Performance of a "stop & wait" protocol

- rdt3.0 works, but performance stinks
- example: 1 Gbps link, 15 ms e-e prop. delay, 1KB packet:

$$T_{transmit} = \frac{8kb/pkt}{10**9\ b/sec} = 8\ microsec$$

$$\text{Utilization} = U = \frac{\text{fraction of time sender busy sending}}{} = \frac{8\ microsec}{30.008\ msec} = 0.00027$$

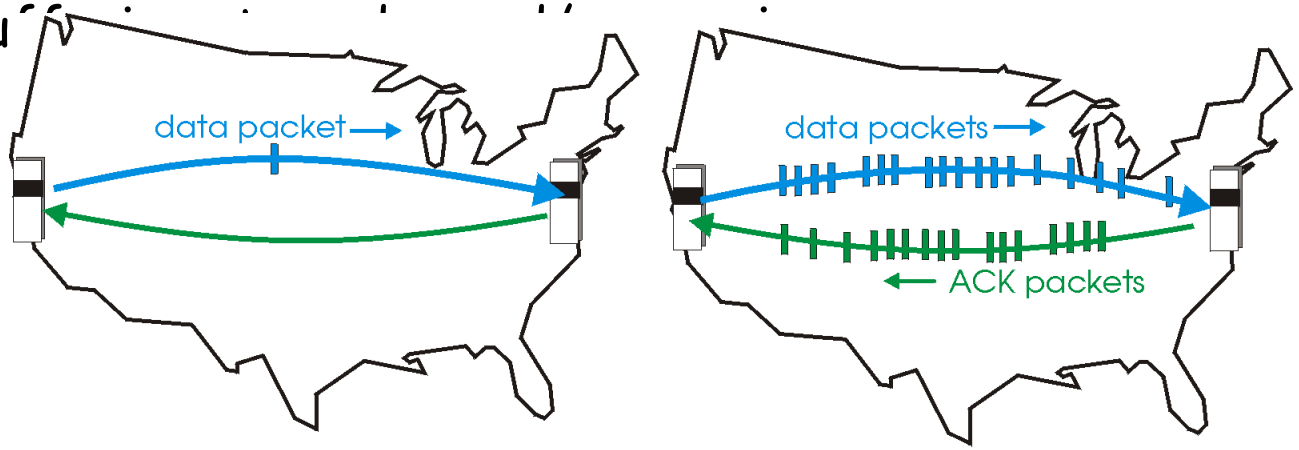
30.008 msec = total propagation delay + transfer delay,
when ignoring the transmission delay of ACK

- 1KB pkt every 30 msec (πόσα KB ανά sec ?) -> 33kB/sec throughput over 1 Gbps link
  - **network protocol limits use of physical resources!**

Pipelined protocols

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

- range of sequence numbers must be increased
- bu[illegible]



(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*

Sequence numbers

- Πρώτη γραμμή άμυνας ως προς: packet loss, reordering and mis-insertion.

  Tx → 0, 1, 2, 3, 4, … (αν το #4 χαθεί)

  Rx → 0, 1, 2, 3, 5, 6… (γίνεται εύκολα αντιληπτό από την αρίθμηση)

  Mis-insertion: 0, 1, 2, 3, 569, 4, 5, 6, … (ομοίως, αντιληπτό)

- Πόσο μεγάλο πρέπει να είναι το πεδίο Sequence number ενός πρωτοκόλλου;

  Πεδίο Seq. # : ν bits ⇒ αρίθμηση 2^{ν}

  Έστω **ν=3** τότε: 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3, 4, 5, …
  Έστω **ν=5** τότε:

  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,…

  Αν γίνει λάθος στο πακέτο 20, αν είχαμε πεδίο Seq. #, **ν=5 bits**, τότε θα μας πληροφορούσε με **σαφήνεια** ο παραλήπτης ότι υπάρχει λάθος στο πακέτο 20.

