COSC264

Introduction to Computer Networks and the Internet

Transport Layer Protocols: TCP Reliable Data Transfer

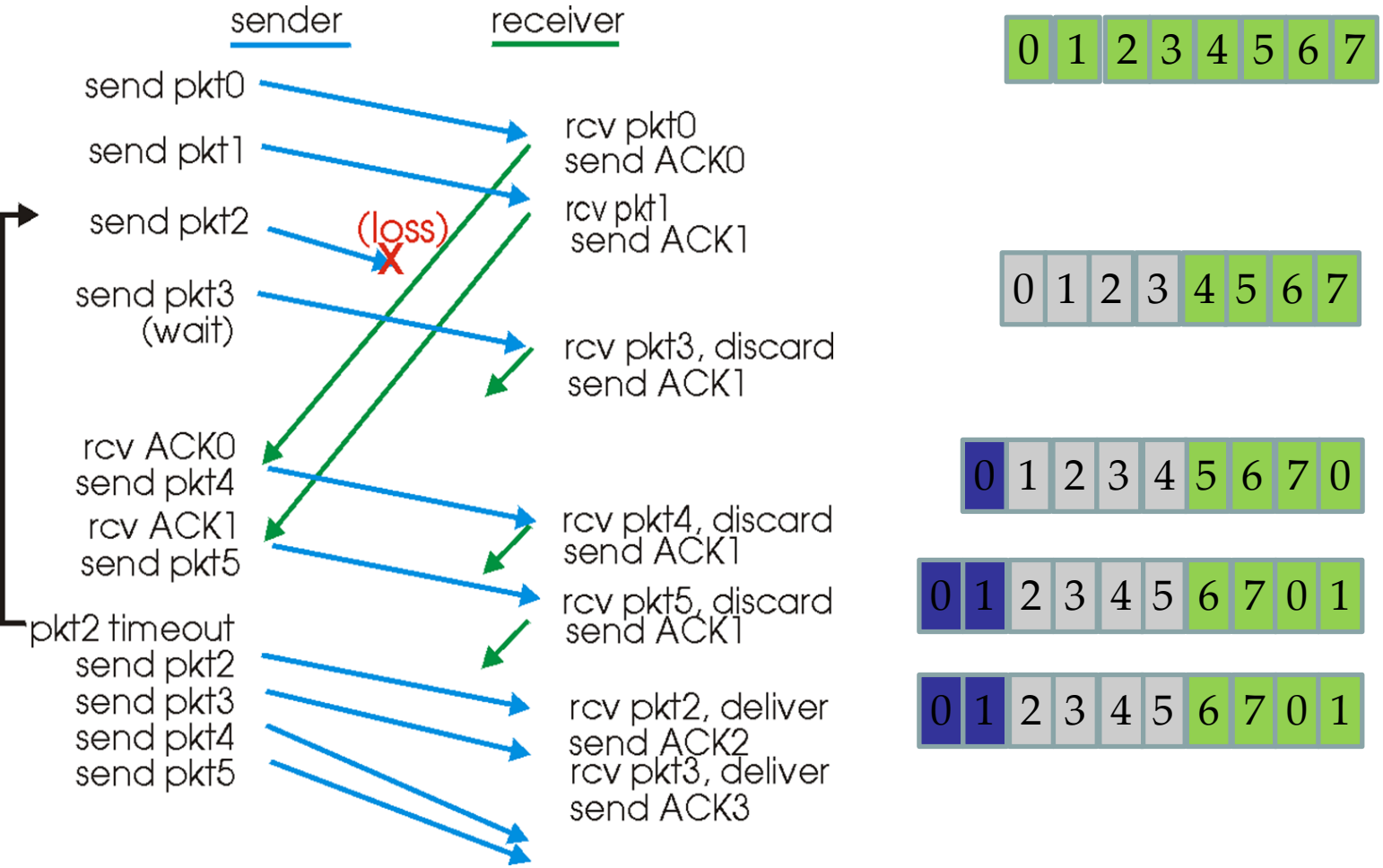
Dr. Barry Wu

Wireless Research Centre

