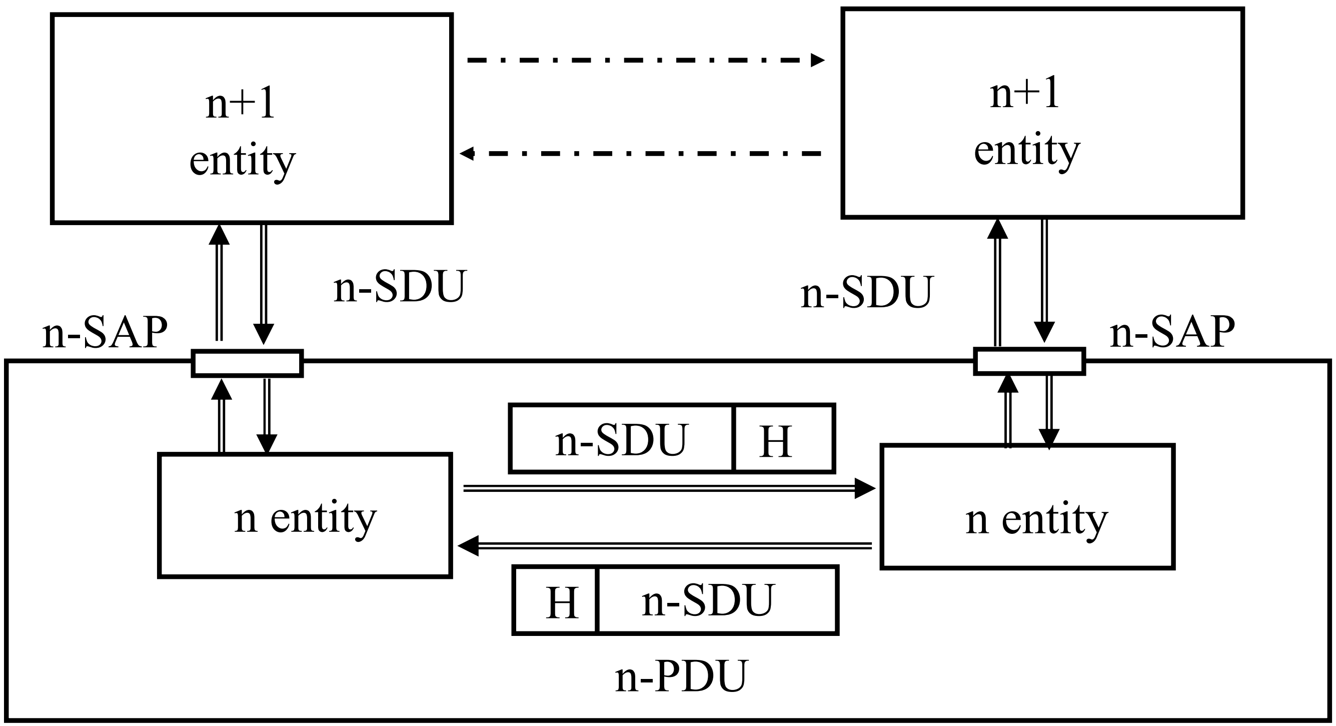
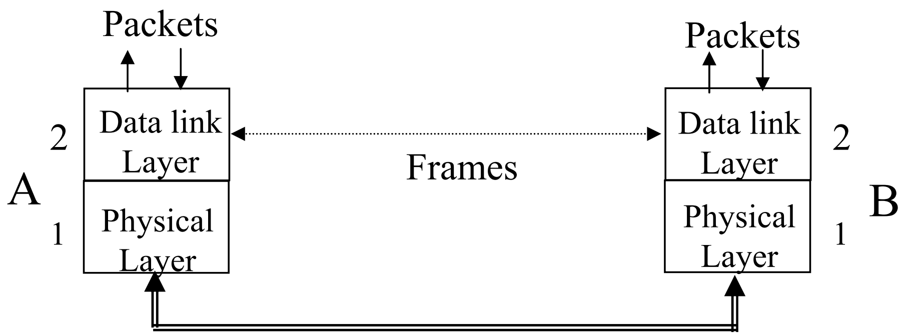
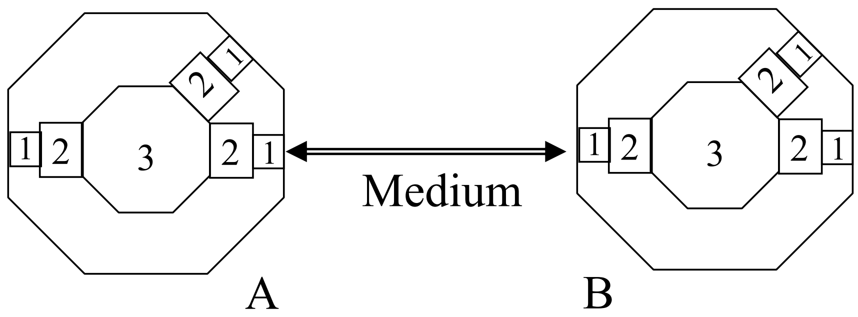
Peer-to-Peer Protocols

- the interaction of two processes through the exchange of messages (PDUs)
  - to provide a service to a higher level
    - » e.g. confirmation of receipt, sequence order guarantees, maximum delay etc.

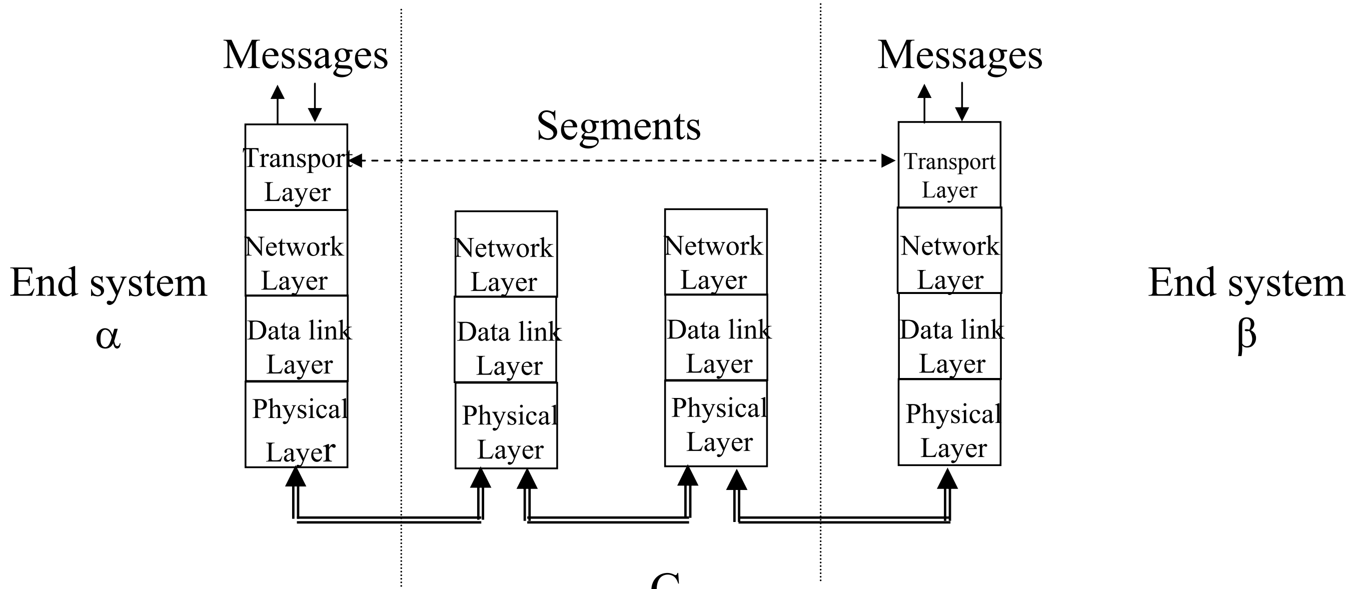


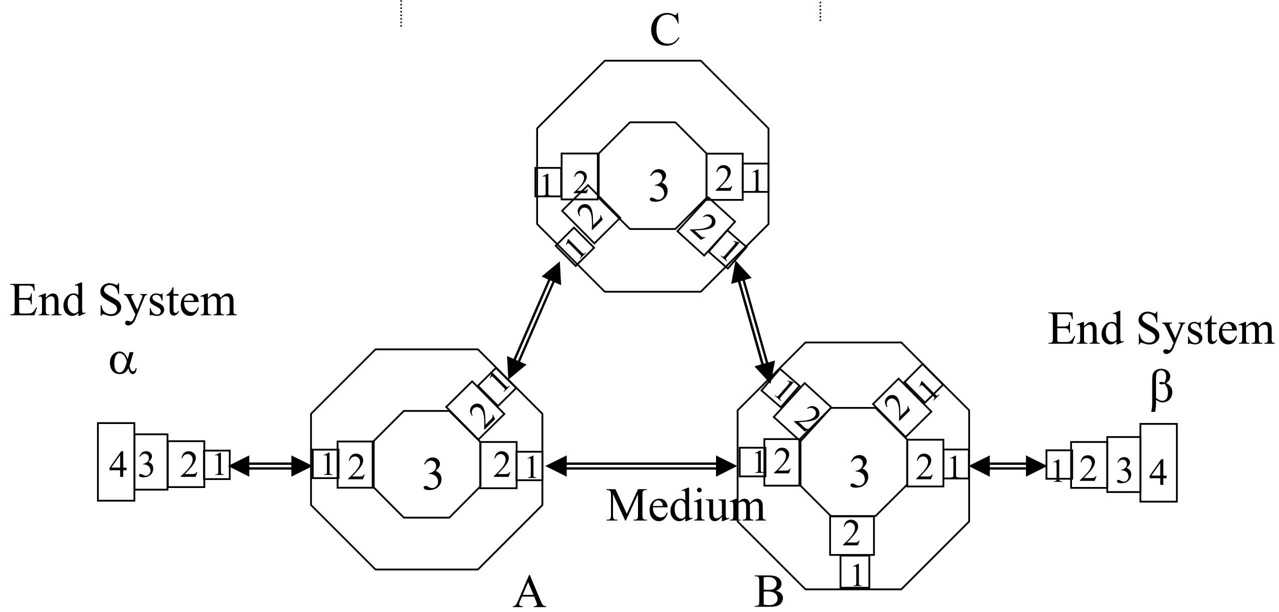
- either across a single hop in the network or across an entire network
  - affects whether PDUs arrive in order, how long they take to arrive etc.
  - corresponding protocols have to take these characteristics into account
- across a single hop at the Data Link layer:



