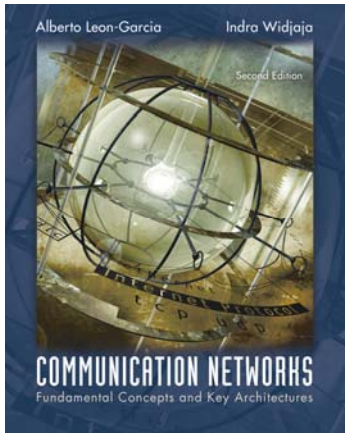
Chapter 5
Peer-to-Peer Protocols and Data Link Layer



PART I: Peer-to-Peer Protocols

Peer-to-Peer Protocols and Service Models

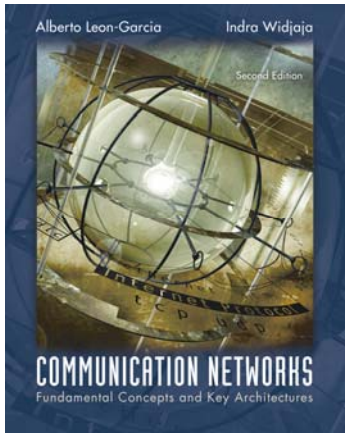
ARQ Protocols and Reliable Data Transfer

Flow Control

Timing Recovery

TCP Reliable Stream Service & Flow Control

Chapter 5
Peer-to-Peer Protocols and Data Link Layer



PART II: Data Link Controls

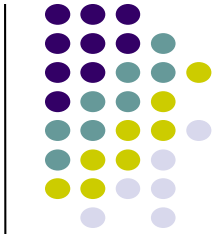
Framing

Point-to-Point Protocol

High-Level Data Link Control

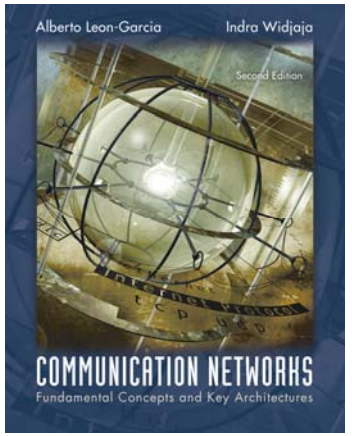
Link Sharing Using Statistical Multiplexing

