN bit set
   - port number of **client and server**
   - initial sequence number (X) of client
2. **PASSIVE OPEN:** Server responds with a segment with
   - SYN bit set
   - initial sequence number (Y) of **server**
   - ACK for ISN of client (X+1)
3. **Client acknowledges by sending a segment with:**
   - ACK ISN of server (Y+1)

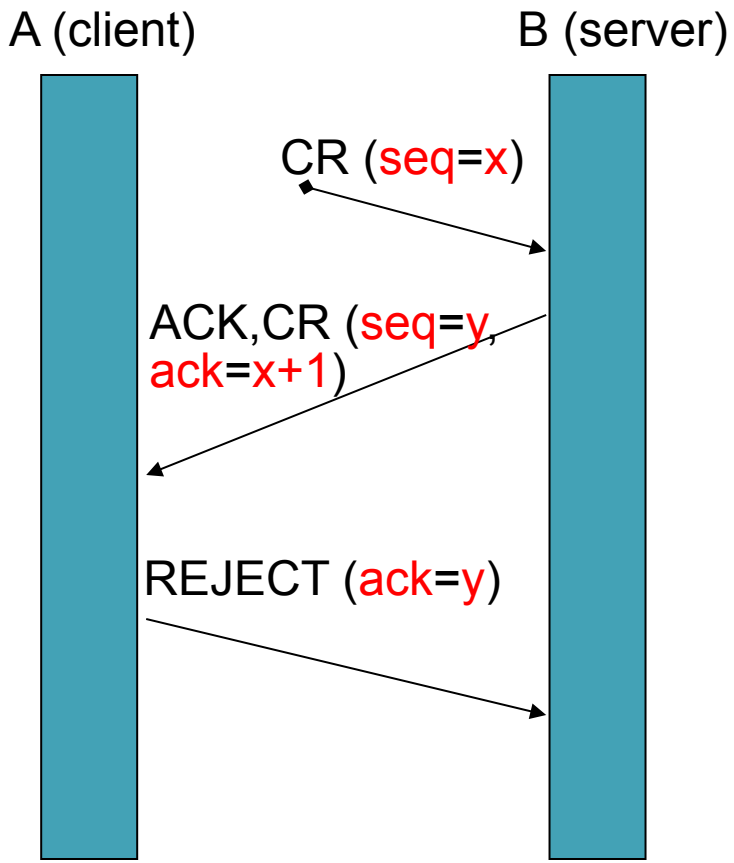
Establishing a Connection

- Normal connection
- A initiates and chooses its initial sequence number.
- B replies with its own initial sequence number and acknowledges host 1.
- A begins data transmission.



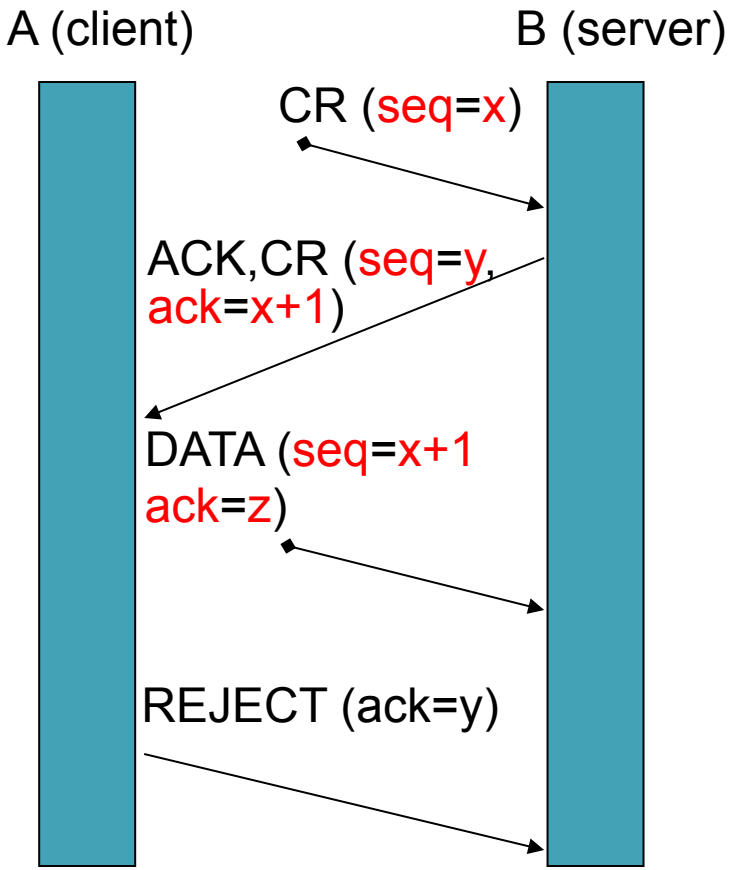
Duplicate Request

- A previous connection request appears from nowhere.
- B replies with its own initial sequence number and acknowledges host 1.
- A rejects the connection request.