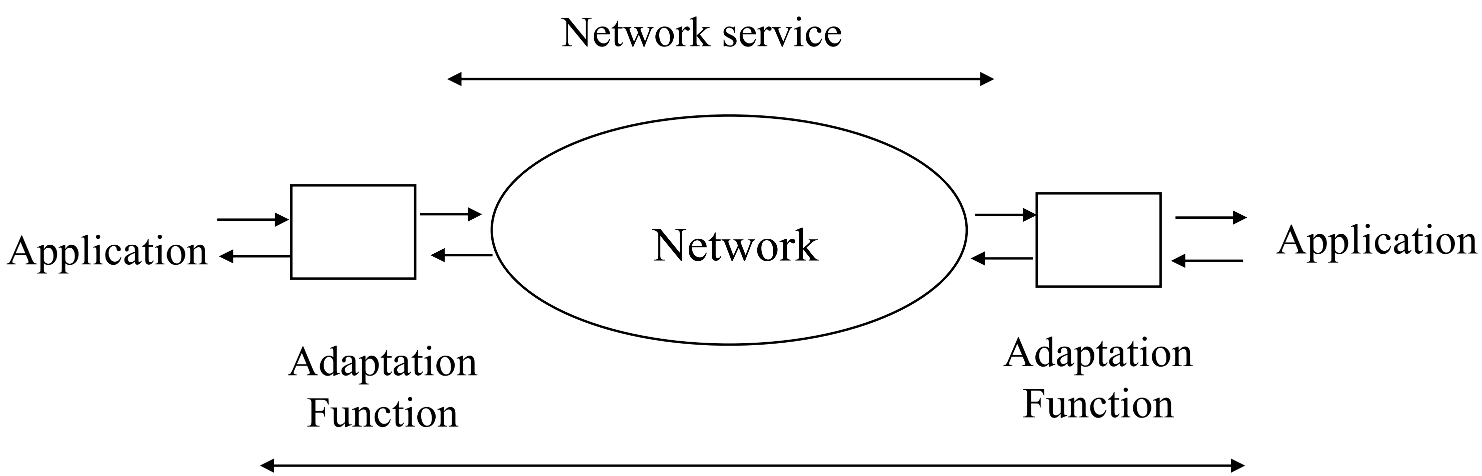


• end-to-end across an entire network:





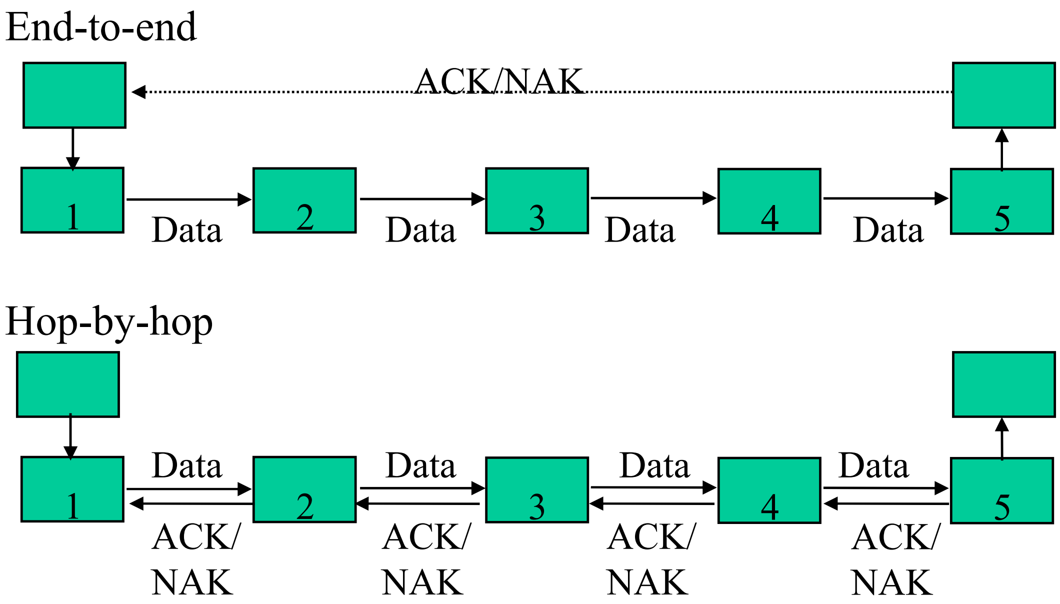
- Service models
  - quality of service
    - » required levels of performance
    - » probability of errors, probability of incorrect delivery, transfer delay etc.
  - end-to-end requirements
    - » arbitrary message size, reliability and sequencing, pacing and flow control, timing, addressing, privacy and authentication
    - » may be provided by adaptation functions between applications and network



    - » cannot all be provided by interposed functions
      - underlying network may not be able to provide level of service required

– many adaptation functions can be introduced either on a hop-by-hop basis or end-to-end across an entire network

» e.g. error control with acknowledgement of packets and possible retransmission:



– trade-off :

» hop-by-hop initiates error recovery more quickly – good for unreliable links

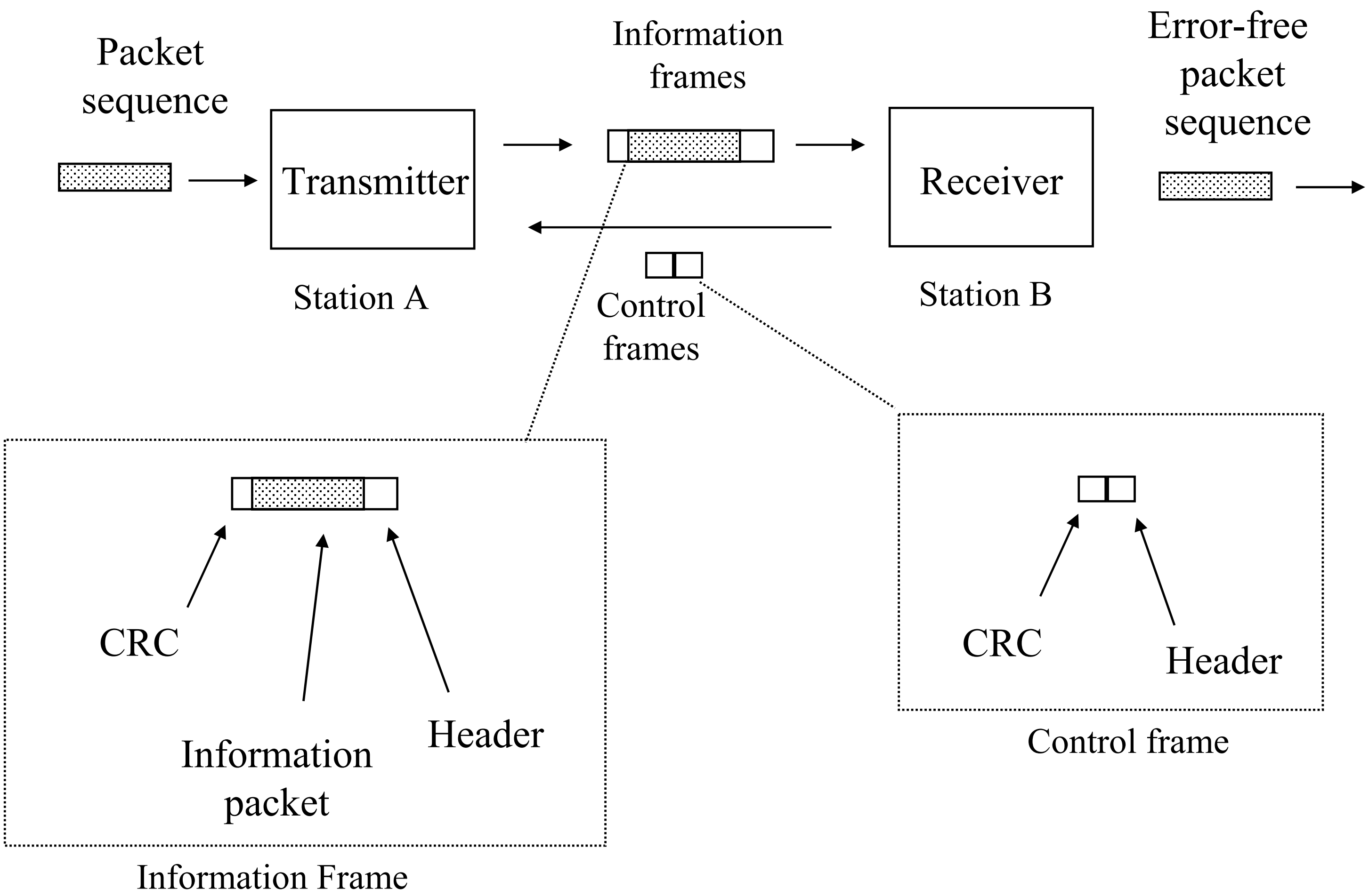
- but processing in each node more complex
- may be slower depending on the ACK/NAK protocol used

» where errors are infrequent, end-to-end mechanism is preferred

– hop-by-hop versus end-to-end choice also relevant to flow control & congestion

- ARQ (Automatic Repeat Request) Protocols
  - used in protocol layers where reliable delivery of a data stream is required in the presence of errors
  - assumes a steady stream of blocks to be transmitted
  - blocks required to contain a header with control information and a trailer with CRC information (covering the header and the data) to allow error detection
    - » assume trailer allows error detection with high degree of probability
  - can be used over a single hop or end-to-end
    - » for end-to-end use, frames assumed to arrive *in order* and just *once*
      - if they arrive at all
    - » where a connection is set up which all frames follow, such as ATM networks
  - *information frames* (*I-frames*) transfer user packets
  - *control frames* are short blocks that just consist of a header and a CRC
    - » include ACKs which acknowledge correct receipt of a frame
    - » and *enquiry frames*, ENQs, which require the receiver to report its status
  - *time-outs* are required to prompt certain actions to maintain the flow

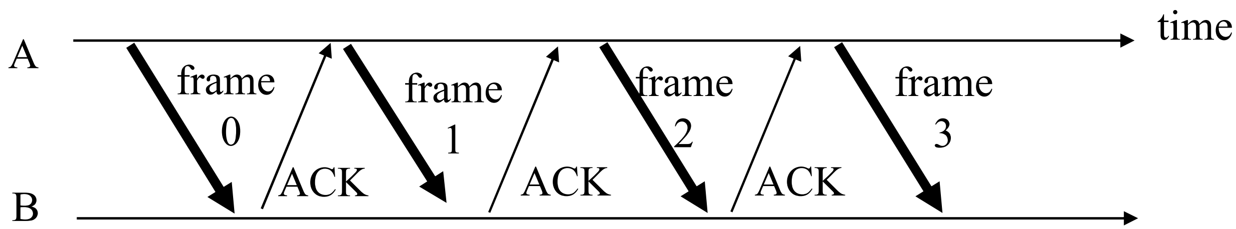
– basic elements of ARQ:



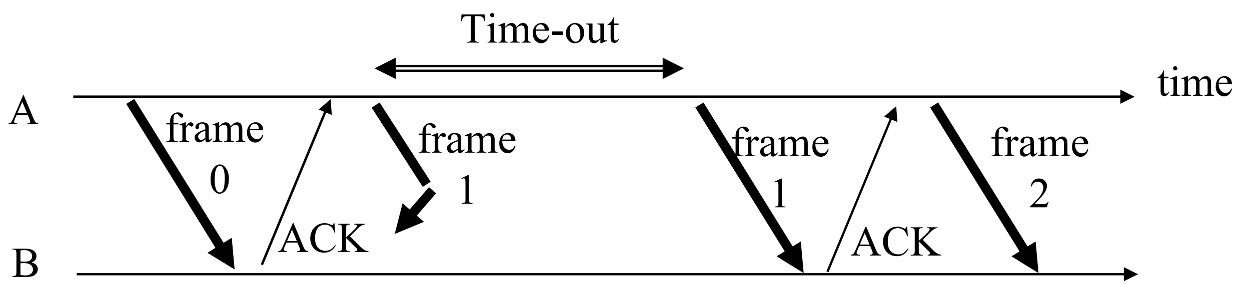
– assume initially that information flow is unidirectional

» reverse channel only used of transmission of control information

- Stop-and-Wait ARQ
  - transmitter and receiver deal with one frame at a time
    - » transmitter waits to receive an acknowledgment from receiver that it has received the frame correctly before it transmits the next frame



  - each time A transmits a frame it also starts a timer
    - » set to expire at some chosen point in the future
    - » depending on speed of channel, length of frame, time for B to respond etc.
  - example of a frame lost in transmission from A:



– station A transmits frame 0 (and starts its timer), then waits for an ACK frame from station B

– frame 0 is received without error, so station B transmits an ACK frame back

– the ACK from B is also received without error, so A knows frame 0 has been received by B correctly

– A now proceeds to transmit frame 1 (and restarts its timer)

– when frame 1 has *errors* in transmission

  » B may receive the frame and detect an error from the CRC

  » B maybe did not receive any frame at all

  » in either case, B takes *no action* i.e. does not return an ACK

    - NAKs not used in this protocol

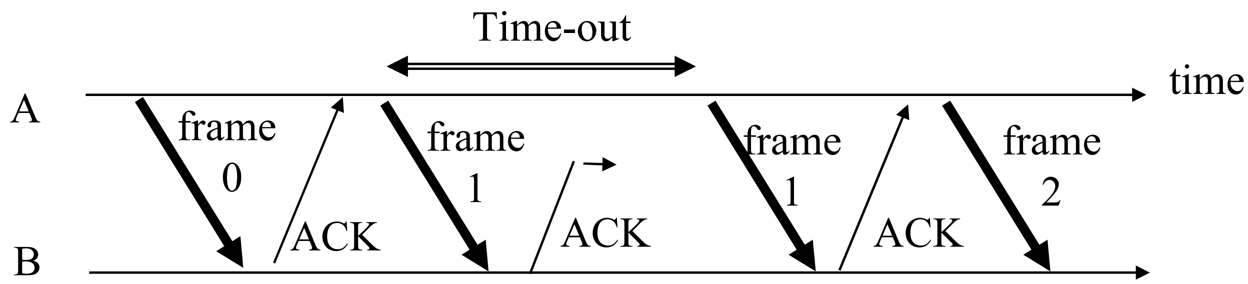
– A's time-out expires, so it retransmits frame 1

– A continues this sequence of waiting for its time-out to expire and retransmission

  » until B acknowledges frame 1 successfully

  » then A moves on to transmit frame 2

– suppose the ACK of frame 1 is lost on the way back to A :



– after receiving frame 1, B *delivers its contents to the higher-level user of it*

– A does not receive the ACK, so its time-out expires

» A cannot tell the difference of this from the first case

– A retransmits frame 1

– B accepts the retransmitted frame 1 as a new frame

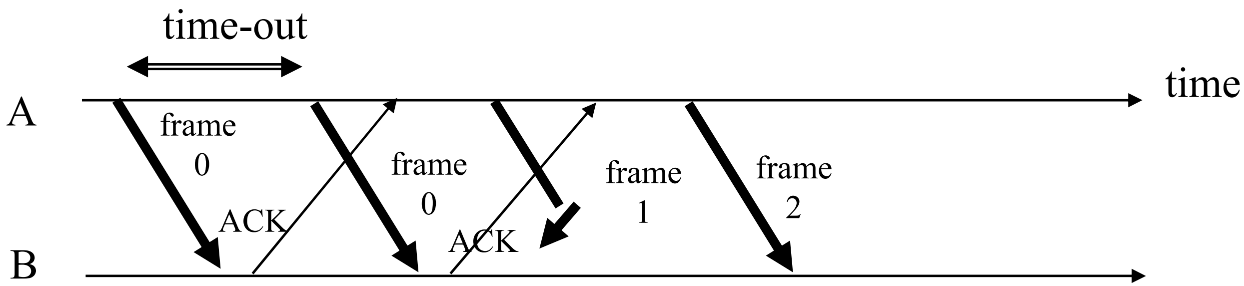
» and delivers it to its user *again*

» i.e. the user receives a *duplicate* packet

– this ambiguity is eliminated by including a sequence number in the I-frame header

» B can then recognise that the packet was a retransmission, discard it, and send the ACK again back to A

– another type of ambiguity can arise if the time-out has been inadvertently set too short :



– frame 0 is received correctly and the ACK returned

– in the meantime, the time-out expired too soon and A has retransmitted frame 0

– A assumes that the ACK it now receives is for the retransmitted frame 0

  » so then transmits the next frame, frame number 1

– B meantime transmits an ACK back to A for the retransmitted frame 0

– A assumes the next ACK (really for the retransmitted frame 0) is for frame 1

  » and transmits frame 2 to B

– if frame 1 does not reach B, it will not transmit an ACK back to A for it

  » but overall frames received match ACKs sent

– upshot is that frame 1 is *lost forever*

– this ambiguity can be resolved by providing a sequence number in the acknowledgment

» transmitter then knows which frame has been received

– transmitter keeps track of the sequence number S_{last} of the frame being sent

» plus the frame itself, in case it needs to be retransmitted

– receiver keeps track only of the sequence number of the next frame it is expecting to receive, R_{next}

– sequence number must not get too large

» limited space in the I-frame header and the ACK packet

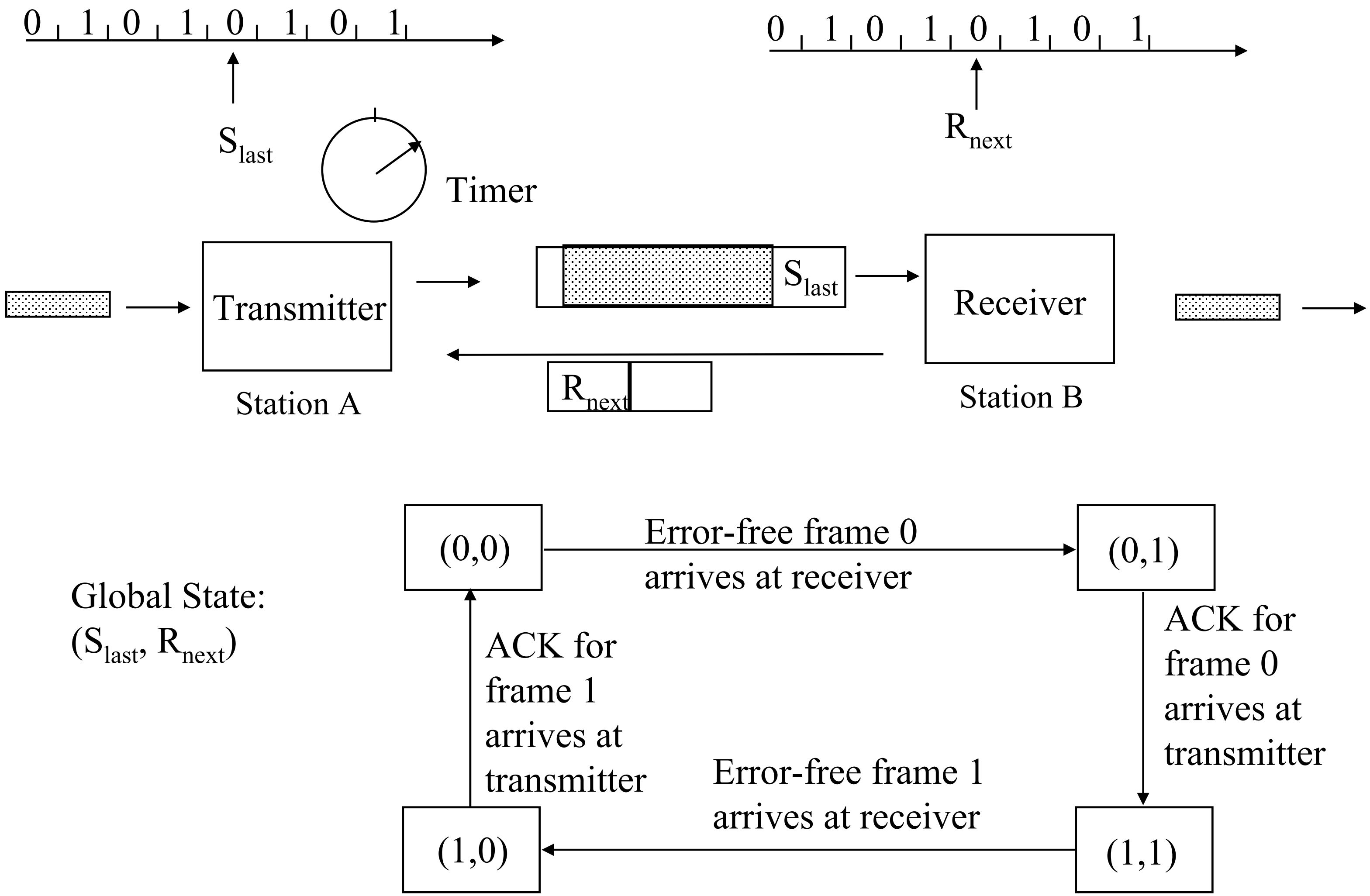
– a 1-bit sequence number adequate in this case

– the combination of S_{last} and R_{next} forms the *state* of the transmission link

» S_{last} will be 0 or 1; R_{next} will be 0 or 1

» therefore four states : (0,0), (0,1), (1,0), (1,1)

» depending on which frame has been transmitted and which ACKs received


0 1 0 1 0 1 0 1
S_last
Timer
Transmitter
Station A
S_last
R_next
Receiver
Station B
0 1 0 1 0 1 0 1
R_next
Global State:
(S_last, R_next)
(0,0)
Error-free frame 0 arrives at receiver
(0,1)
ACK for frame 0 arrives at transmitter
(1,1)
Error-free frame 1 arrives at receiver
(1,0)
ACK for frame 1 arrives at transmitter

– assume A ann B are synchronised and start in state (0,0)