  Αν όμως είχαμε πεδίο Seq. #, **ν=3 bits**, τότε ο παραλήπτης θα μας έλεγε ότι υπάρχει πρόβλημα στο πακέτο 4. Ο αποστολέας όμως έχει στείλει **3 πακέτα** με Seq. # 4, και επομένως δεν ξέρει σε ποιο από τα 3 πακέτα υπάρχει λάθος.

Υπολογισμός του μήκους του πεδίου Seq.#

- **MPL** (max packet life) ή **MSL** (max segment life), in sec.
- **T** (sec) : max time a sender waiting for ACK persists in transmitting a packet.
- **A** (sec) : max time a receiver can delay before sending ACK.
- **R** (packets/sec) : max transmission rate of sender.

A packet has Seq. # 4 and is repeatedly lost by the network.

The sender keeps retransmitting the packet until time T, after its 1st transmission of this packet. In the worst case:

The receiver successfully receives the packet the final time it is transmitted: at time T + MPL. «Κοιμάται» για χρόνο A πριν απαντήσει ACK, το οποίο για να φθάσει στον αποστολέα περνά χρόνος MPL.

Εν τω μεταξύ, δηλ. στο διάστημα T+MPL+A+MPL, ο αποστολέας έχει στείλει (T+2MPL+A)*R packets, δηλ. επιπρόσθετους Seq. #.

Υπολογισμός του μήκους του πεδίου Seq.#

- Για να μη μηδενιστεί η αρίθμηση των πακέτων και αρχίσει από την αρχή, δηλ. για να είναι σίγουρος ο αποστολέας για ποιο πακέτο έλαβε ACK, θα πρέπει: πεδίο seq.# = v bits, όπου:
- **$2^v \geq (T+2MPL+A)*R$**
- Παράδειγμα:

  MPL = 2 min

  T = 1 min

  A = 0,5 sec

  R = 2 Mbps

  min packet size = 40 bytes

⇒ $2^v \geq (60 + 2x2x60 + 0,5)*(2x10^6)/(40x8)$ ⇒

$2^v \geq 1878125$ ⇒ **$v \geq 21$**

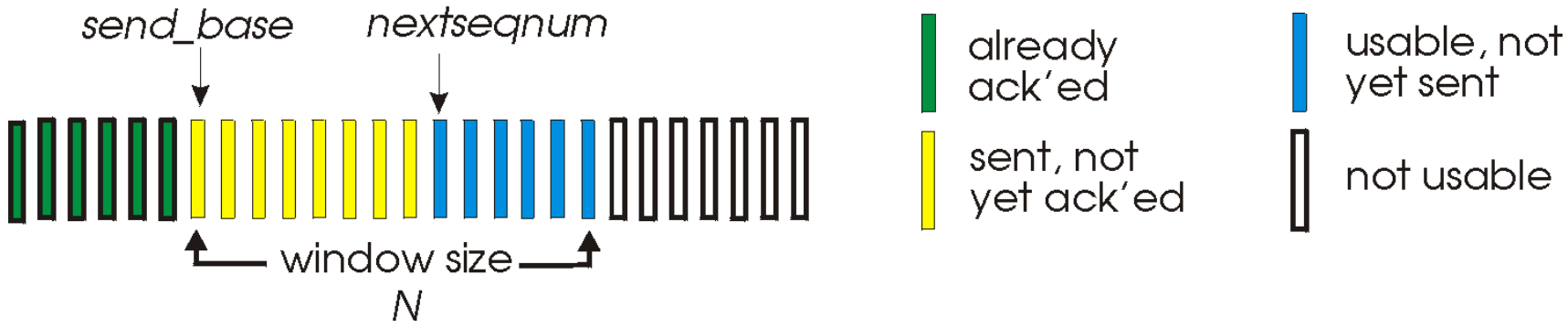
Σημειωτέον, $2^{20} = 1048576$

$2^{21} = 2096152$

Go-Back-N

Sender:

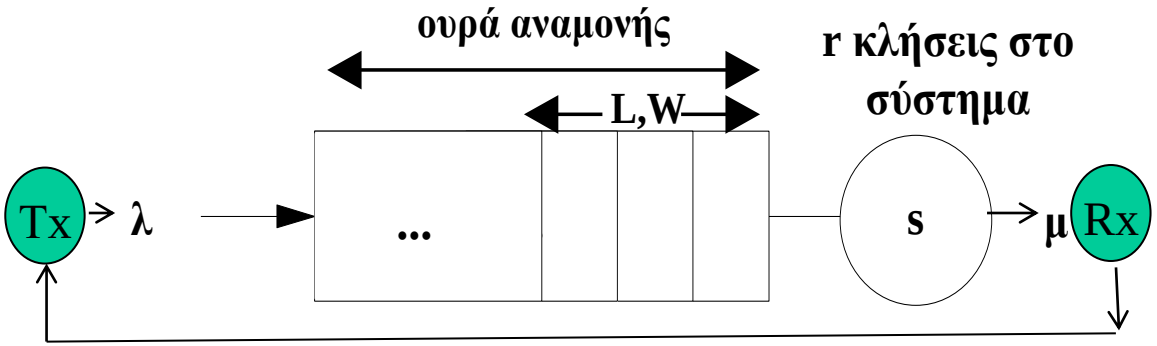
- k-bit seq # in pkt header
- "window" of up to N, consecutive unack'ed pkts allowed (= **ECW**)



- ACK(n): ACKs all pkts up to, including seq # n - "cumulative ACK"
  - may deceive duplicate ACKs (see receiver)
- timer for each in-flight pkt
- *timeout(n)*: retransmit pkt n and all higher seq # pkts in window

Error Control Window - ECW

- Στέλνει w packets και περιμένει ACK. Ο αποστολέας δεν γνωρίζει αν τα πακέτα στο ECW έχουν ληφθεί σωστά από τον παραλήπτη ή έχουν χαθεί.
- Bit errors cause a bit to flip from "1" to "0" or vice versa.
- Packet errors: packet loss, damage, duplication or reordering.
- Static window flow control - Ένα εύκολο μοντέλο: αργός Router



- Stop & wait protocol (ECW = 1), συγχρόνως κάνει:

  error control: αν χαθεί το πακέτο (timeout) ο αποστολέας το ξαναμεταδίδει μέχρις ότου λάβει ACK.

και **flow control**: ο αποστολέας πρέπει να περιμένει ACK πριν μεταδώσει το επόμενο πακέτο. Επομένως η μετάδοση γίνεται ανάλογα με την δυνατότητα επεξεργασίας του δέκτη (έλεγχος ροής)

Error Control Window – ECW (συνέχεια)

- Stop & wait **Throughput** = 1 /RTT (packets/s) **Goodput** (bytes χρήστη/s)
- Αν έχουμε ECW με w = 3 packets τότε

  Max. Throughput = w /RTT (packets/s)
- Έστω ο αργός Router με μ packets/s
- Θέλουμε $\lambda = w/RTT$ packets/s $\approx \mu \Rightarrow$ **$w = \mu * RTT$**