Chapter Overview



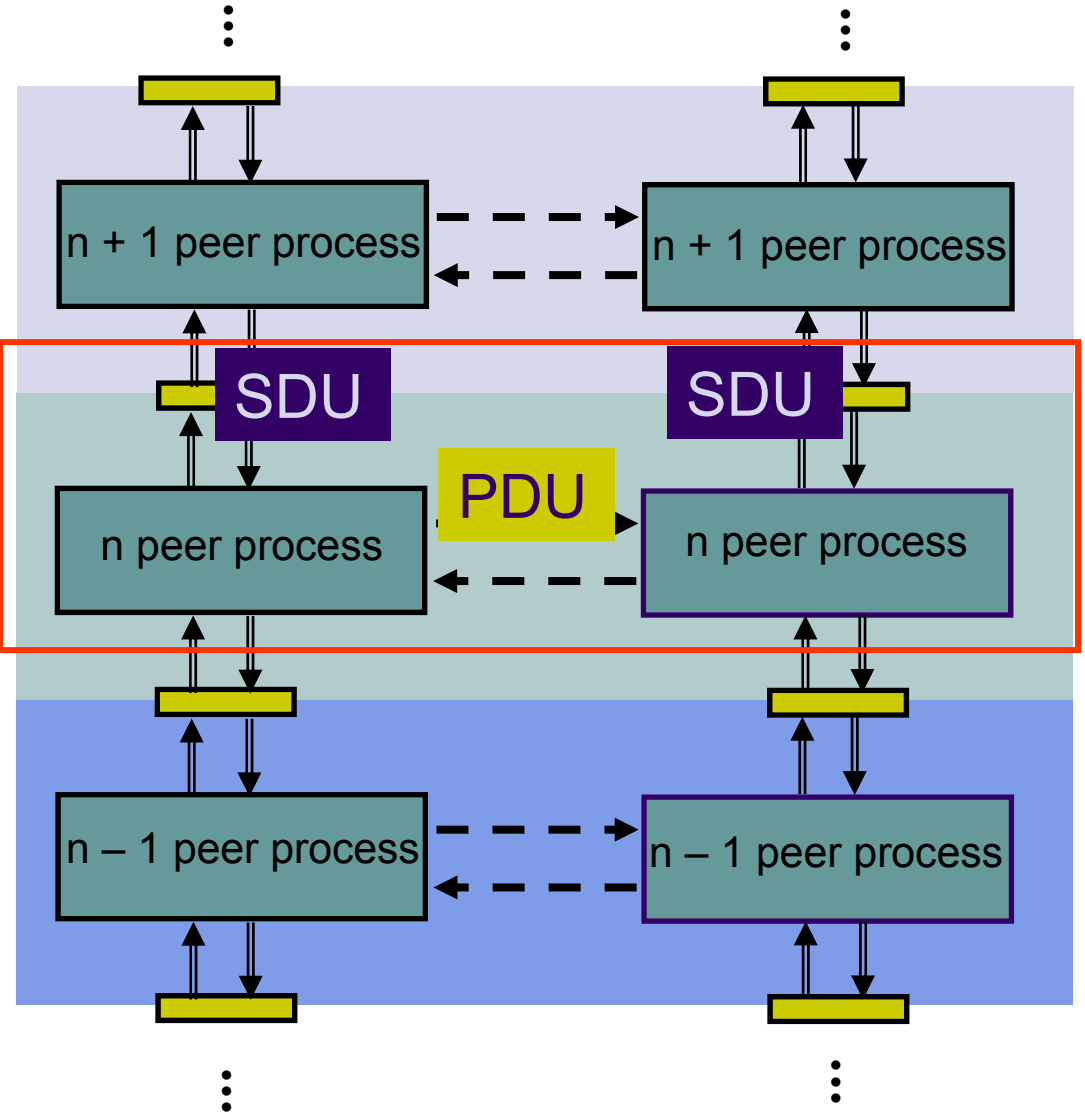
- Peer-to-Peer protocols: many protocols involve the interaction between two peers
  - Service Models are discussed & examples given
  - Detailed discussion of ARQ provides example of development of peer-to-peer protocols
  - Flow control, TCP reliable stream, and timing recovery
- Data Link Layer
  - Framing
  - PPP & HDLC protocols
  - Statistical multiplexing for link sharing

Chapter 5
Peer-to-Peer Protocols and Data Link Layer



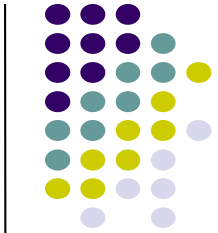
Peer-to-Peer Protocols and Service Models

Peer-to-Peer Protocols



- *Peer-to-Peer processes* execute layer-n protocol to provide service to layer-(n+1)
- Layer-(n+1) peer calls layer-n and passes Service Data Units (SDUs) for transfer
- Layer-n peers exchange Protocol Data Units (PDUs) to effect transfer
- Layer-n delivers SDUs to destination layer-(n+1) peer