– station A transmits frame 0 with S_{last} = 0

– system state does not change until B receives an error-free frame 0

  » i.e. A continues to retransmit according to its time-out mechanism

– eventually B receives frame 0, changes R_{next} to 1 and sends an ACK to A with R_{next} set to 1

  » implicitly acknowledging receipt of frame 0

– state is now (0,1)

– any subsequent frames with sequence number 0 are recognised as duplicates and discarded by B

  » and an acknowledgment resent to A with R_{next} = 1

– eventually A receives an acknowledgment with R_{next} = 1

  » and starts to transmit frame 1 using S_{last} = 1

– A and B are now synchronised again and system is in state (1,1)

– A and B now work together to deal with frame 1

  » and an orderly delivery of the sequence of frames is ensured

– error recovery can be expedited by use of an ENQ control frame

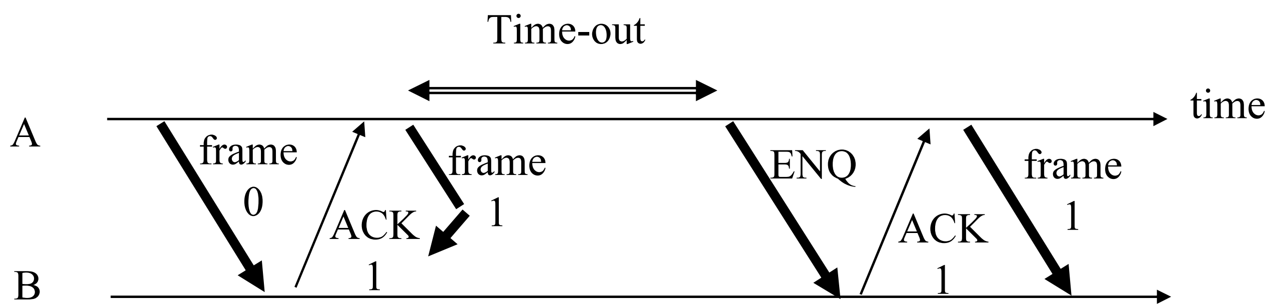
» which requires the receiver to retransmit its previous message

– it may be more efficient to enquire of the receiver whether it actually received a frame correctly instead of always retransmitting the frame

» e.g. if the ACK went missing

» and the frame is very long

– example: a frame is lost *en route*



» station A sends an ENQ instead of retransmitting frame 1

» if B returns its last message i.e. an ACK with $R_{next} = 1$

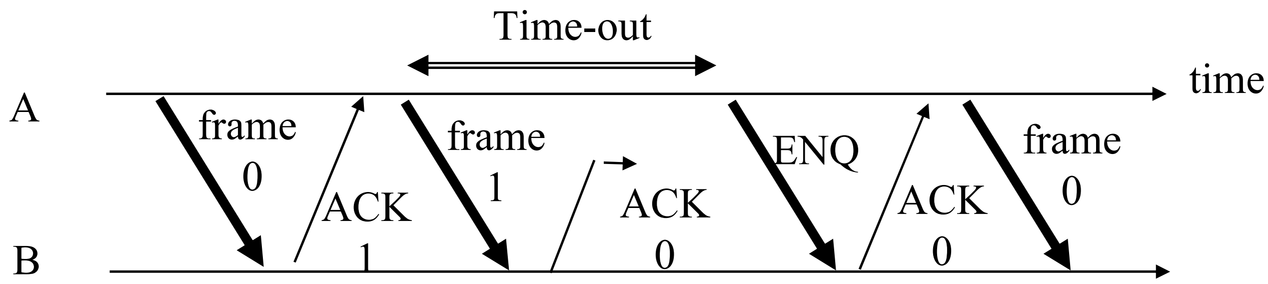
» A knows that frame 1 was not received correctly

- and can retransmit it

– if it was the ACK with R_{next} = 0 that got lost, A knows that frame 1 was correctly received by B

» it can proceed with transmitting the next frame

» and avoid wasting time retransmitting frame 1 :



– in general in ARQ protocols, the value of R_{next} implies that *all* the previous frames have been correctly received

- Stop-and-Wait can become very *inefficient*
  - » when propagation delay is significant in comparison with transmission time
- example: a 1.5Mbps channel, 1000bit packets, for 100km :
  - » propagation delay : c = 3×10^8m/s means 3.3ns/m; hence 0.33ms per 100km
  - » packet transmission time : $1000/1.5 \times 10^6$ secs = 0.67ms
  - » time between transmission of successive packets =

    prop delay A to B + packet trans time + prop delay of ACK B to A + overhead

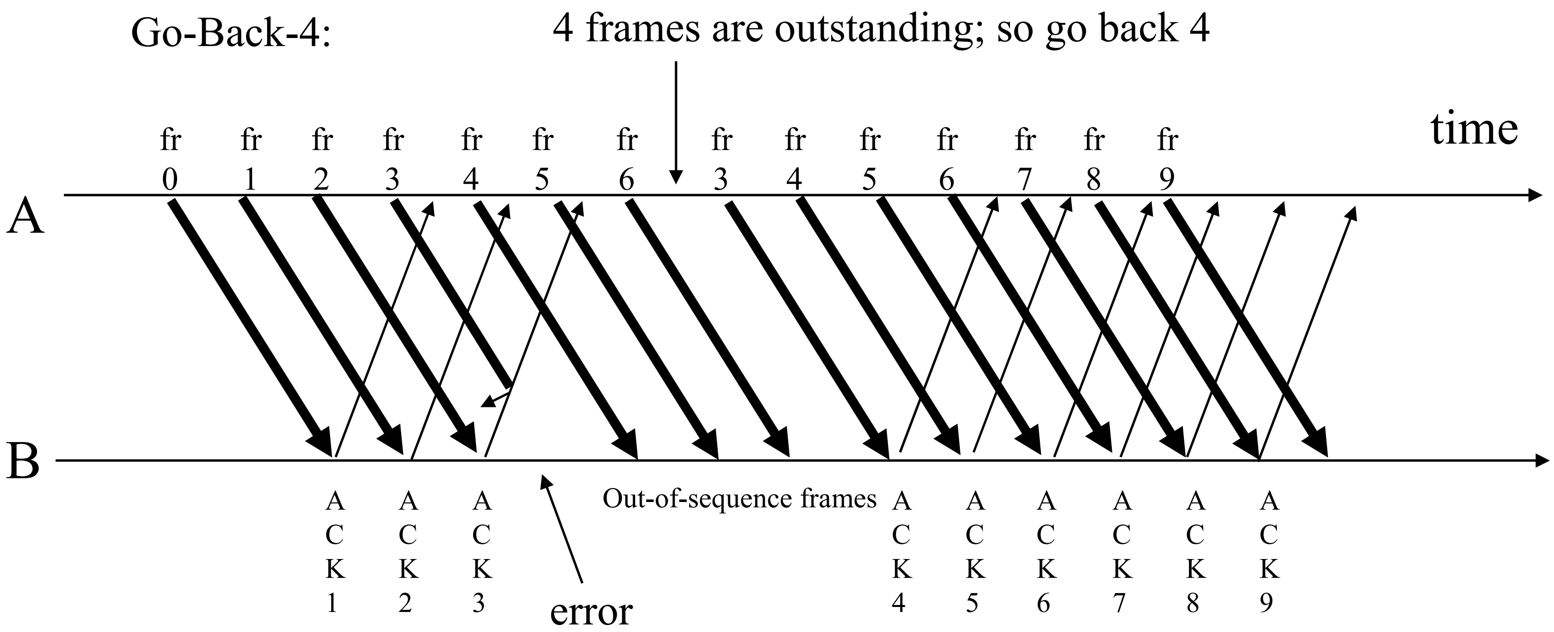
    = 0.33ms + 0.67ms + 0.33ms + ? = 1.33ms
  - » could transmit 2000 bits in this time : 50% efficient
- example: a 2.5Gbps channel, 10000bit packets, for 1000km
  - » prop delay : 3.3ms
  - » packet transmission time : $10000/2.5 \times 10^9$ = 4µs
  - » time between packets =

    3.3ms + 4µs + 3.3ms = 6.6ms = 6.6×10^{-3}secs
  - » could transmit $2.5 \times 10^9 \times 6.6 \times 10^{-3}$bits = 1.65×10^7bits : 0.06% efficient
- this *delay-bandwidth product* is a measure of *lost opportunity* to transmit bits

- Stop-and-Wait protocol was used by IBM for their Bisync protocol
  - » Binary Synchronous Communications
  - » used for transmitting blocks of character-oriented data from remote terminals to mainframes
    - e.g. from card-readers, Remote Job Entry terminals, ATM machines etc.
    - 2400bps usual
  - » parity coded character codes + final checksum
  - » replaced eventually by SNA (Systems Network Architecture)
  - » but not dead yet!
    - Serengeti Systems Inc still make terminals using Bisync
- Xmodem :
  - » a popular file transfer protocol using Bisync
  - » 128bit – 1024-bit packets at 4096bps
  - » error detection by checksum or CRC
  - » Ymodem and Zmodem higher speed versions

- Go-Back-N ARQ
  - – inefficiency of Stop-and-Wait ARQ can be improved by allowing the transmitter to continue sending frames while waiting for acknowledgments
  - – forms the basis of the HDLC protocol at the OSI Data Link level
  - – suppose frames are numbered 0, 1, 2, 3, …
  - – transmitter has a limit on the number of frames that can be outstanding without acknowledgment : W_s
    - » W_s is chosen to allow the channel to be fully utilised
  - – consider the transfer of frame 0 :
    - » after frame 0 is sent, W_s-1 additional frames are sent
    - » hoping that frame 0 will be correctly received and not need retransmission
    - » all being well the ACK for frame 0 will arrive back
      - \- while it is already dealing with later frames
    - » continues transmitting ahead as long as ACKs arrive as expected
  - – a *pipelined* system

– what happens when an error occurs?