University of Canterbury

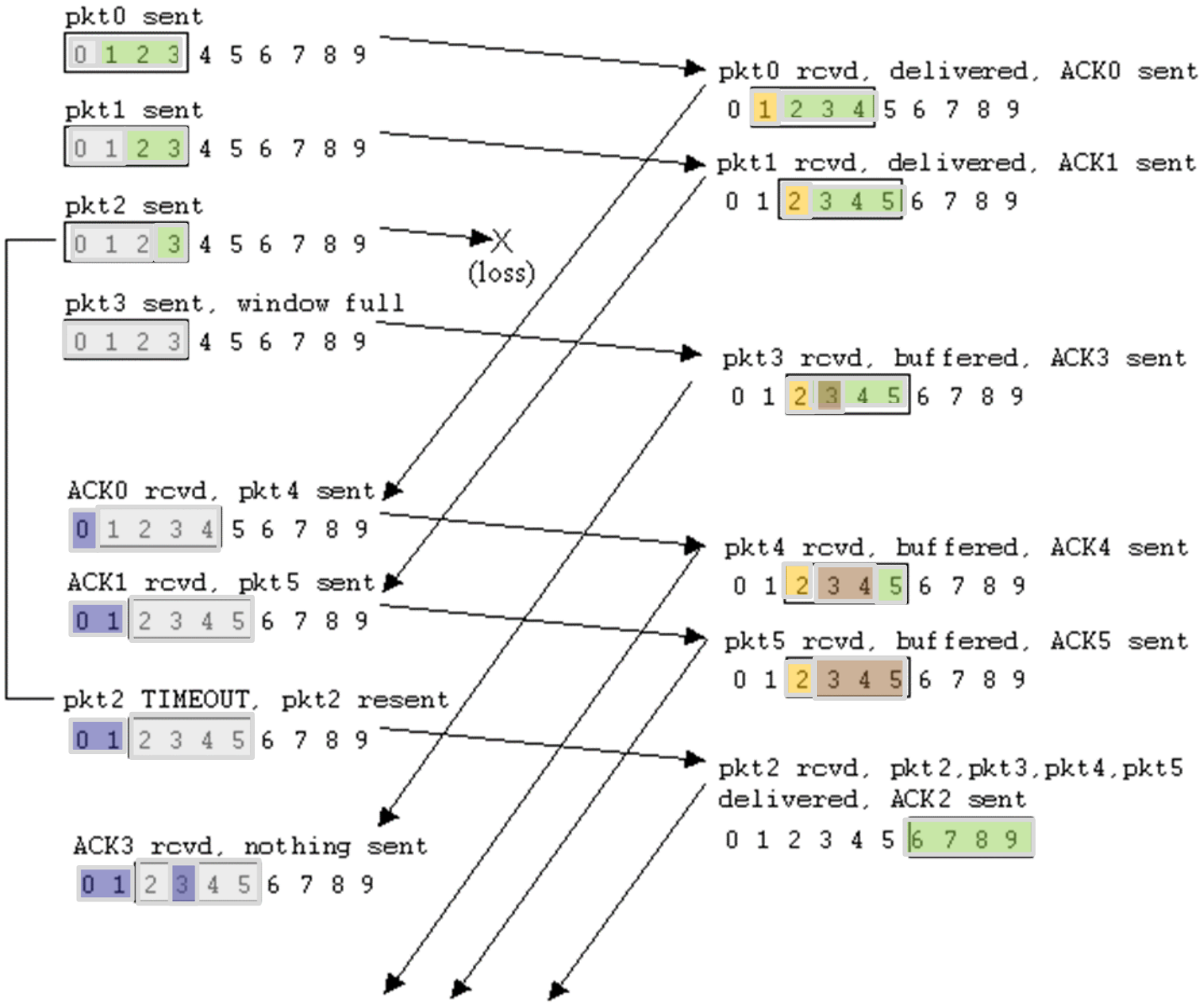
barry.wu@canterbury.ac.nz

Go-Back-N: an example



[AK05, KR3]

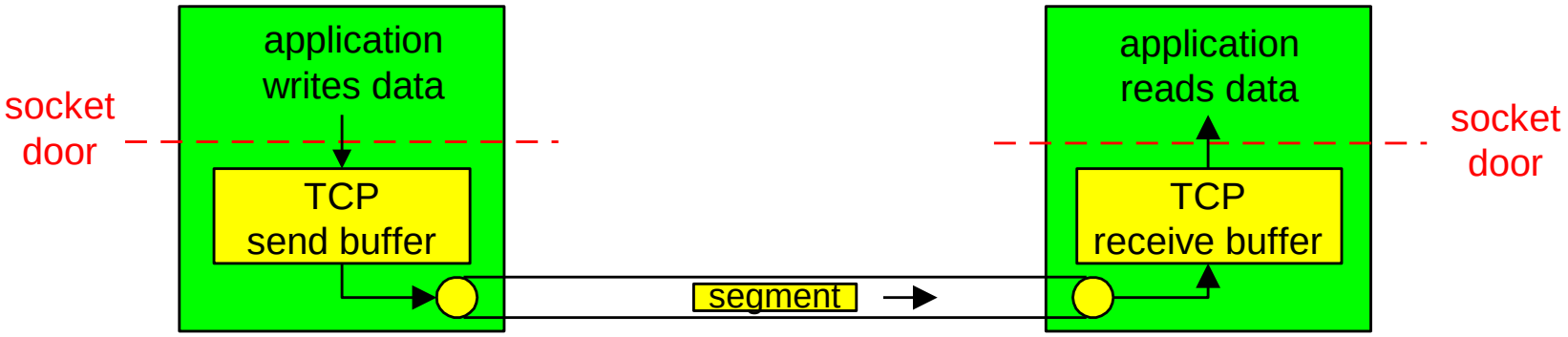
SR: an example (after-receipt window state)