  Βέλτιστο παράθυρο μετάδοσης : Bandwidth x Delay product!
- Παράδειγμα

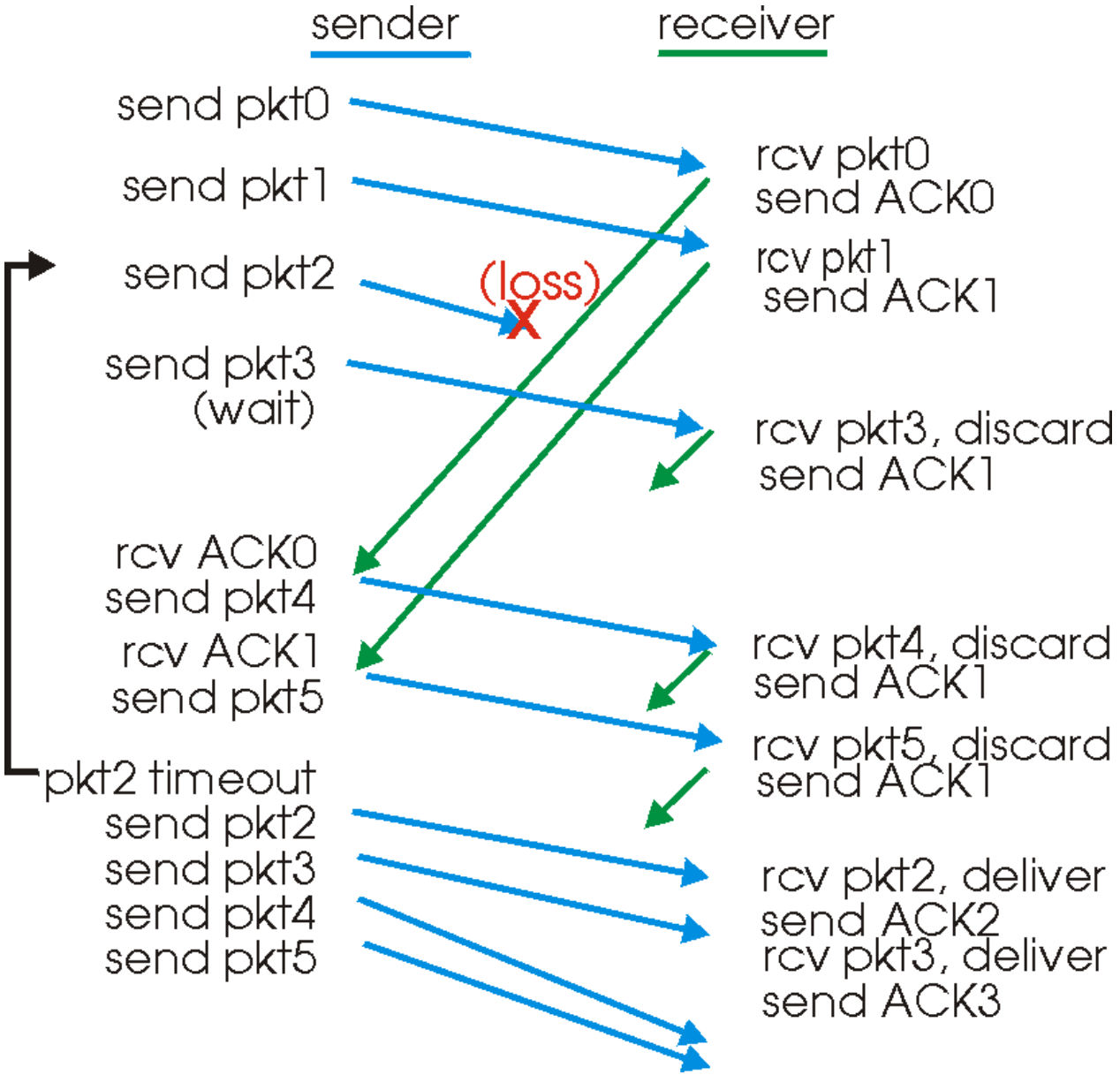
  Ποιο είναι το βέλτιστο παράθυρο μετάδοσης όταν packet size = 53 bytes