» if frame 3 has errors in transmission,

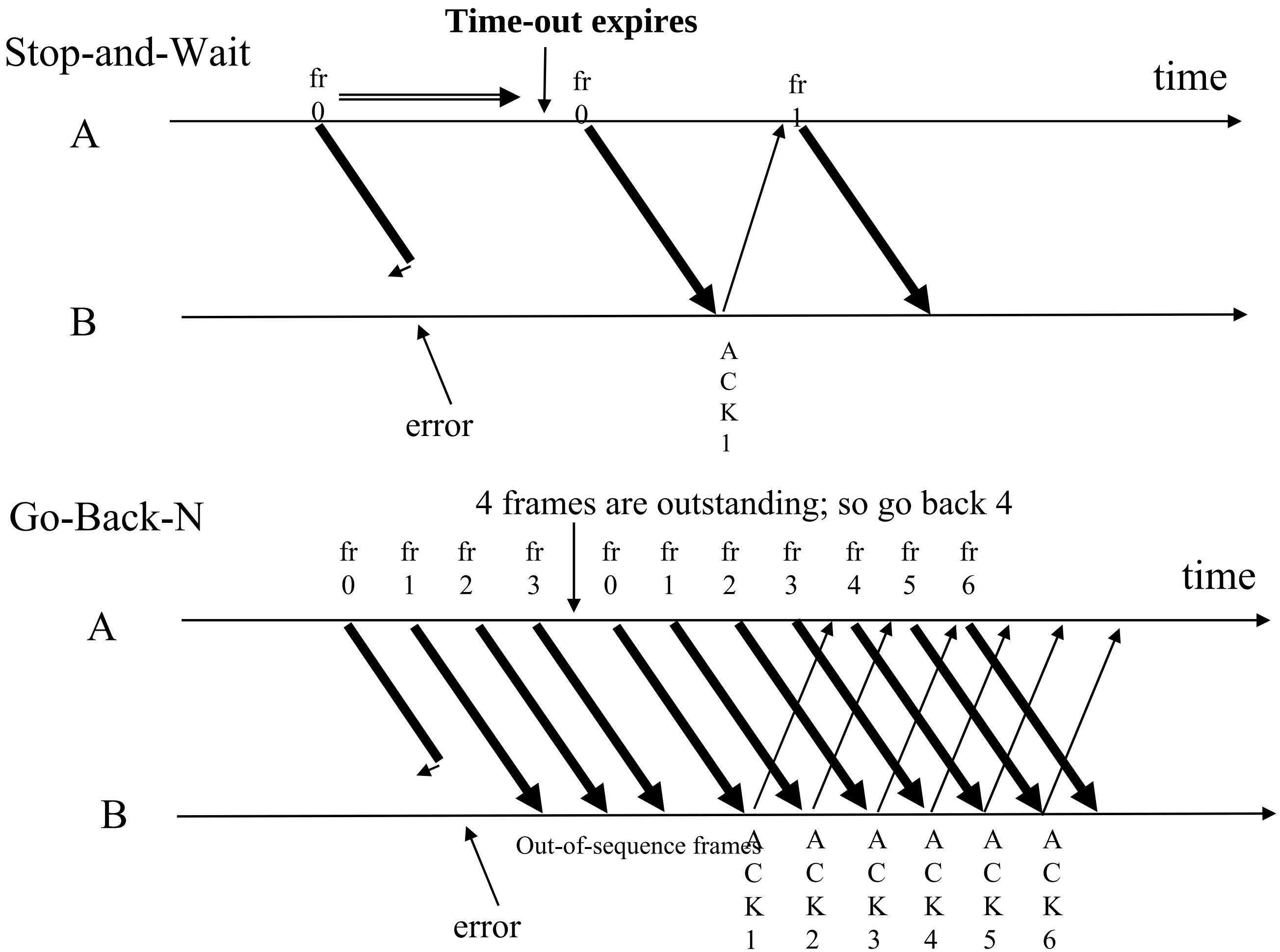
» receiver ignores frame 3 and all subsequent frames

» transmitter eventually reaches its maximum number of outstanding frames

» and forced to *go back N* frames, where $N = W_s$

» and begin retransmitting all packets from frame 3 onwards again

– correspondence between Go-Back-N and Stop-and-Wait protocols :



» loss of transmission time equal to the time-out for Stop-and-Wait

» loss of time corresponding to W_s frames for Go-Back-N

– Go-Back-N protocol depends on ensuring that the oldest frame is eventually delivered correctly

» since the protocol triggers the retransmission of this frame and the subsequent W_s*-1* frames each time the send window is exhausted

» protocol will operate correctly as long as any frame can eventually get through

– this protocol works as long as the transmitter has an unlimited supply of packets that need to be transmitted

– where there are *fewer* than Ws-1 subsequent packets to send

» retransmissions are *not* triggered, since the window is not exhausted

– need to associate a *timer* with every packet

» that can expire to trigger retransmissions

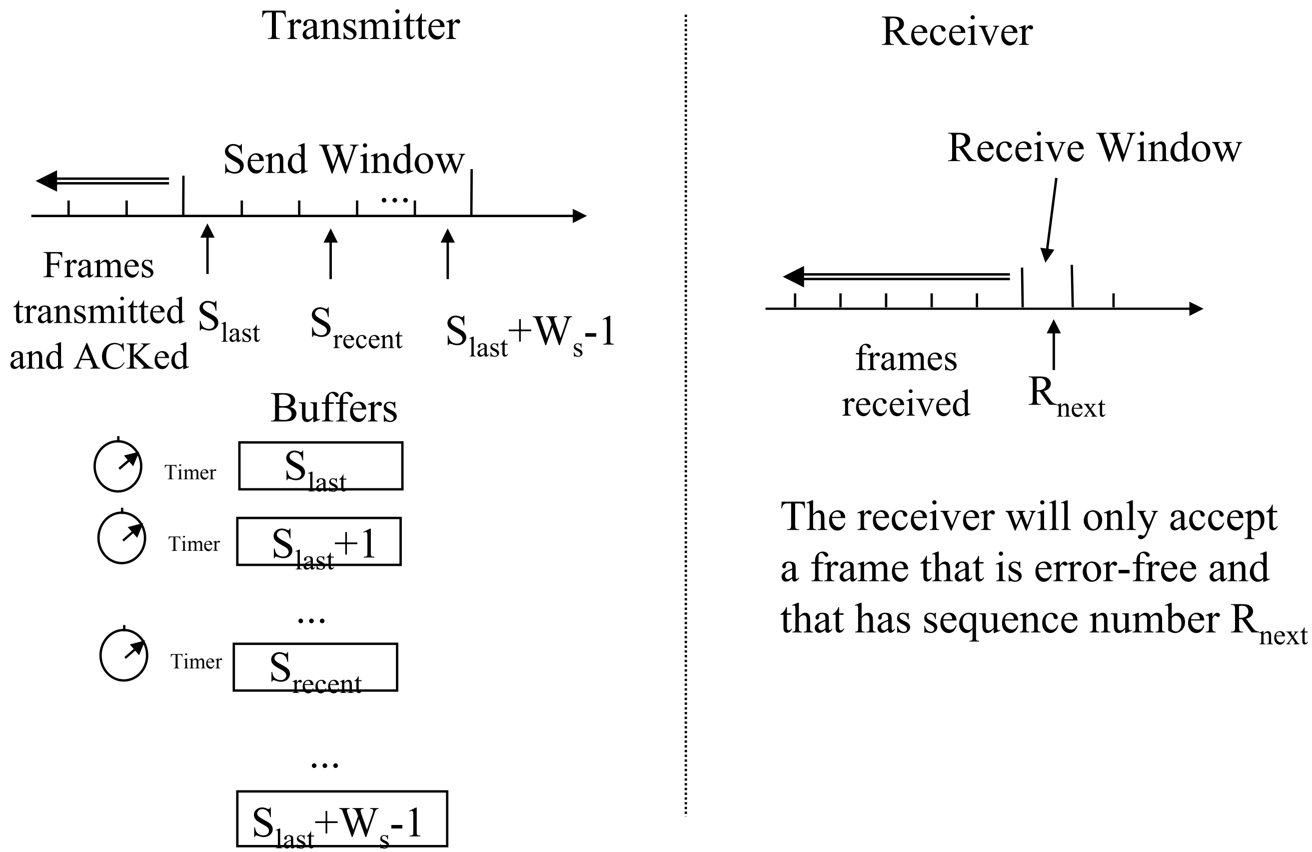
– R_{next} is now the sequence number of the specific frame receiver is looking for

– S_{last} is the oldest unacknowledged frame at the transmitter

– let S_{recent} be the number of the most recently transmitted frame

– the transmitter maintains a list of the frames it is processing

» and must *buffer* all frames sent but not yet acknowledged



» transmitter has a *send window* of sequence numbers : S_{last} to $S_{last} + W_s - 1$

- if S_{recent} reaches upper limit, transmitter must wait for a new acknowledgment

» receiver maintains a *receive window* of size 1 for the next frame R_{next} it expects

– Go-Back-N is a *sliding window* protocol

» if an arriving frame passes the CRC check and has the correct sequence number, R_{next}, it is accepted and R_{next} is incremented

- the receive window *slides forward*

» the receiver sends an acknowledgment containing the incremented R_{next}

- which implicitly acknowledges receipt of all frames prior to R_{next}
- assuming a *wire-like* channel in which packets cannot get re-ordered

» when the transmitter receives an ACK with a value R_{next}, it can assume that all prior frames have been received correctly

- even if it has not received ACKs for all those frames
- because either they got lost or the receiver chose not to transmit them

» upon receiving an ACK with value R_{next}, transmitter updates its value of S_{last} to R_{next} and slides the window forward

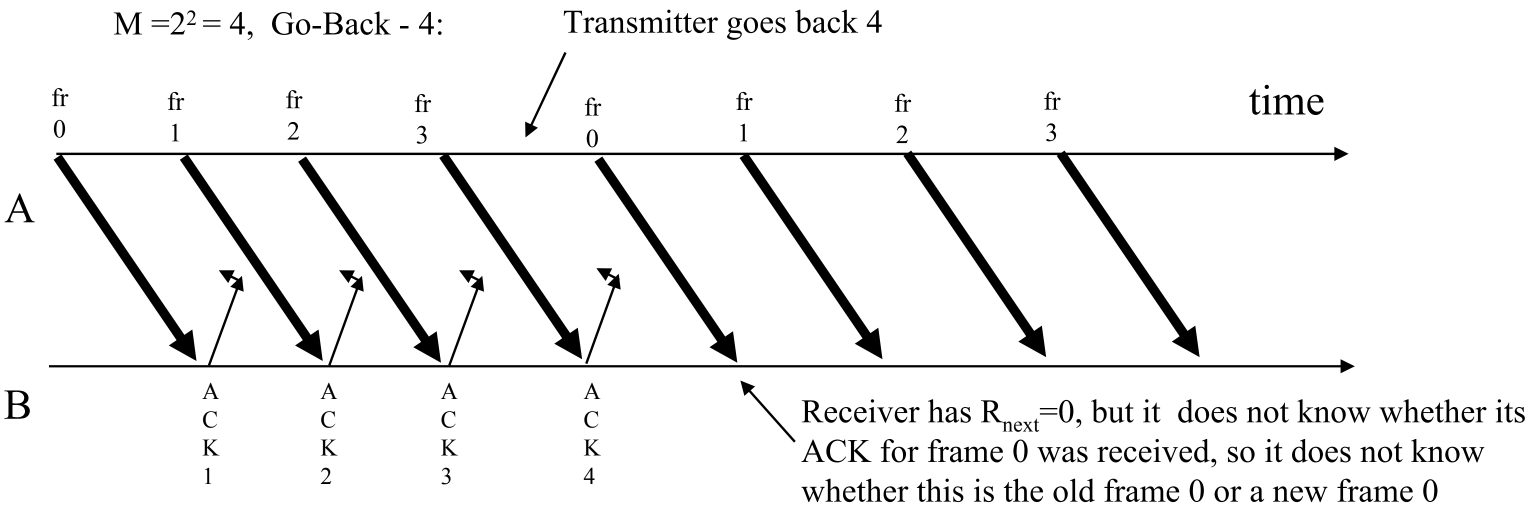
– Sequence numbers versus Window size, W_s