Duplicate Connection

- A previous connection request appears from nowhere.
- B replies.
- The first data packet of the previous connection also appears.
- A rejects the connection request.



Quick exercise

- The connection establishment procedure in TCP is susceptible to a serious security problem called the _____ attack.
  - A. ACK flooding
  - B. FIN flooding
  - C. SYN flooding
  - D. None of the above

TCP Connection Termination

- Each end of the data flow must be shut down independently (**"half-close"**)

- Four steps involved:
  - (1) A sends a FIN to B **(active close)**
  - (2) B ACKs the FIN,

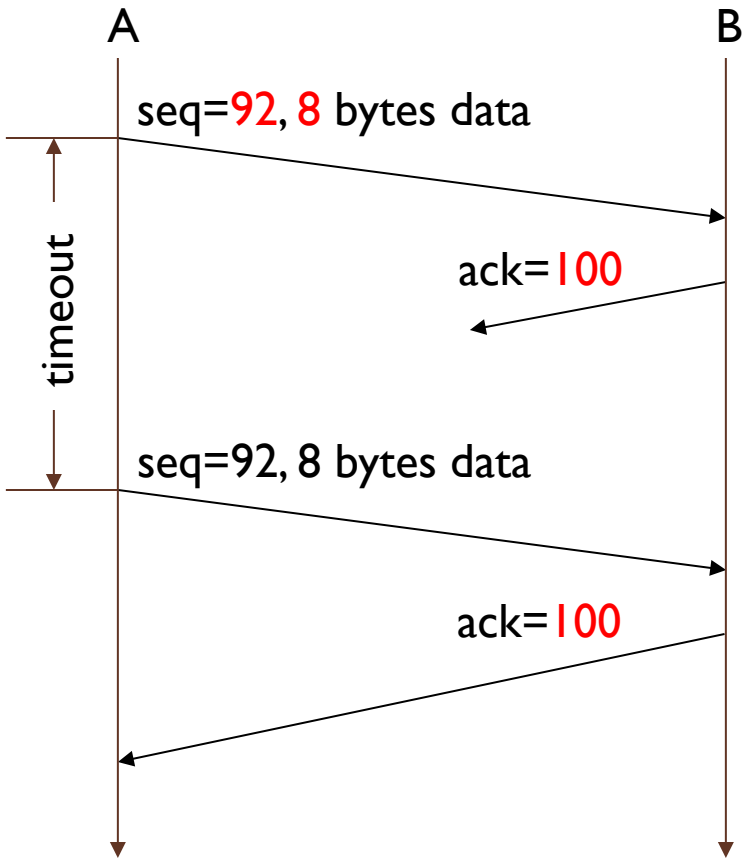
    (at this time: B can still send data to A)
  - (3) and B sends a FIN to A **(passive close)**
  - (4) A ACKs the FIN.

Quick exercise

- Which process can be used to terminate a TCP connection?
  - A. three-way handshake
  - B. four-way handshake
  - C. two-termination sequence
  - D. None of the above