  RTT = 60 ms και μ = 2 Mbps

  Τότε $\mu = 2x10^6/(53x8) = 4716,98$ packets/sec

  και w = 4716,98 x 0,06 = 283 packets

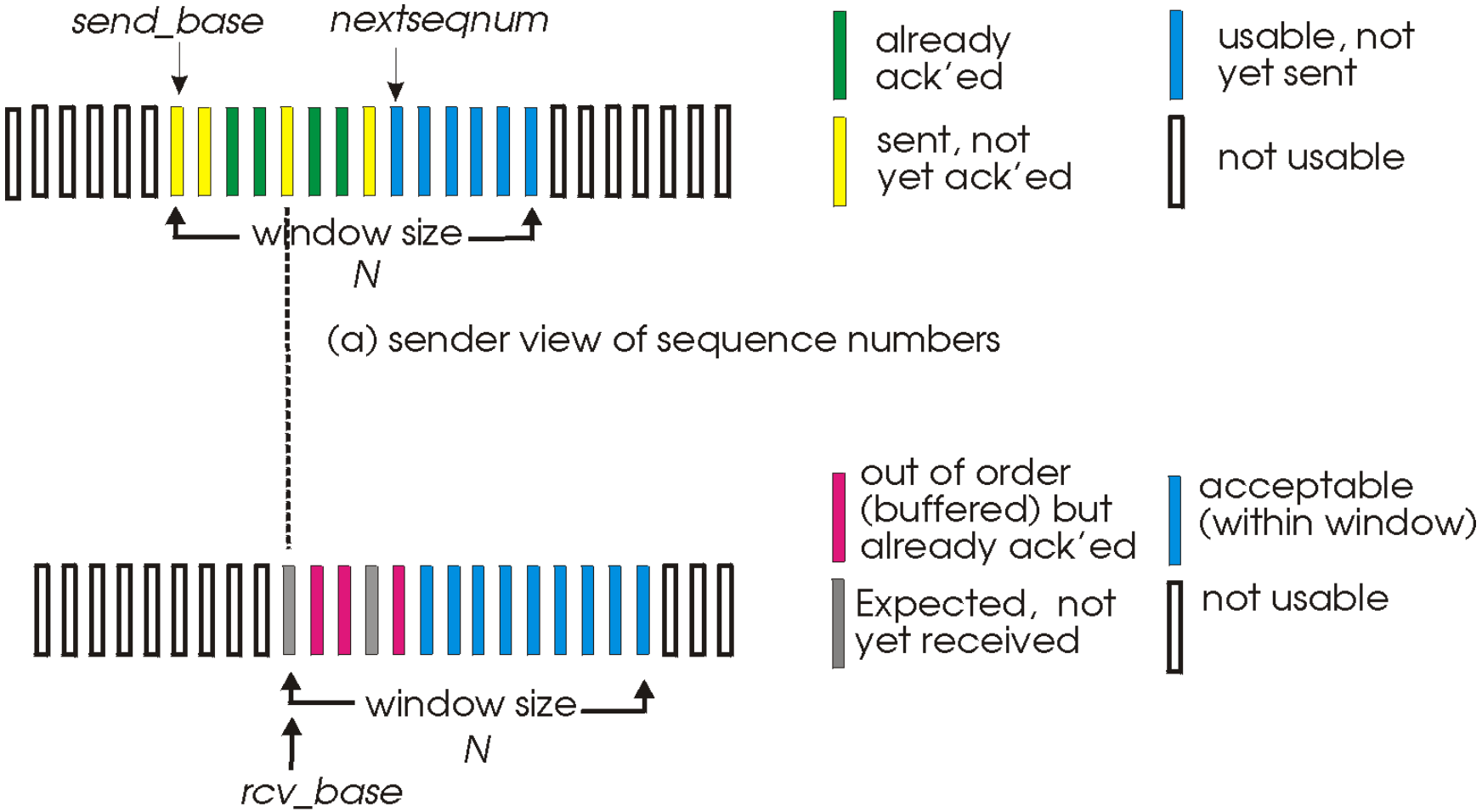
GBN in action



Selective Repeat

- receiver *individually* acknowledges all correctly received pkts
  - buffers pkts, as needed, for eventual in-order delivery to upper layer
- sender only resends pkts for which ACK not received
  - sender timer for each unACKed pkt
- sender window
  - N consecutive seq #'s
  - again limits seq #s of sent, unACKed pkts

Selective repeat: sender, receiver windows



Selective repeat

sender

data from above :

- if next available seq # in window, send pkt

timeout(n):

- resend pkt n, restart timer

ACK(n) in [sendbase,sendbase+N]:

- mark pkt n as received
- if n smallest unACKed pkt, advance window base to next unACKed seq #

receiver

pkt n in [rcvbase, rcvbase+N-1]

- send ACK(n)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order pkts), advance window to next not-yet-received pkt