» for *m* bits of sequence number in the frame, 2^m possible sequence numbers

» therefore must be counted modulo 2^m

– receiver must be able to determine *unambiguously* which frame has been received *taking into account the wrapping around* when count reaches 2^m

– consider *m = 2* i.e. 4 sequence numbers, and $W_s = N = 4$:



» transmitter initially sends 4 frames one after the other

» receiver sends 4 corresponding acknowledgments

- but *all of them get lost*

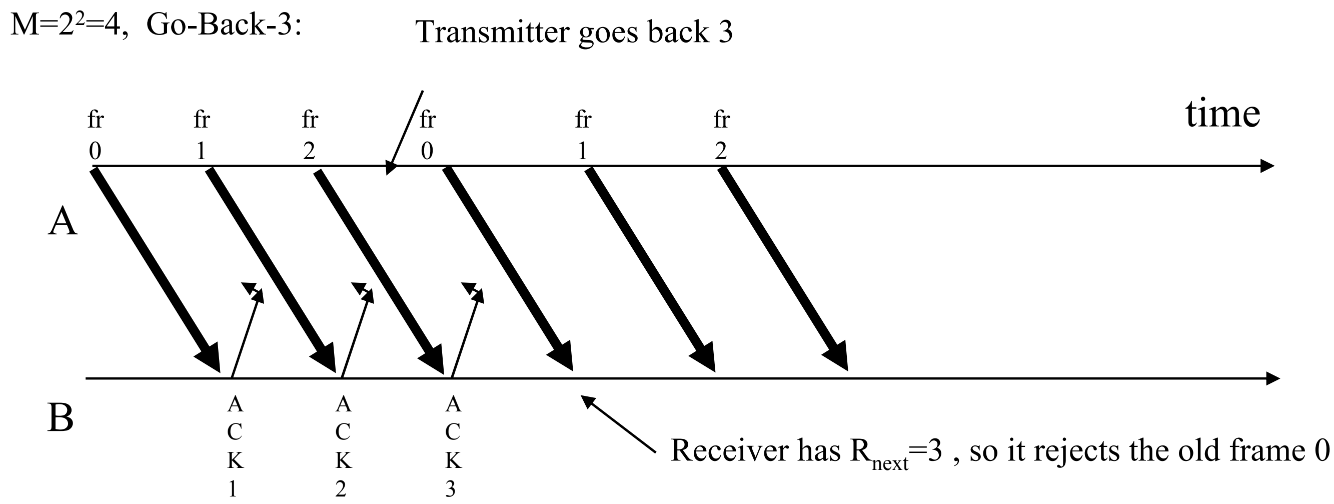
» when transmitter reaches its window size, 4, it goes back 4

- and begins retransmitting frame 0

» when the retransmitted frame 0 reaches the receiver, the receiver has R_{next}=0, so it accepts this as the next valid frame i.e. frame number 5

- it does not know from the frame number whether this is the next frame or a retransmitted frame

– consider *m = 2* and $W_s = N = 3$:



» now when frame 0 is retransmitted, receiver is looking for R_{next}=3

- so knows this frame is an old one

– in general, for m bits of sequence number, with $W_s \le 2^m-1$

» assume current send window is 0 to W_s-1

» suppose frame 0 is received, an acknowledgment sent back but *lost*

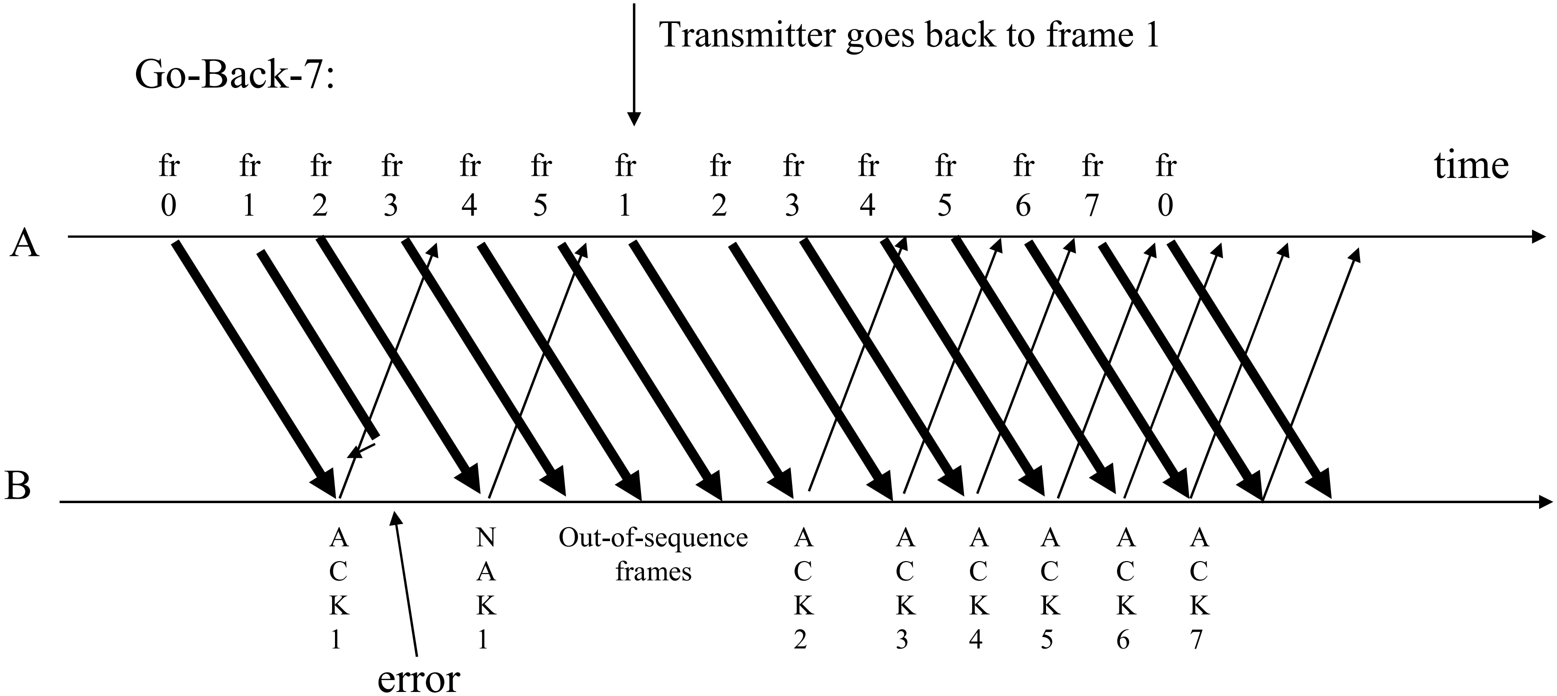
» subsequent frames may also be received and the acknowledgments lost

- but receiver's R_{next} is still incremented so R_{next} is somewhere *in the range 1 to* W_s

» when transmitter resends frame 0, receiver will not be looking for frame 0

- and knows it is an old frame

– performance of Go-Back-N ARQ can be improved by sending a NAK immediately after the first out-of-sequence frame is received :



» the NAK with sequence number R_{next} acknowledges all frames up to R_{next}-1

- and informs the transmitter that an error has been detected in frame R_{next}

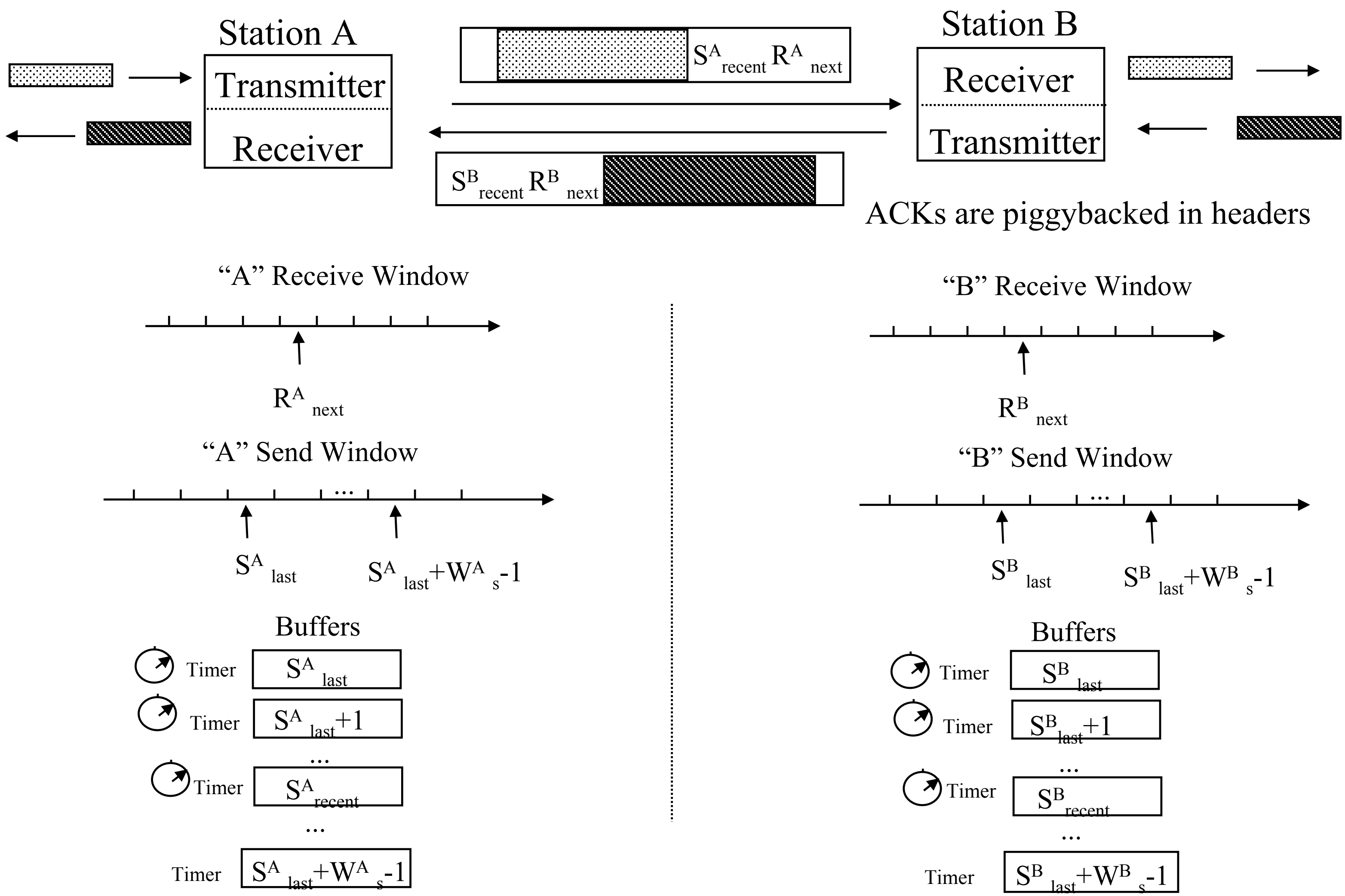
» the receiver now discards all subsequent frames sent