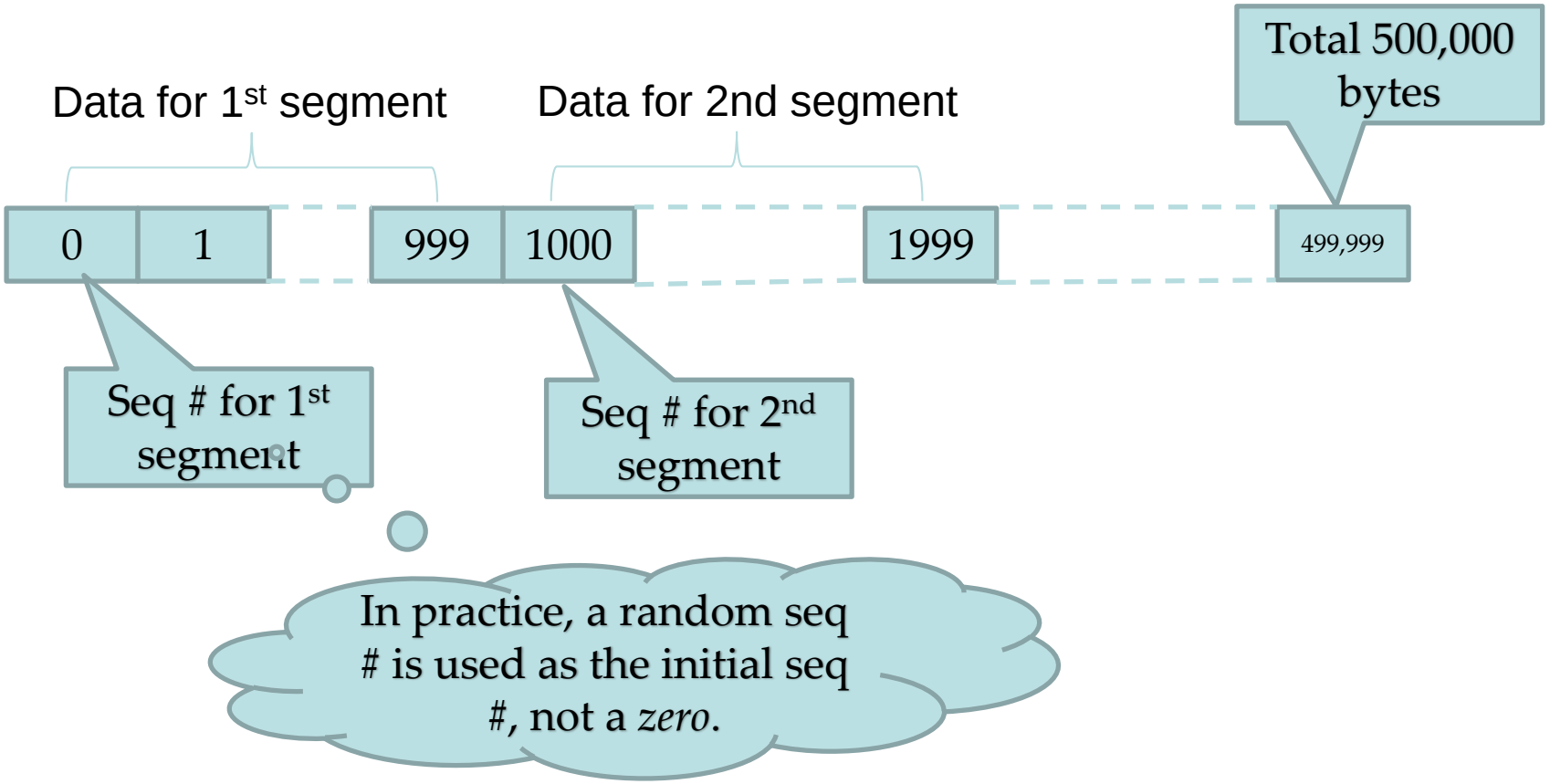
Outline

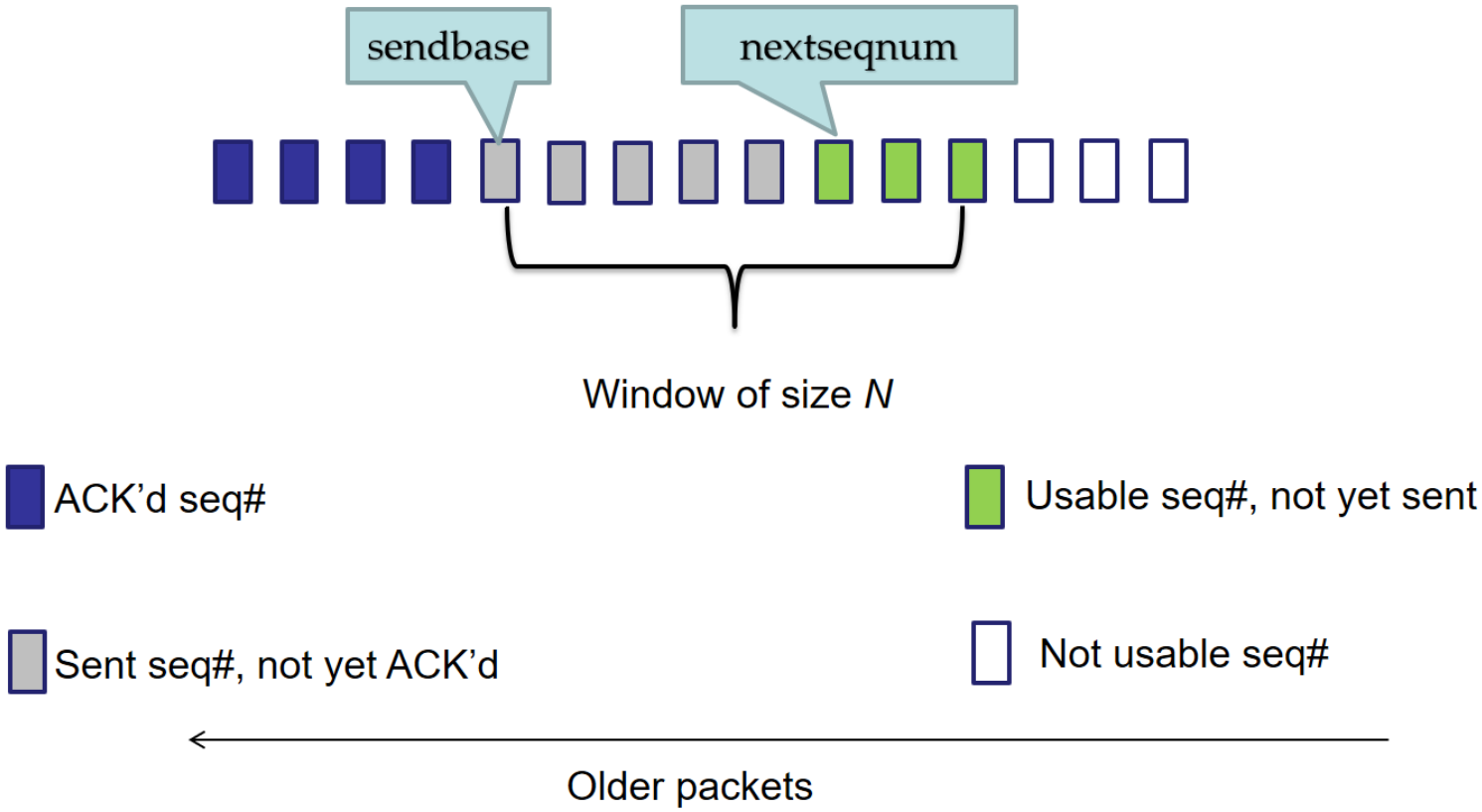
- TCP Reliable data transfer

TCP basics

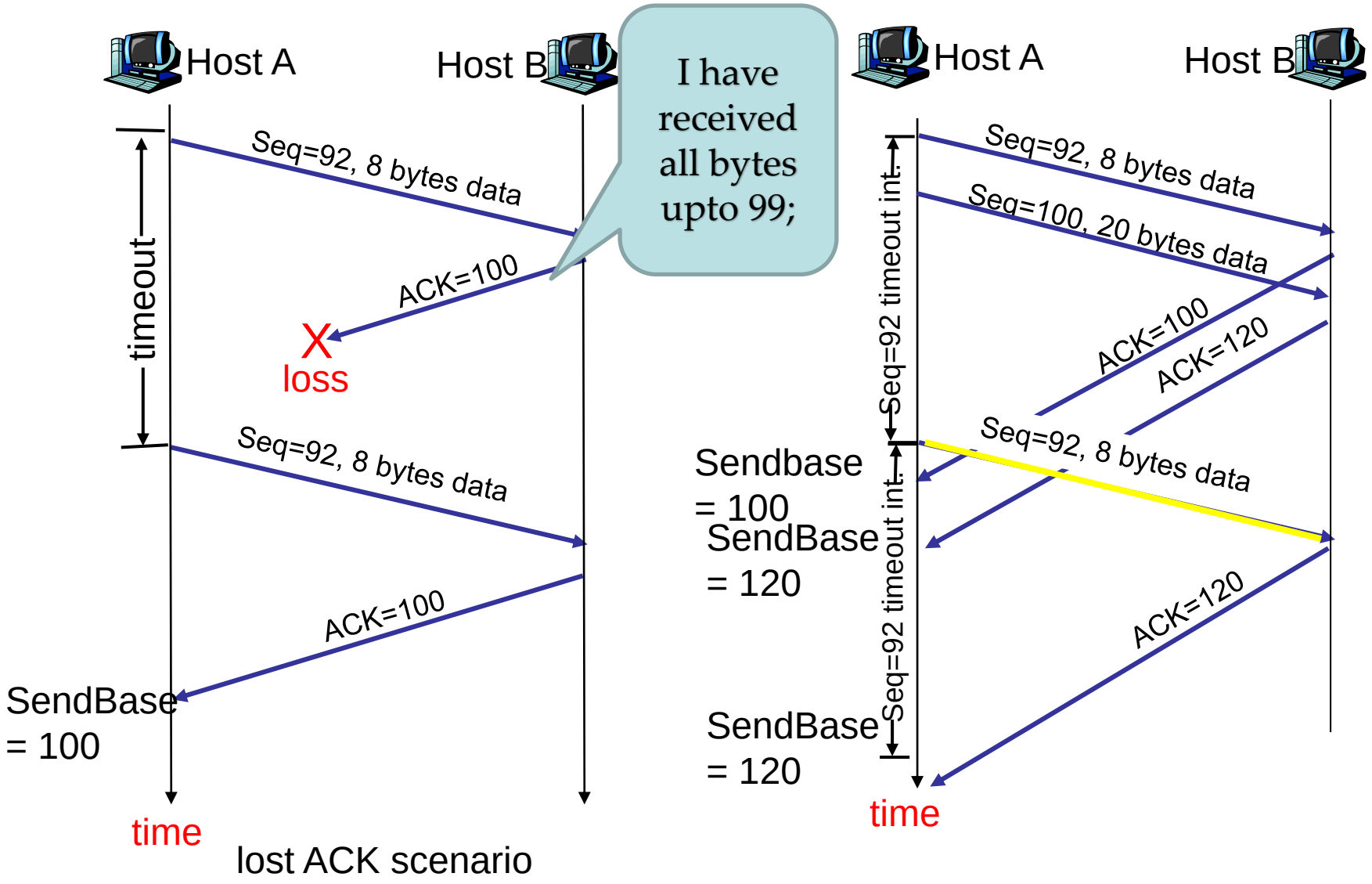

application
writes data
socket
door
TCP
send buffer
segment
application
reads data
socket
door
TCP
receive buffer

TCP segments

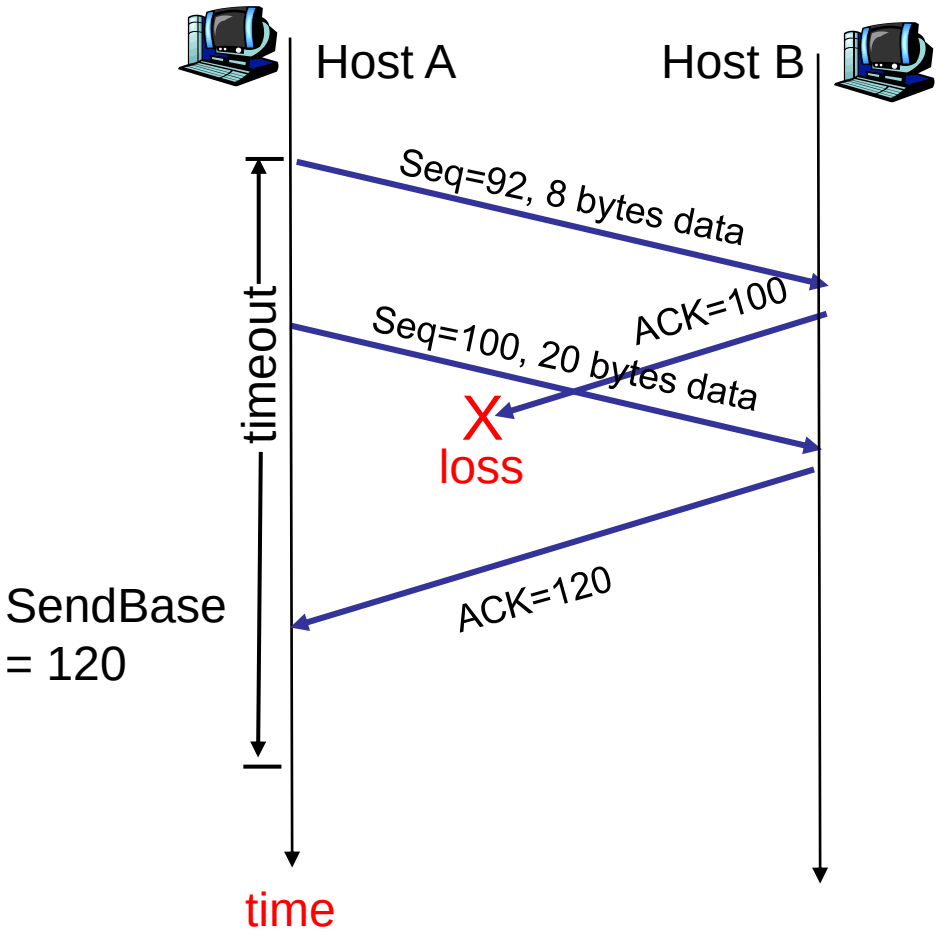



sendbase
nextseqnum
Window of size N
ACK'd seq#
Usable seq#, not yet sent
Sent seq#, not yet ACK'd
Not usable seq#
Older packets

TCP: retransmission scenarios



TCP retransmission scenarios (more)



Cumulative ACK scenario

TCP sender events:

data rcvd from app:

- Create segment with seq #
- seq # is byte-stream number of first data byte in segment
- start timer if not already running (think of timer as for oldest unacked segment)

TCP sender events:

timeout:

- retransmit segment with the *smallest* seq#;
- restart timer

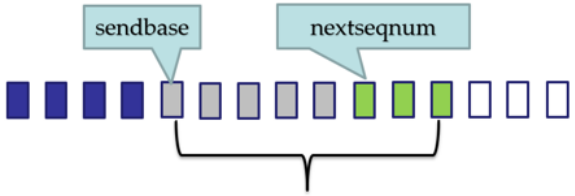
Ack rcvd:

- Cumulative ACK;
- Slide window;
- … (later);

NextSeqNum = InitialSeqNum
SendBase = InitialSeqNum

TCP sender (simplified)