Service Models

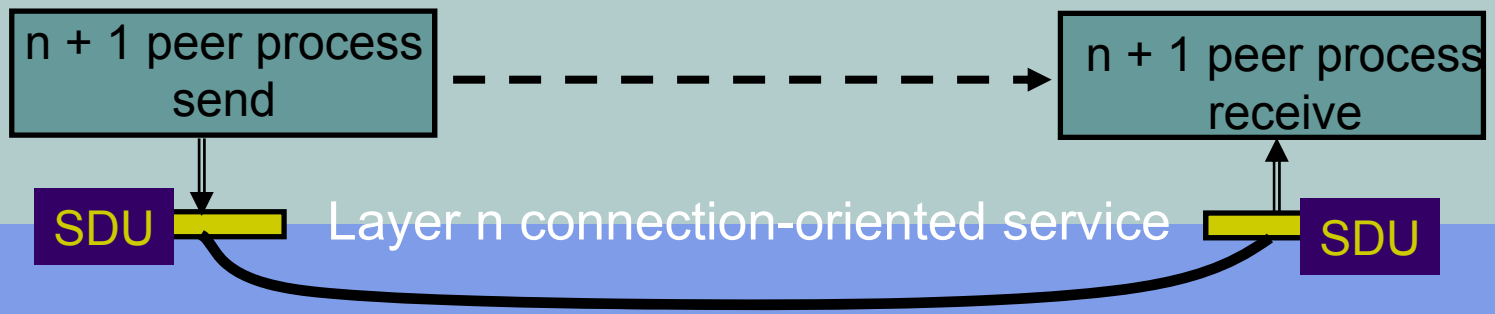


- The *service model* specifies the information transfer service layer-n provides to layer-(n+1)
- The most important distinction is whether the service is:
  - Connection-oriented
  - Connectionless
- Service model possible features:
  - Arbitrary message size or structure
  - Sequencing and Reliability
  - Timing, Pacing, and Flow control
  - Multiplexing
  - Privacy, integrity, and authentication

Connection-Oriented Transfer Service



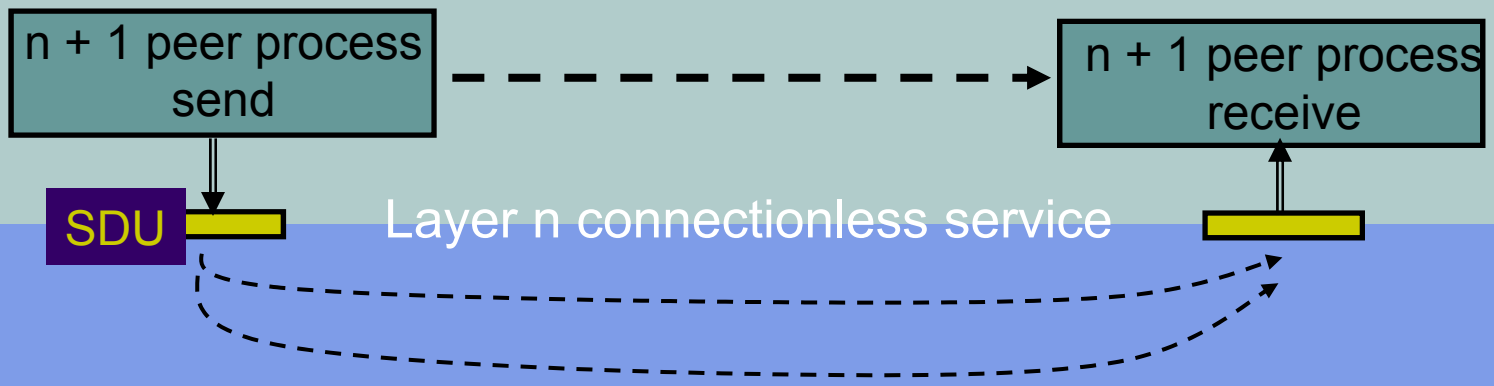
- Connection Establishment
  - Connection must be established between layer-(n+1) peers
  - Layer-n protocol must: Set initial parameters, e.g. sequence numbers; and Allocate resources, e.g. buffers
- Message transfer phase
  - Exchange of SDUs
- Disconnect phase
- Example: TCP, PPP



Connectionless Transfer Service



- No Connection setup, simply send SDU
- Each message send independently
- Must provide all address information per message
- Simple & quick
- Example: UDP, IP



Message Size and Structure



- What message size and structure will a service model accept?
  - Different services impose restrictions on size & structure of data it will transfer
  - Single bit? Block of bytes? Byte stream?
  - Ex: Transfer of voice mail = 1 long message
  - Ex: Transfer of voice call = byte stream

1 voice mail= 1 message = entire sequence of speech samples

(a)



1 call = sequence of 1-byte messages

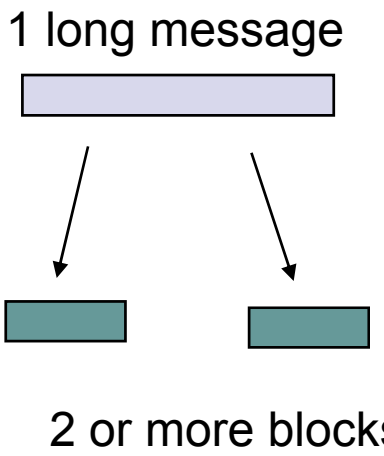
(b)