- and instructs the transmitter to go back and retransmit frame R_{next} and all subsequent frames

– in general, the NAK system results in having the transmitter go back less than N frames

– Bidirectional flow : transmitter and receiver functions of the protocol are implemented at both ends

» acknowledgement frames can be *piggybacked* into headers of I-frames



– piggybacking results in significant improvement in use of bandwidth

» separate acknowledge frames can often be avoided

– if no frames are yet ready to be transmitted to piggyback into, receiver can set an *ACK timer*

» that defines the maximum time it will wait for a suitable I-frame

» if it expires, a separate control frame can be sent with the acknowledgment

– a receiver handles out-of sequence packets slightly differently

» a frame that arrives in error is ignored

» subsequent frames that are out of sequence but error free are only discarded after the ACK sequence number i.e. R_{next}, has been extracted

- this allows the local S_{last} to be updated for what it previously transmitted

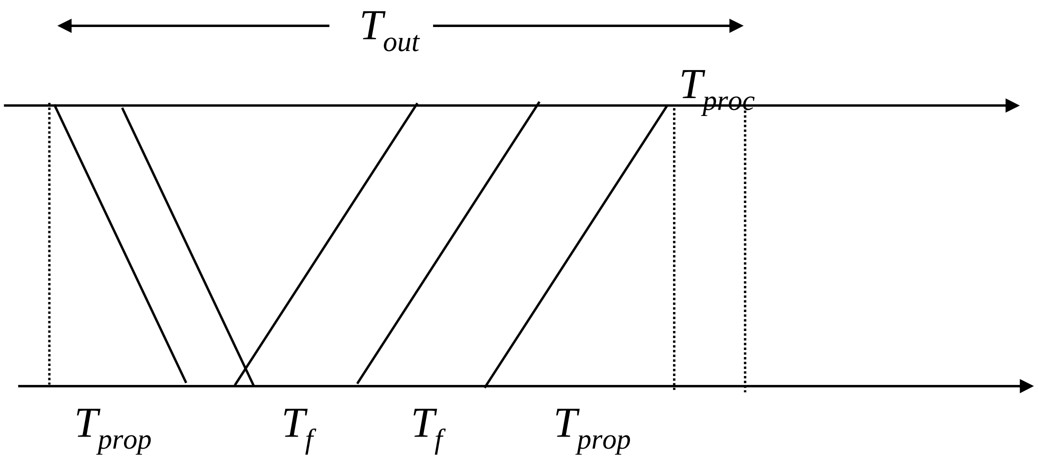
– the time-out value needs to be chosen so that it exceeds the normal time for a frame acknowledgment to be received

» this includes 2 propagation delays, one in each direction

» plus 2 transmission times for the frames used for piggybacking

- the first one might just have been sent too soon to insert an ACK and so missed

» plus some overhead

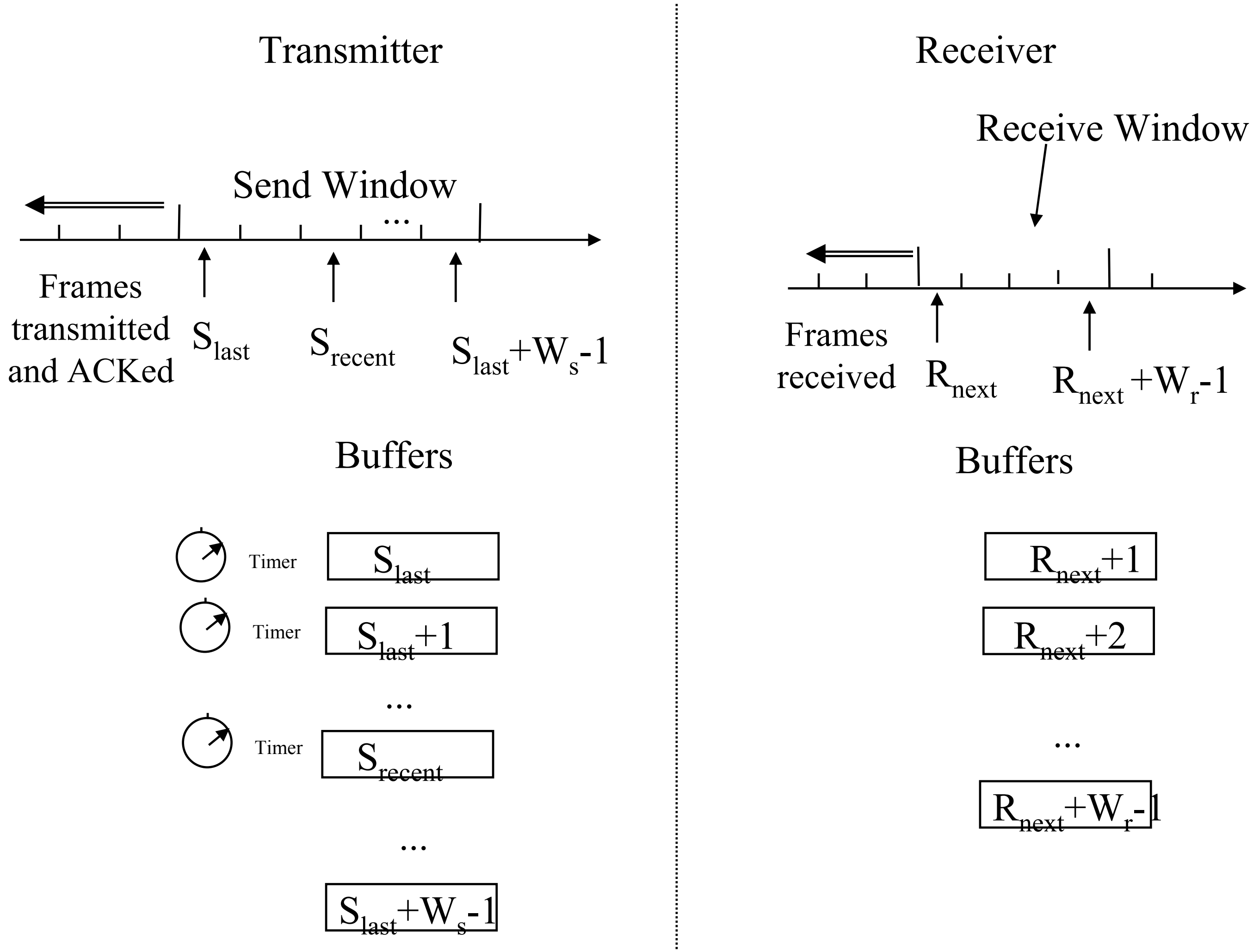


– long piggybacked I-frames can delay receipt of the ACK at the transmitter

» and might exceed the normal time-out, thus triggering extra retransmissions

» if the return I-frame is known to be long, a dedicated control frame can be inserted before it to avoid this

- Selective Repeat ARQ
  - in channels with high error rates, Go-Back-N ARQ is inefficient
    - » because of the need to retransmit *not only* the frame in error but *also* all the subsequent frames
  - *Selective Repeat* modifies Go-Back-N ARQ :
    - » by allowing frames that are out-of-sequence but error free to be accepted by the receiver
    - » and by only retransmitting the individual frames in error
  - extra buffering is required at the receiver to hold the out-of-sequence frames
    - » until the missing frames are received
    - » and the sequence of frames delivered in the correct order
  - the receive buffer now spans the range R_{next} to $R_{next} + W_r - 1$
    - » where W_r is the maximum number of frames the receiver is prepared to accept at once
  - basic objective remains to advance the values of R_{next} and S_{last} by delivery of the oldest outstanding frame



– ACK frames carry R_{next}, the oldest frame not yet received

» the receive window is advanced with error-free receipt of a frame with sequence number R_{next}

– unlike Go-Back-N, the receive window may now advance by several frames at once :

» occurs when one or more of the frames that immediately follow R_{next} have already been received correctly

» and are buffered in the receiver

– all these frames can now be delivered in order to the final destination user

– Retransmission mechanism :

– when a timer expires, only the corresponding frame is retransmitted

– whenever an out-of-sequence frame is received, a NAK is sent back with sequence number R_{next}

» when the transmitter receives such a NAK, it retransmits *that specific frame*

» piggybacking used for bidirectional channels, as before

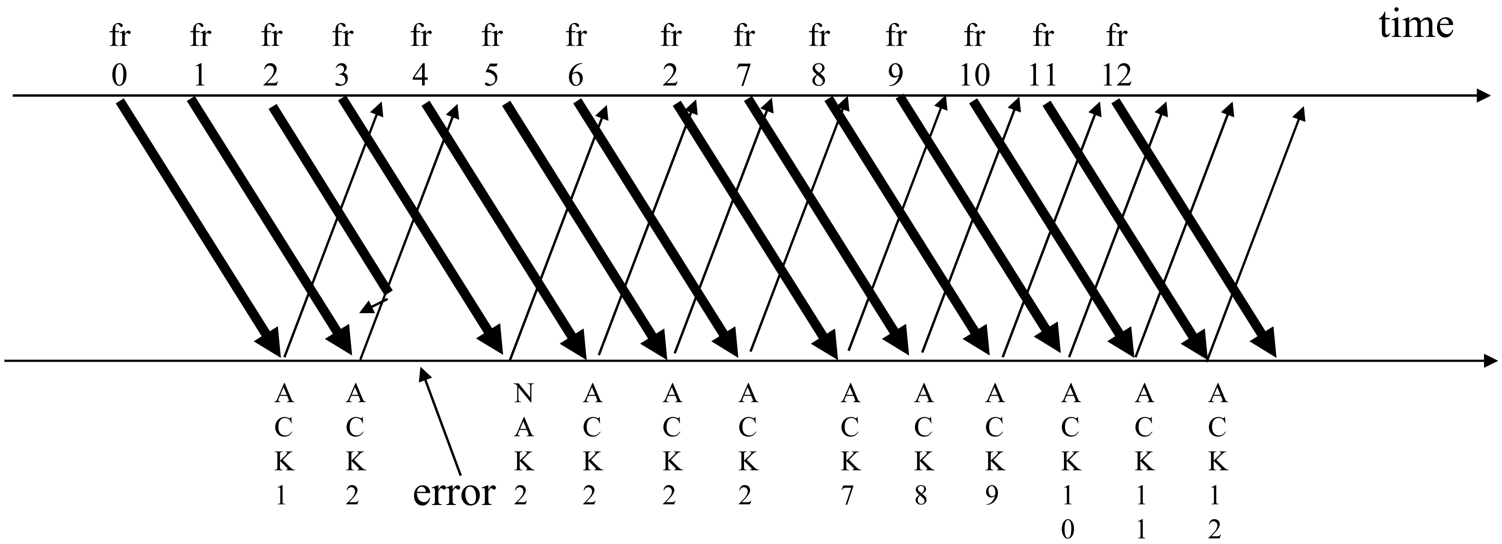
» the ACK frames for subsequent frames continue to hold R_{next}