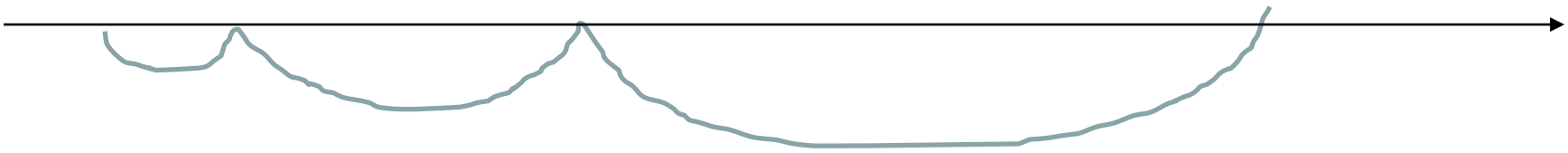
```
loop (forever) {
   switch(event)
   event: data received from application above
        create TCP segment with sequence number NextSeqNum
        if (timer currently not running)
             start timer
        pass segment to IP
        NextSeqNum = NextSeqNum + length(data)

    event: timer timeout
        retransmit not-yet-acknowledged segment with
              smallest sequence number
        start timer

    event: ACK received, with ACK field value of y
        if (y > SendBase) {
             SendBase = y
            if (there are currently not-yet-acknowledged segments)
                   start timer
            }

} /* end of loop forever */
```

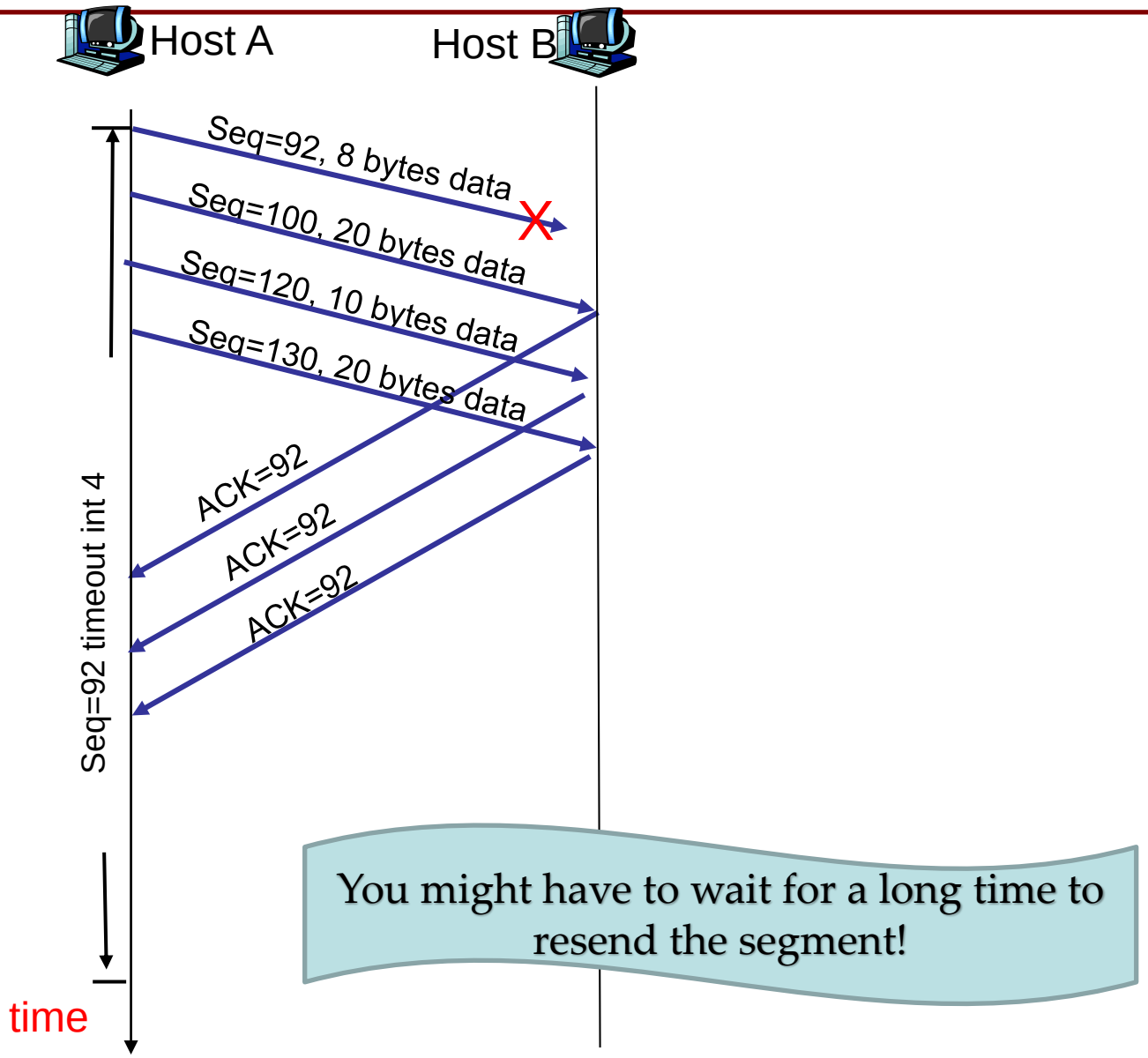


Some refinement

timeout:

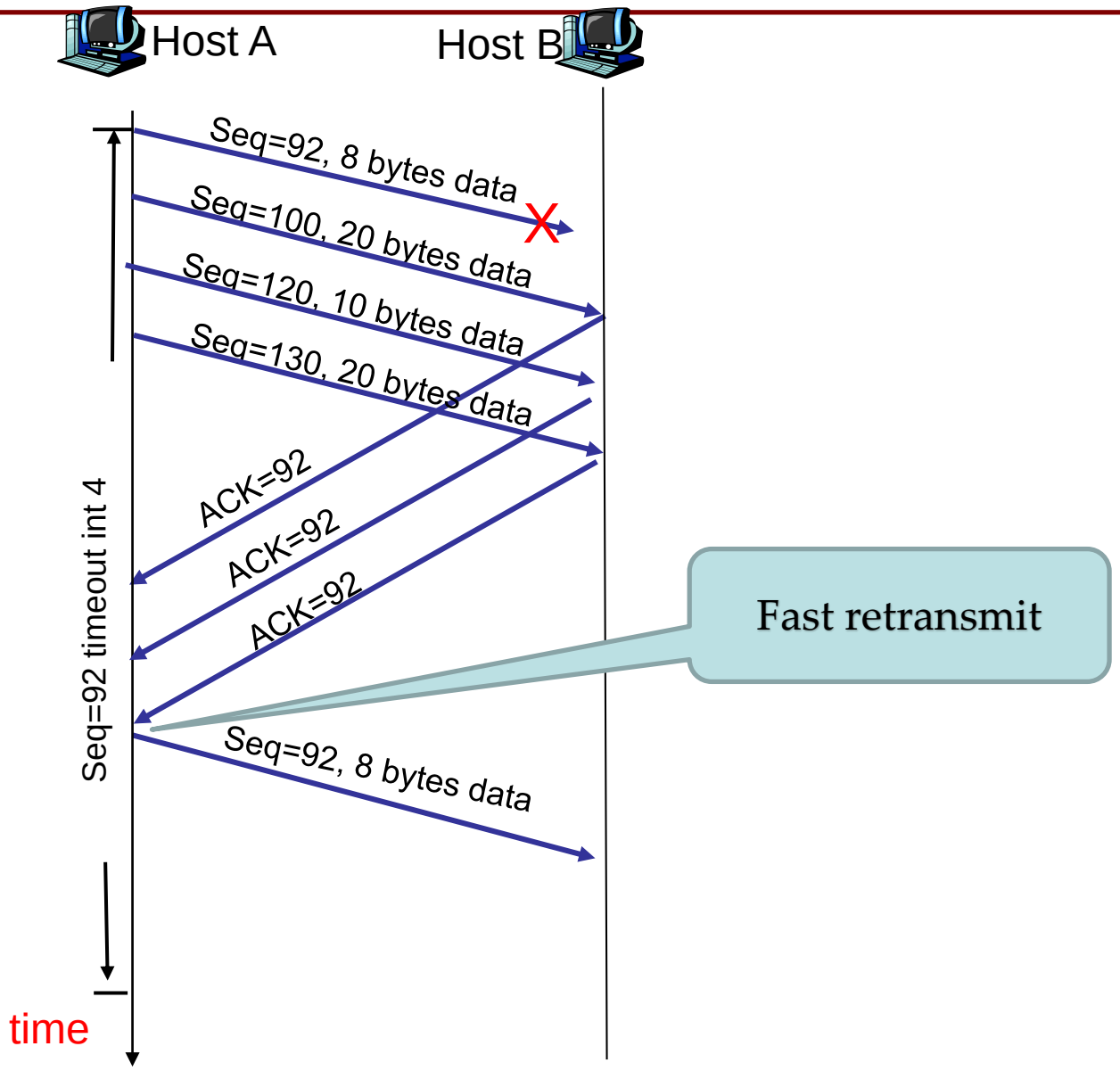
- When timeout expires after `TimoutInterval;`
  - Retransmits the unAck'd segment with smallest seq#; (*not all of the unAck'd pkts, different from Go-Back-N*)
  - (new) `TimoutInterval` =2* (previous) `TimeoutInterval`
- Motivation: no packet was received for the last timeout period
  - Could be a congestion*… trying to slow down the sending process*;
- `TimoutInterval` uses *initial value* when event "call from above" or "receipt of ACK" is activated;




Host A
Host B
Seq=92, 8 bytes data
Seq=100, 20 bytes data
Seq=120, 10 bytes data
Seq=130, 20 bytes data
ACK=92
ACK=92
ACK=92
Seq=92 timeout int 4
time
You might have to wait for a long time to resend the segment!

Fast Retransmit

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


Host A
Host B
Seq=92, 8 bytes data
Seq=100, 20 bytes data
Seq=120, 10 bytes data
Seq=130, 20 bytes data
ACK=92
ACK=92
ACK=92
Seq=92 timeout int 4
Fast retransmit
Seq=92, 8 bytes data
time

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

After receiving 3 dup ACKs, the sender is very sure that the related pkt has been lost;

TCP ACK generation [RFC 1122, RFC 2581]

Event at Receiver	TCP Receiver action
Arrival of in-order segment with expected seq #. All data up to expected seq # already ACKed	**Delayed ACK**. Wait up to 500ms for next segment. If no next segment, send ACK
Arrival of in-order segment with expected seq #. One other segment has ACK pending	**Immediately** send single cumulative ACK, ACKing both in-order segments
Arrival of out-of-order segment 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	**Immediately** send ACK, provided that segment starts at lower end of gap

GBN or SR?

TCP rdt

- Cumulative ACK (similar to GBN)
- Many TCP implementation buffers out-of-order segments (similar to SR);
- When there is timeout; sender retransmits the unAck'ed segments with smallest seq# only! (no "go-back-n")
- A mixture;

TCP reliable data transfer: a summary

- TCP creates rdt service on top of IP's unreliable service
- Pipelined segments
- Cumulative ACKs
- TCP uses single retransmission timer
- Retransmissions are triggered by:
  - timeout events
  - duplicate ACKs (*fast retransmit*)

Tricks for reliable data transfer

The length of timeout intervals?

Mechanism	Use, Comments
Checksum	To detect bit errors
Timer	To timeout/retransmit a packet (lost/premature packet)
Sequence number	To detect a lost packet (gap in seq#) To detect a duplicate packet (duplicate seq#)
Acknowledgement	To notify successful reception (individual/cumulative)
Negative ACK	To notify unsuccessful reception
Window, pipelining	To improve sender utilisation (utilisation vs performance)

TCP Round Trip Time and Timeout

Q: how to set TCP timeout value?

- longer than RTT
  - but RTT varies
- too short: premature timeout
  - unnecessary retransmissions
- too long: slow reaction to segment loss

TCP Round Trip Time and Timeout

Q: how to estimate RTT?