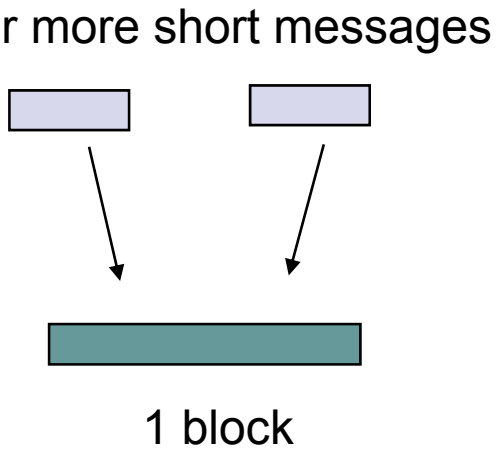


Segmentation & Blocking

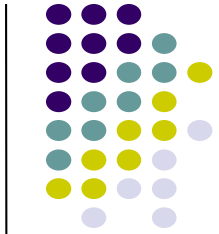


- To accommodate arbitrary message size, a layer may have to deal with messages that are too long or too short for its protocol
- *Segmentation & Reassembly*: a layer breaks long messages into smaller blocks and reassembles these at the destination
- *Blocking & Unblocking*: a layer combines small messages into bigger blocks prior to transfer



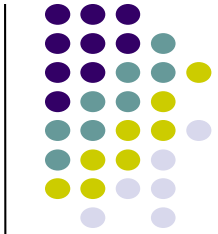


Reliability & Sequencing



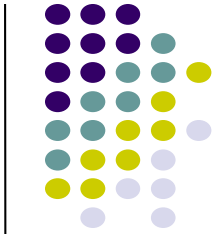
- *Reliability*: Are messages or information stream delivered error-free and without loss or duplication?
- *Sequencing*: Are messages or information stream delivered in order?
- *ARQ protocols* combine error detection, retransmission, and sequence numbering to provide reliability & sequencing
- Examples: TCP and HDLC

Pacing and Flow Control



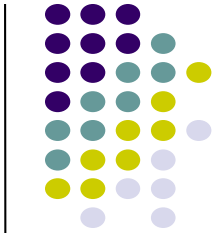
- Messages can be lost if receiving system does not have sufficient buffering to store arriving messages
- If destination layer-(n+1) does not retrieve its information fast enough, destination layer-n buffers may overflow
- *Pacing & Flow Control* provide backpressure mechanisms that control transfer according to availability of buffers at the destination
- Examples: TCP and HDLC

Timing



- Applications involving voice and video generate units of information that are related temporally
- Destination application must reconstruct temporal relation in voice/video units
- Network transfer introduces delay & jitter
- *Timing Recovery* protocols use *timestamps* & *sequence numbering* to control the delay & jitter in delivered information
- Examples: RTP & associated protocols in Voice over IP

Multiplexing



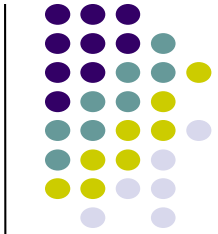
- *Multiplexing* enables multiple layer-(n+1) users to share a layer-n service
- A multiplexing tag is required to identify specific users at the destination
- Examples: UDP, IP