» only when the frame in error is finally received is the sequence number returned updated

» NAKs are then returned each later frame received in error, one by one

- each one being a request for that specific frame to be retransmitted

– example:



» NAK 2 returned when frame 3 arrives out of sequence

» receiver continues to reply with ACK 2s for subsequent frames

- indicating that it is still waiting for a correct transmission of frame 2

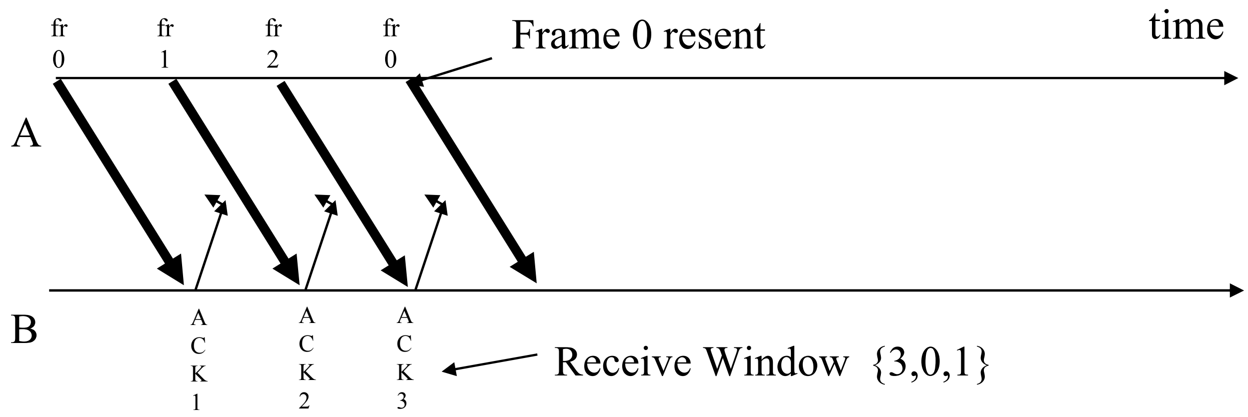
» until the retransmitted frame 2 is successfully received

» then the receive window is moved up to R_{next} = 7, to take account of it having already successfully received frames 3, 4, 5 and 6

- even if the ACKs for the intervening frames did not get back to the transmitter
- ACK 7 implies that all the previous frames have now been received correctly

– Maximum send window size for *m*-bit sequence numbers

– example: m = 2, send and receive window size = 3 :



» initially A transmits frames 0, 1, 2

» all three arrive *correctly* but all the ACKs are *lost*

- R_{next} is incremented to 3 and window advanced

» receiver's window is now ready to accept frames 3, 0, 1

- receiver does not know that A did not get the ACKs
- receiver assumes that frames 3, 4{0}, 5{1} will be transmitted next

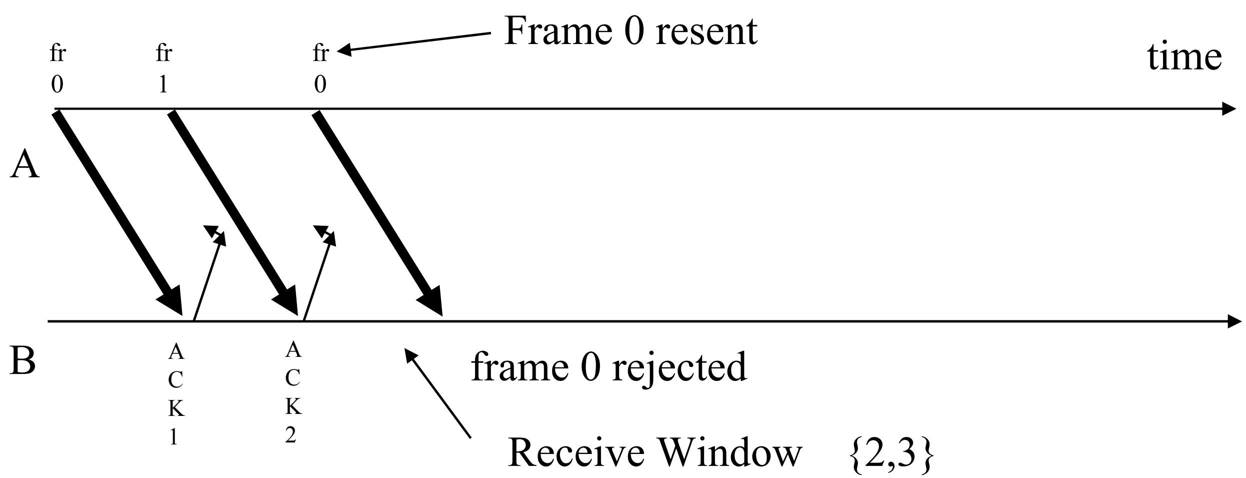
» when A's timer for frame 0 expires, it retransmits frame 0

» upon receiving this frame 0, B now cannot tell whether it is the old retransmitted frame 0 or a new frame with sequence number 0

- possible since frame 3 may have gone missing in transit

– window size = 2^m-1 too large

– example: m = 2, send and receive window size = 2 :



» A transmits frames 0, 1

» both received correctly but both ACKs get lost again

- R_{next} incremented to 2

» receiver now ready to accept frames 2, 3

» when A’s timer for frame 0 expires, it retransmits frame 0

» when B receives this frame 0, it knows that it is a retransmitted old frame 0

- since a new frame 0 cannot be transmitted by A until an ACK 2 has been sent
- B discards this old frame 0 since it has already received it correctly

– in general, suppose window size is $W_s = W_r = 2^{m-1}$

» i.e. half the sequence number space

– suppose initial send and receive windows are both 0 to W_s-1

» suppose frame 0 is received correctly but the ACK for it is lost

» transmitter can transmit subsequent frames up to frame W_s-1

» depending on which frames are received correctly, R_{next} can be anywhere in the range 1 to W_s

- $R_{next} = W_s$ if all the frames transmitted are received correctly

» the end of the receive window, $R_{next} + W_r - 1$, can be anywhere in the range from W_s to $2W_s$-1

- = $2W_s$-1 if all the frames have been received correctly and $R_{next} = W_s$

» the receiver will not receive frame 2Ws until the transmitter has received an acknowledgment for frame 0

» any receipt of frame 0 prior to frame $2W_s$ indicates a retransmission of frame 0

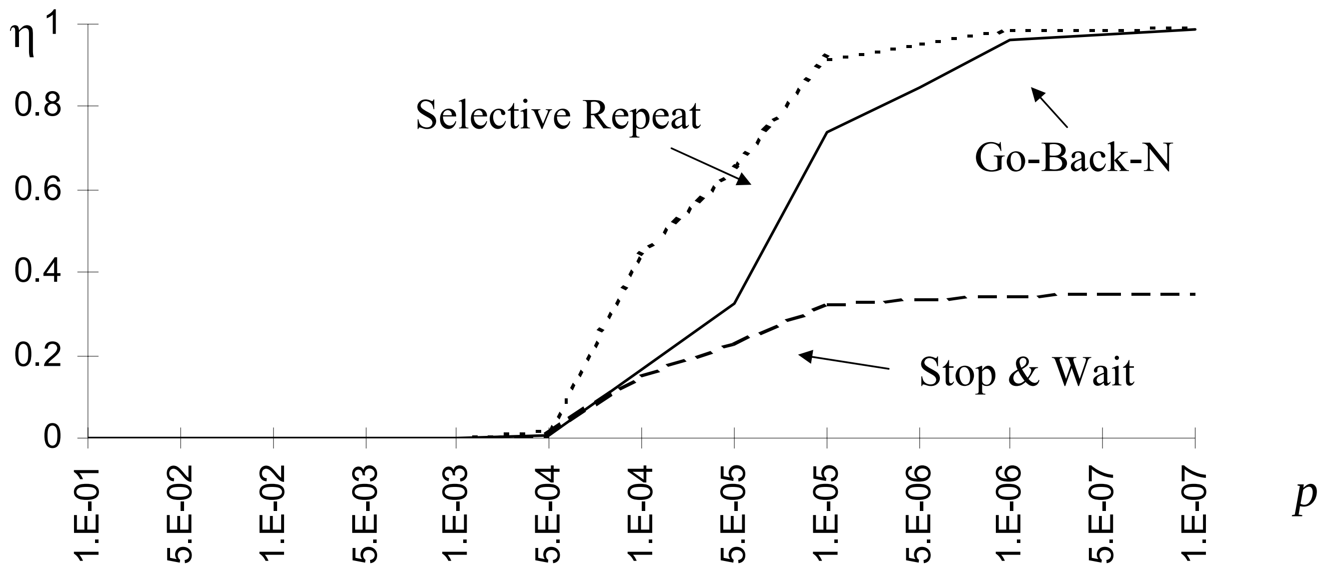
- I.e. frames cannot get more than $2W_s$ ahead

» therefore, $2W_s = 2^m$ indicates the maximum window size before wrap-around

» i.e. $W_s = 2^{m-1}$

- Examples of Selective Repeat ARQ:
  - Transmission Control Protocol (TCP)
    - » slightly more elaborate to deal with a stream of bytes
    - » which the higher level protocol may not immediately send or consume
      - i.e. send and receive windows bigger and need more control pointers
    - » also has to deal with packets arriving out of order
  - Service Specific Connection Oriented Protocol (SSCOP)
    - » originally invented for high-speed satellite links
    - » now used in ATM networks
    - » both have a large delay-bandwidth product which require an efficient transmission protocol

- Transmission Efficiency of ARQ protocols
  - example: 1024 byte frames, 1.5Mbps channel, 5ms delay (Leon-Garcia)
  - random bit errors with probability *p*, efficiency η :



» efficiency of Stop-and-Wait always less than 35%

» Go-Back-N has efficiencies comparable to Selective Repeat for *p* less than 10^{-6}

- but deteriorates to the performance of Stop-and-Wait when p reaches 5×10^{-5}

» Selective Repeat also deteriorates as p becomes larger than 10^{-4}

- at this rate, probability of frame error is $1-(1-10^{-4})^{8192} \approx 0.56$

– optimum frame length

» as frame size increases, impact of delay-bandwidth product is reduced

» increasing frame size also increases probability of frame transmission error

– example: $p = 10^{-4}$