- **`SampleRTT`**: measured time from segment transmission until ACK receipt
  - ignore retransmissions (*do not sample retransmitted segments*);
- **`SampleRTT`** will vary, want estimated RTT "smoother"
  - average several recent measurements, not just current **`SampleRTT`**

TCP Round Trip Time and Timeout

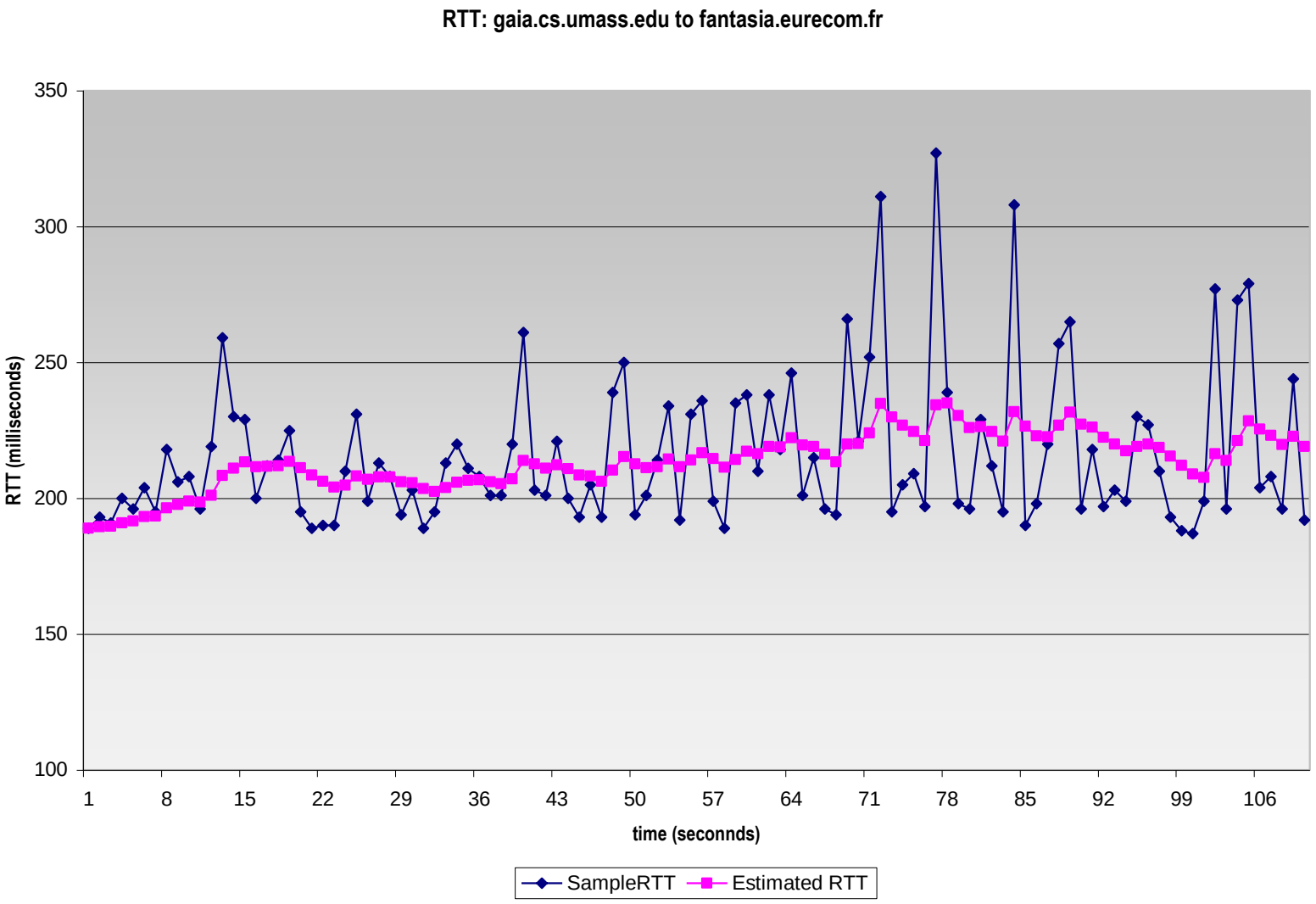
EstimatedRTT = (1- α)*EstimatedRTT + α*SampleRTT

$$E_0 = S_0$$
$$E_1 = (1-\alpha)E_0 + \alpha S_1 = (1-\alpha)S_0 + \alpha S_1$$
$$E_2 = (1-\alpha)E_1 + \alpha S_2 = (1-\alpha)^2 S_0 + \alpha(1-\alpha)S_1 + \alpha S_2$$

$$E_n = (1-\alpha)E_{n-1} + \alpha S_n = (1-\alpha)^n S_0 + \alpha(1-\alpha)^{n-1} S_1 + \alpha(1-\alpha)^{n-2} S_2 + \ldots + \alpha(1-\alpha)S_{n-1} + \alpha S_n$$

- *Exponential weighted moving average (EWMA)*
- influence of past sample decreases exponentially fast
- typical value: α = 0.125
- The weighted average puts more weight on recent samples which better reflect the current congestion

Example RTT estimation:



TCP Round Trip Time and Timeout

Setting the timeout

- To measure the variability of the RTT;
- first estimate of how much SampleRTT deviates from EstimatedRTT:

$$DevRTT = (1-\beta)*DevRTT + \beta*|SampleRTT-EstimatedRTT|$$

(typically, $\beta = 0.25$)

Then set timeout interval:

$$TimeoutInterval = EstimatedRTT + 4*DevRTT$$

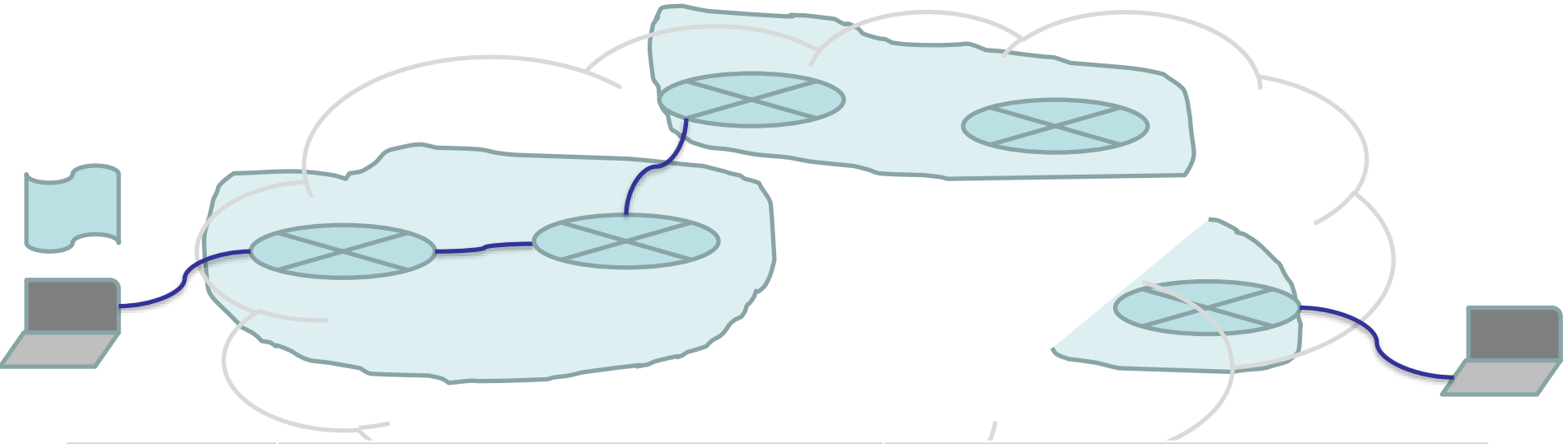
Tricks for reliable data transfer

The length of timeout intervals?