Privacy, Integrity, & Authentication



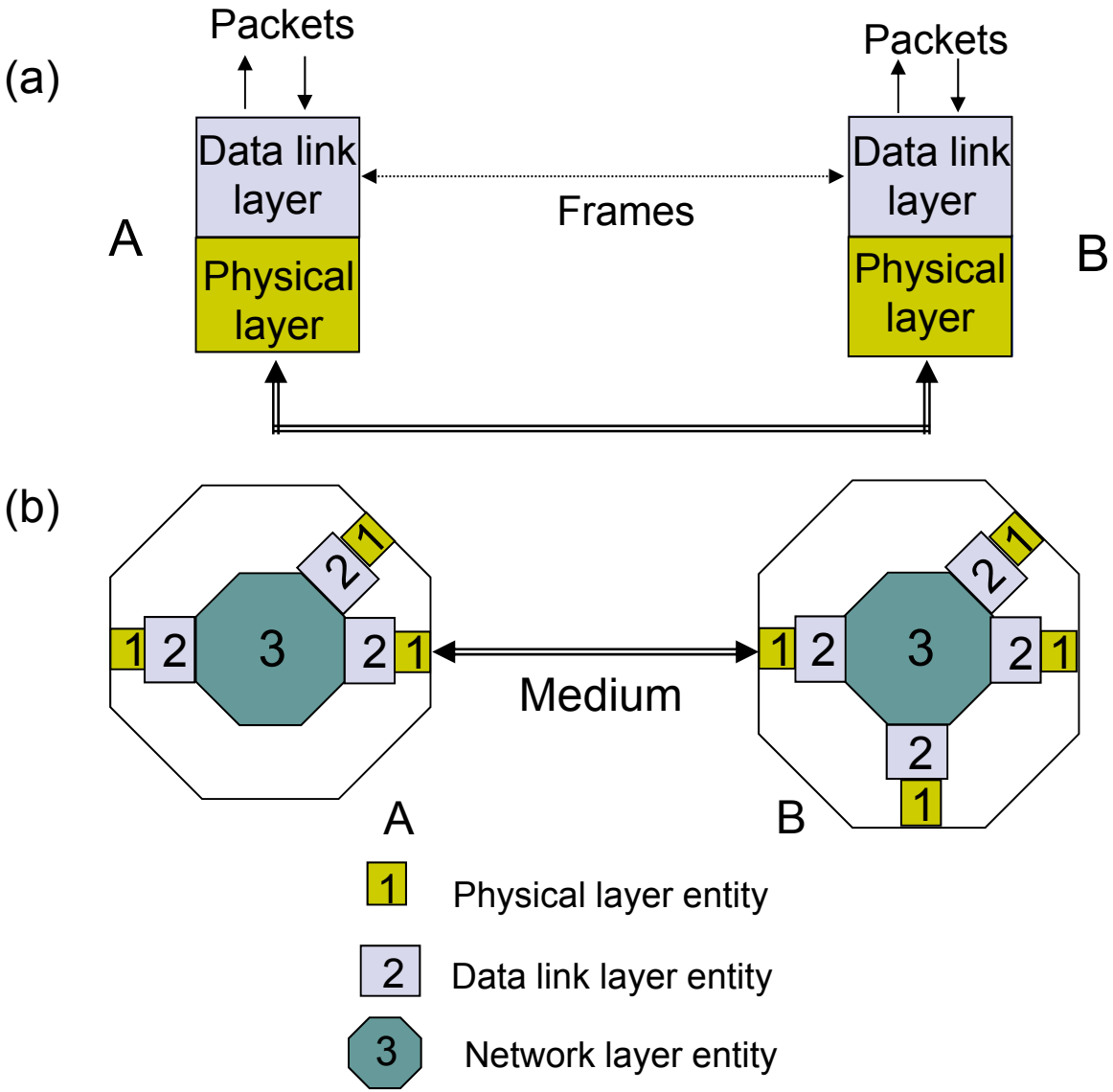
- *Privacy*: ensuring that information transferred cannot be read by others
- *Integrity*: ensuring that information is not altered during transfer
- *Authentication*: verifying that sender and/or receiver are who they claim to be
- *Security protocols* provide these services and are discussed in Chapter 11
- Examples: IPSec, SSL

End-to-End vs. Hop-by-Hop



- A service feature can be provided by implementing a protocol
  - end-to-end across the network
  - across every hop in the network
- Example:
  - Perform error control at every hop in the network or only between the source and destination?
  - Perform flow control between every hop in the network or only between source & destination?
- We next consider the tradeoffs between the two approaches