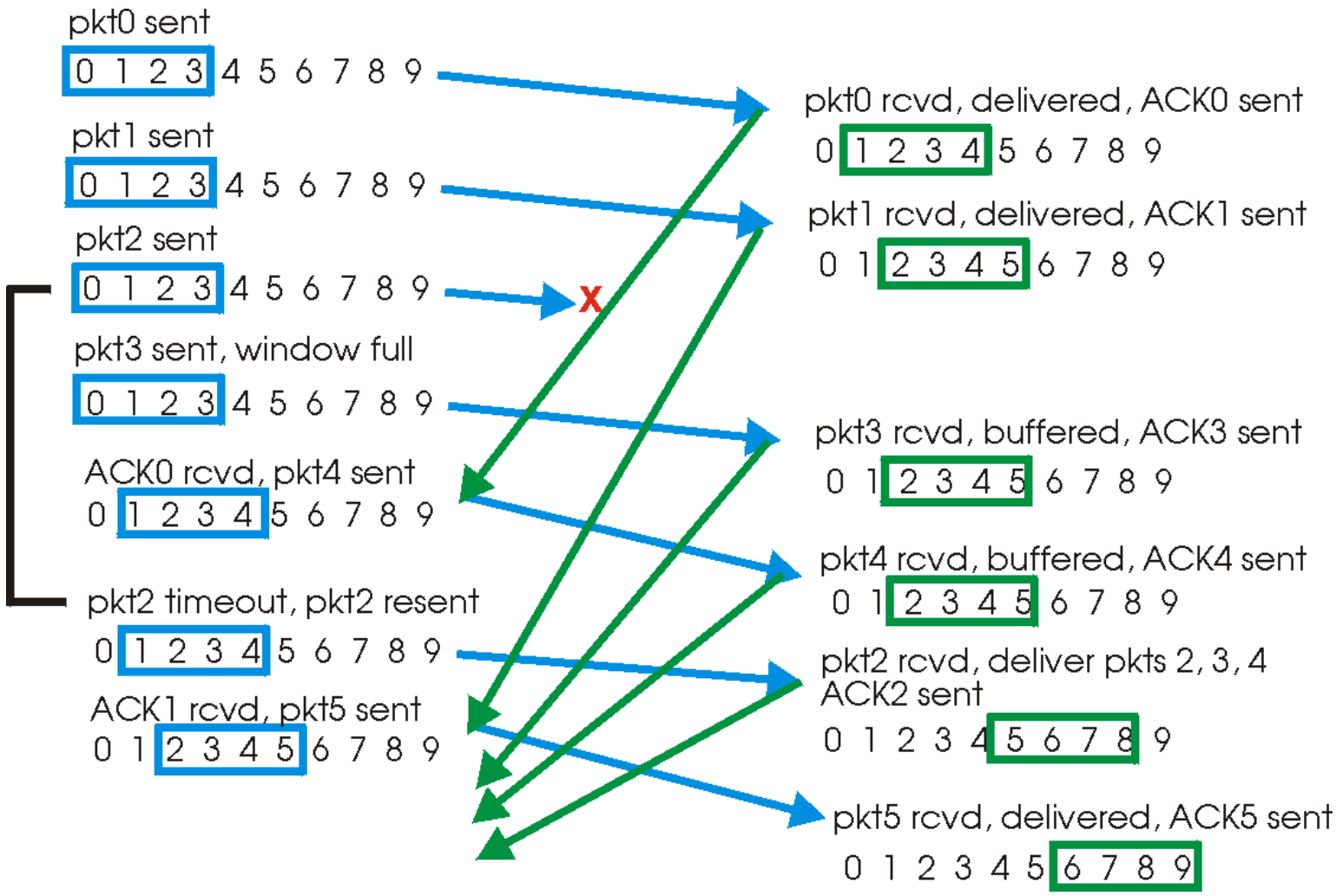
pkt n in [rcvbase-N,rcvbase-1]

- ACK(n)

otherwise:

- ignore

Selective repeat in action



Selective repeat: dilemma

Example:

- seq #'s: 0, 1, 2, 3
- window size=3

- receiver sees no difference in two scenarios!
- incorrectly passes duplicate data as new in (a)

Q: what relationship between seq # size and window size?