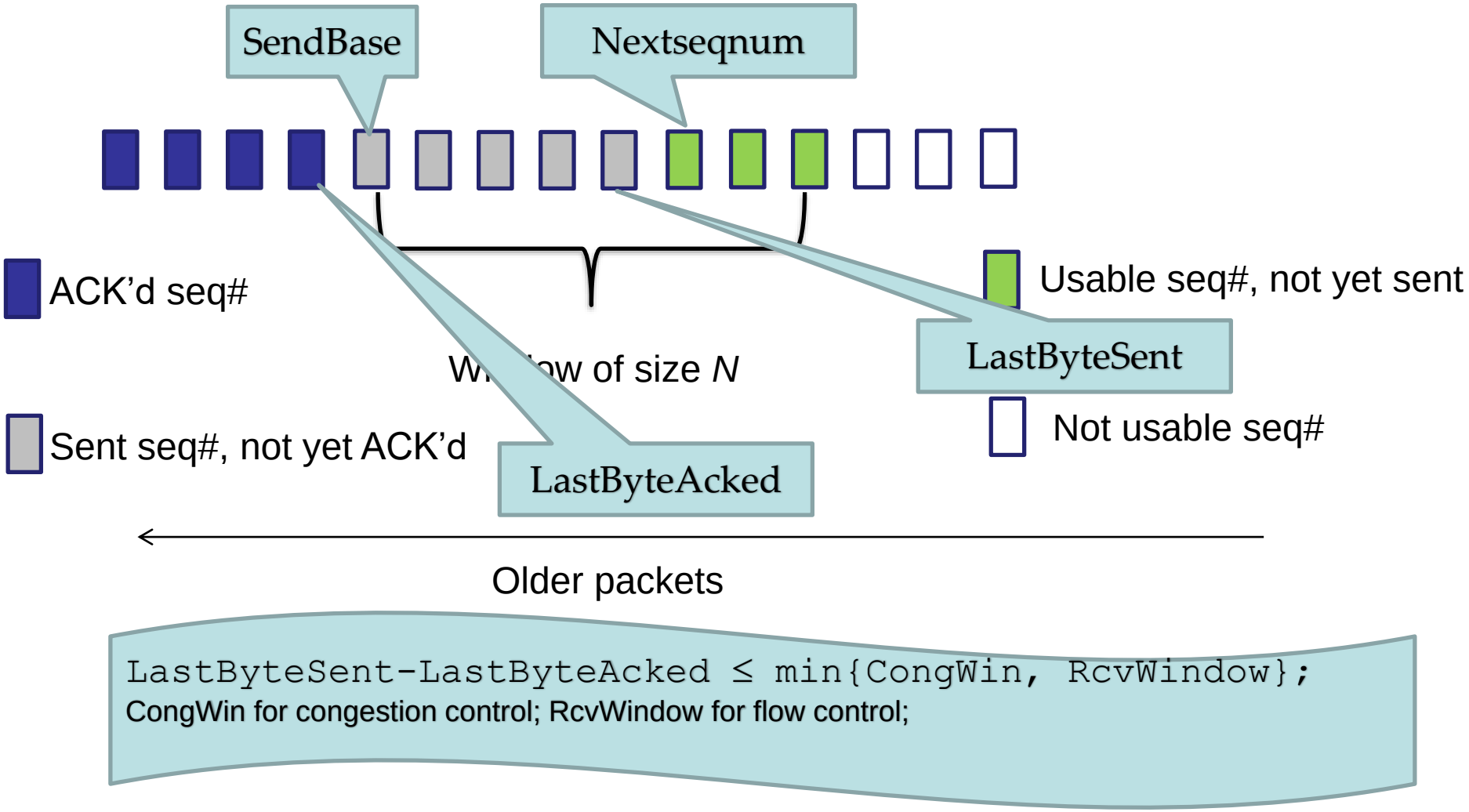
Mechanism	Use, Comments
Checksum	To detect bit errors
Timer	To timeout/retransmit a packet (lost/premature packet)
Sequence number	To detect a lost packet (gap in seq#) To detect a duplicate packet (duplicate seq#)
Acknowledgement	To notify successful reception (individual/cumulative)
Negative ACK	To notify unsuccessful reception
Window, pipelining	To improve sender utilisation (utilisation vs performance)

How to set the window size?

The journey of a packet



Problems	Causes	Solutions
Bit error	e.g., signal attenuation/noise	Error detection and correction
Buffer overflow	e.g., Speed-mismatch; Too much traffic;	Flow control and congestion control
Lost packet	e.g., buffer overflow at host/router	Acknowledgement and retransmission (ARQ) - RDT
Out of order	e.g. an early packet gets lost and retransmitted; a later one arrives first.	Acknowledgement and retransmission (ARQ) - RDT



`LastByteSent-LastByteAcked ≤ min{CongWin, RcvWindow};`

CongWin for congestion control; RcvWindow for flow control;

Outline

- TCP Reliable data transfer
  - A mixture of GBN and SR;
  - Fast retransmit;

References

- [KR3] James F. Kurose, Keith W. Ross, *Computer networking: a top-down approach featuring the Internet*, 3rd edition.
- [PD5] Larry L. Peterson, Bruce S. Davie, *Computer networks: a systems approach*, 5th edition
- [TW5] Andrew S. Tanenbaum, David J. Wetherall, *Computer network*, 5th edition
- [LHBi]Y-D. Lin, R-H. Hwang, F. Baker, *Computer network: an open source approach*, International edition

Acknowledgements

- All slides are developed based on slides from the following two sources:
  - Dr DongSeong Kim's slides for COSC264, University of Canterbury;
  - Prof Aleksandar Kuzmanovic's lecture notes for CS340,Northwestern University, https://users.cs.northwestern.edu/~akuzma/classes/CS340-w05/lecture_notes.htm