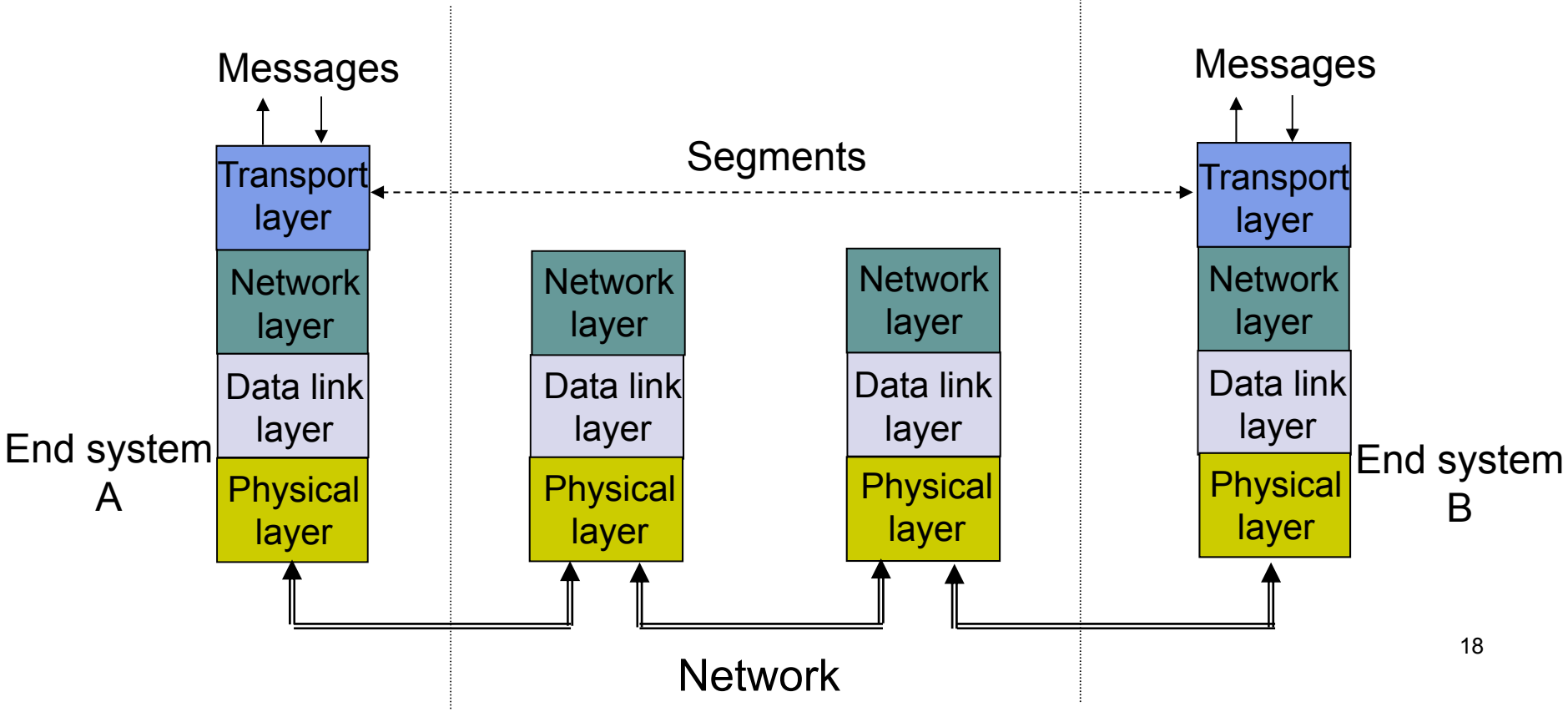
Error control in Data Link Layer



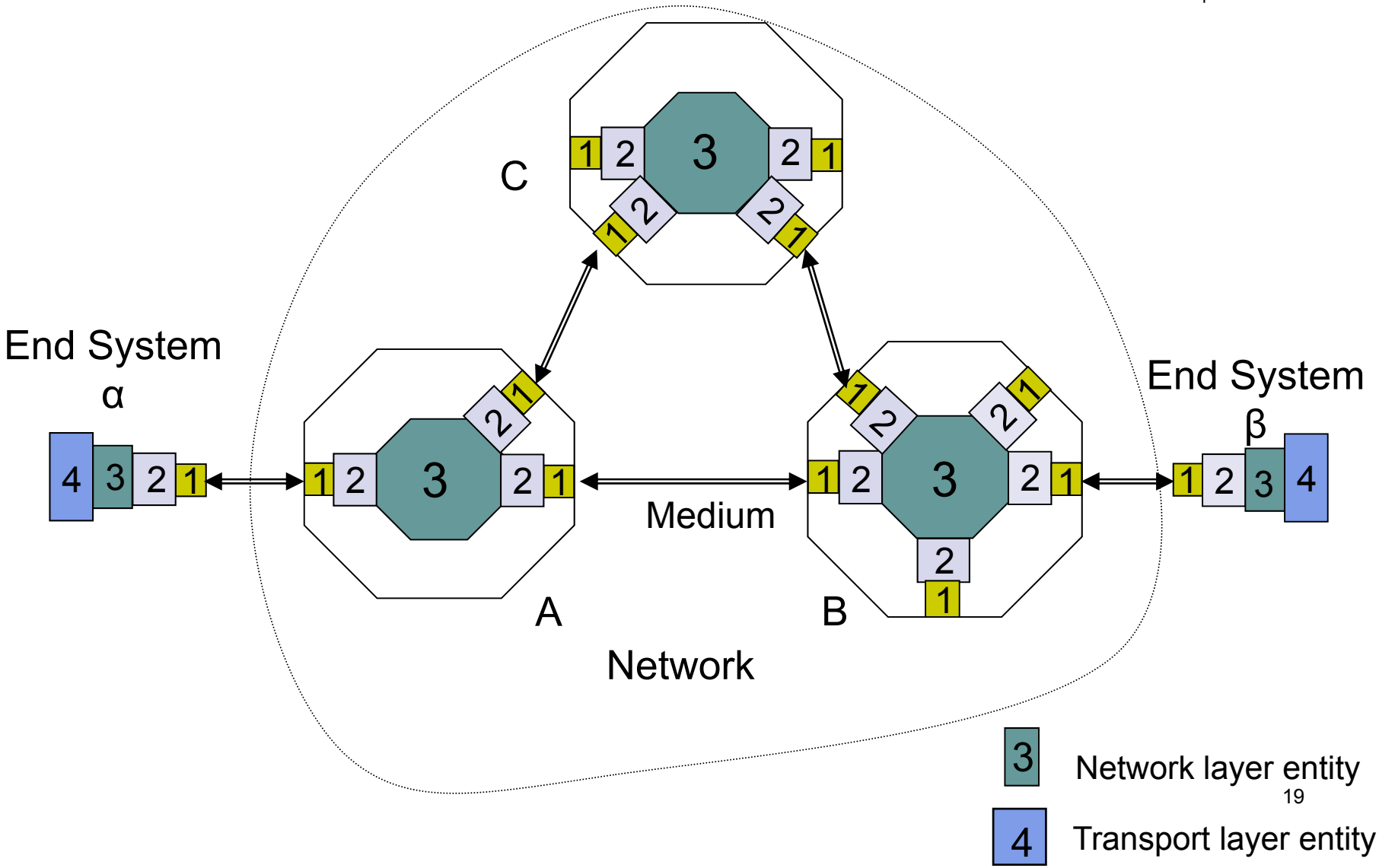
- Data Link operates over wire-like, directly-connected systems
- Frames can be corrupted or lost, but arrive in order
- Data link performs error-checking & retransmission
- Ensures error-free packet transfer between two systems

Error Control in Transport Layer

- Transport layer protocol (e.g. TCP) sends segments across network and performs end-to-end error checking & retransmission
- Underlying network is assumed to be unreliable