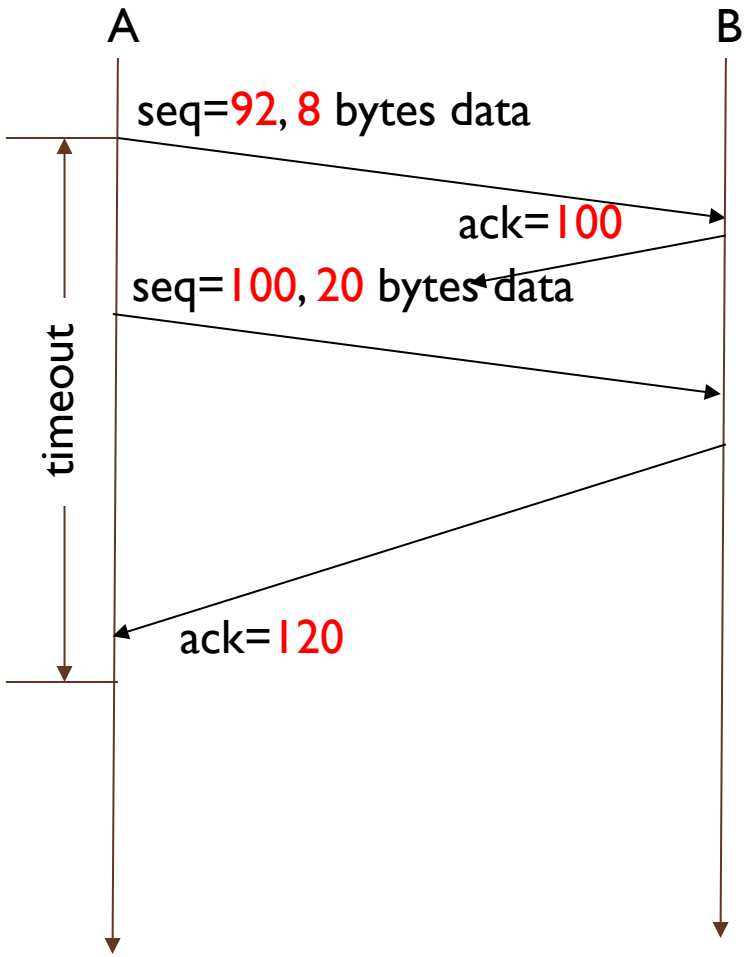
TCP Sequencing

- Retransmission due to lost acknowledgment.



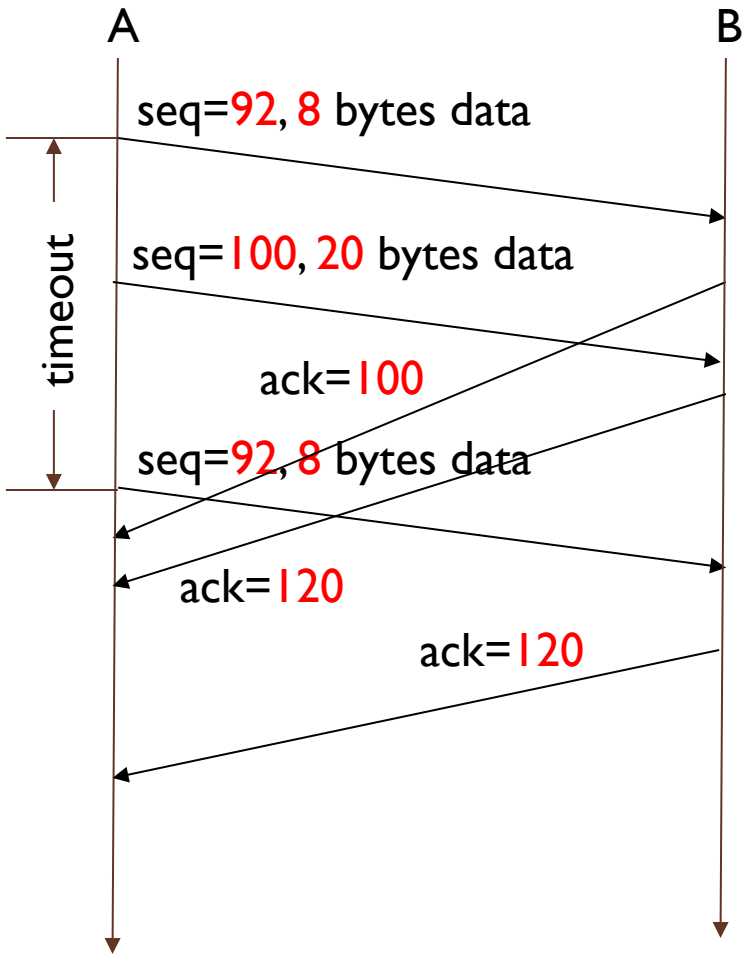
TCP Sequencing cont.

- Cumulative acknowledgment avoids retransmission.



TCP Sequencing cont.

- Delayed ACK causes retransmission



Fast Retransmit

- Time-out period often relatively long:
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs.
  - Sender often sends many segments back-to-back
  - If segment is lost, there will likely be many duplicate ACKs.

- If sender receives 3 ACKs for the same data, it supposes that segment after ACKed data was lost:
  - fast retransmit: resend segment before timer expires

Fast retransmit algorithm:

```
event: ACK received, with ACK field value of y
        if (y > SendBase) {
            SendBase = y
            if (there are currently not-yet-acknowledged segments)
                start timer
        }
        else {
            increment count of dup ACKs received for y
            if (count of dup ACKs received for y = 3) {
                resend segment with sequence number y
            }
```

a duplicate ACK for already ACKed segment

fast retransmit

TCP ACK Generation @ Receiver

Event	TCP Receiver action
in-order segment arrival, no gaps, everything else already ACKed	delayed ACK. Wait up to 500ms for next segment. If no next segment, send ACK
in-order segment arrival, no gaps, one delayed ACK pending	immediately send single cumulative ACK
out-of-order segment arrival higher-than-expect seq. # gap detected	immediately send duplicate ACK, indicating seq. # of next expected byte
arrival of segment that partially or completely fills gap	immediate ACK if segment starts at lower end of gap