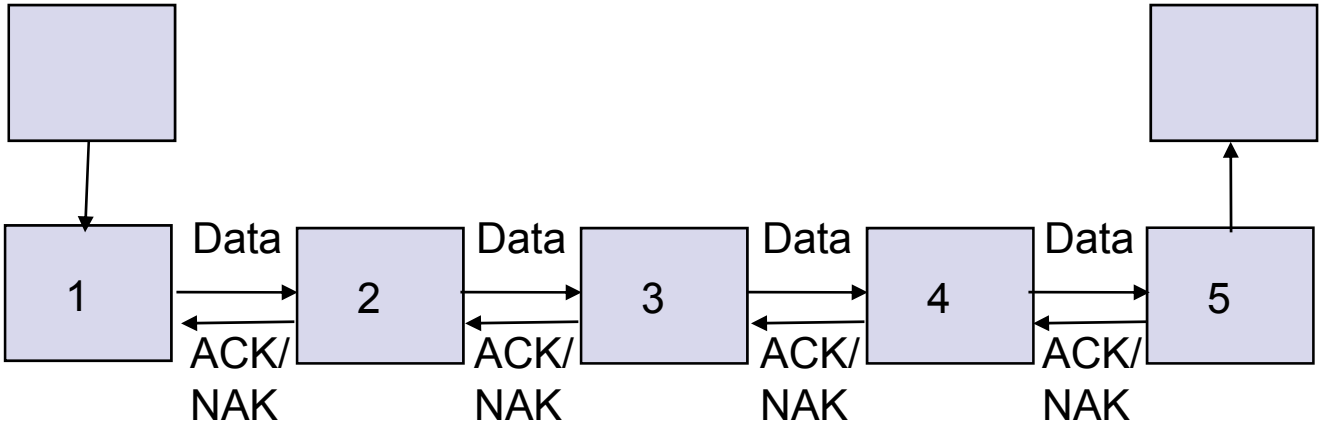


- Segments can experience long delays, can be lost, or arrive out-of-order because packets can follow different paths across network
- End-to-end error control protocol more difficult



End-to-End Approach Preferred

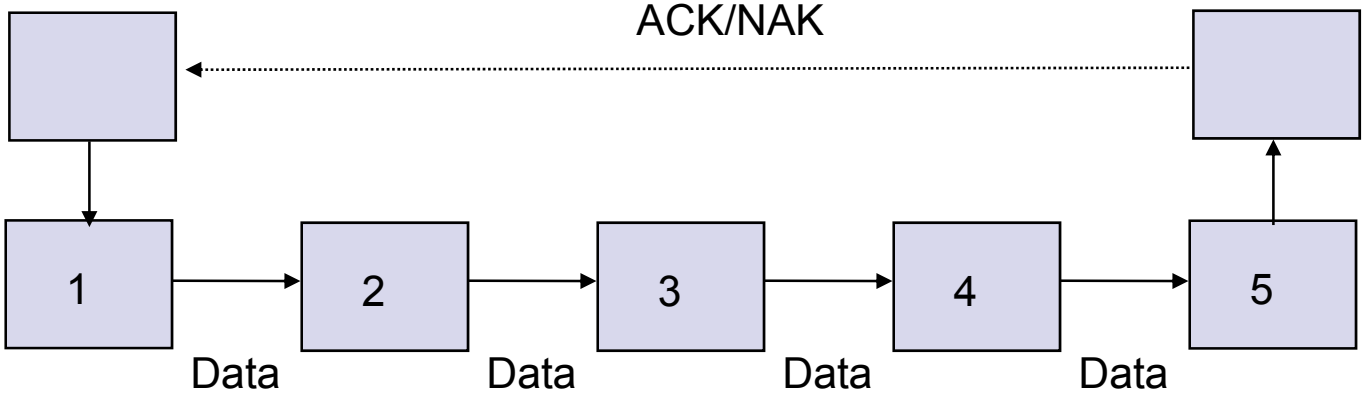
Hop-by-hop



Hop-by-hop cannot ensure E2E correctness

Faster recovery

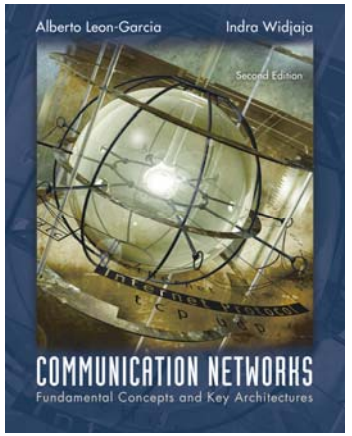
End-to-end

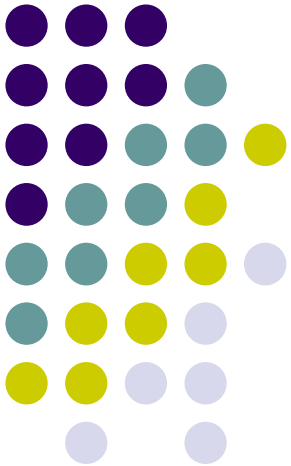


Simple inside the network

More scalable if complexity at the edge

Chapter 5
Peer-to-Peer Protocols and Data Link Layer





ARQ Protocols and Reliable Data Transfer

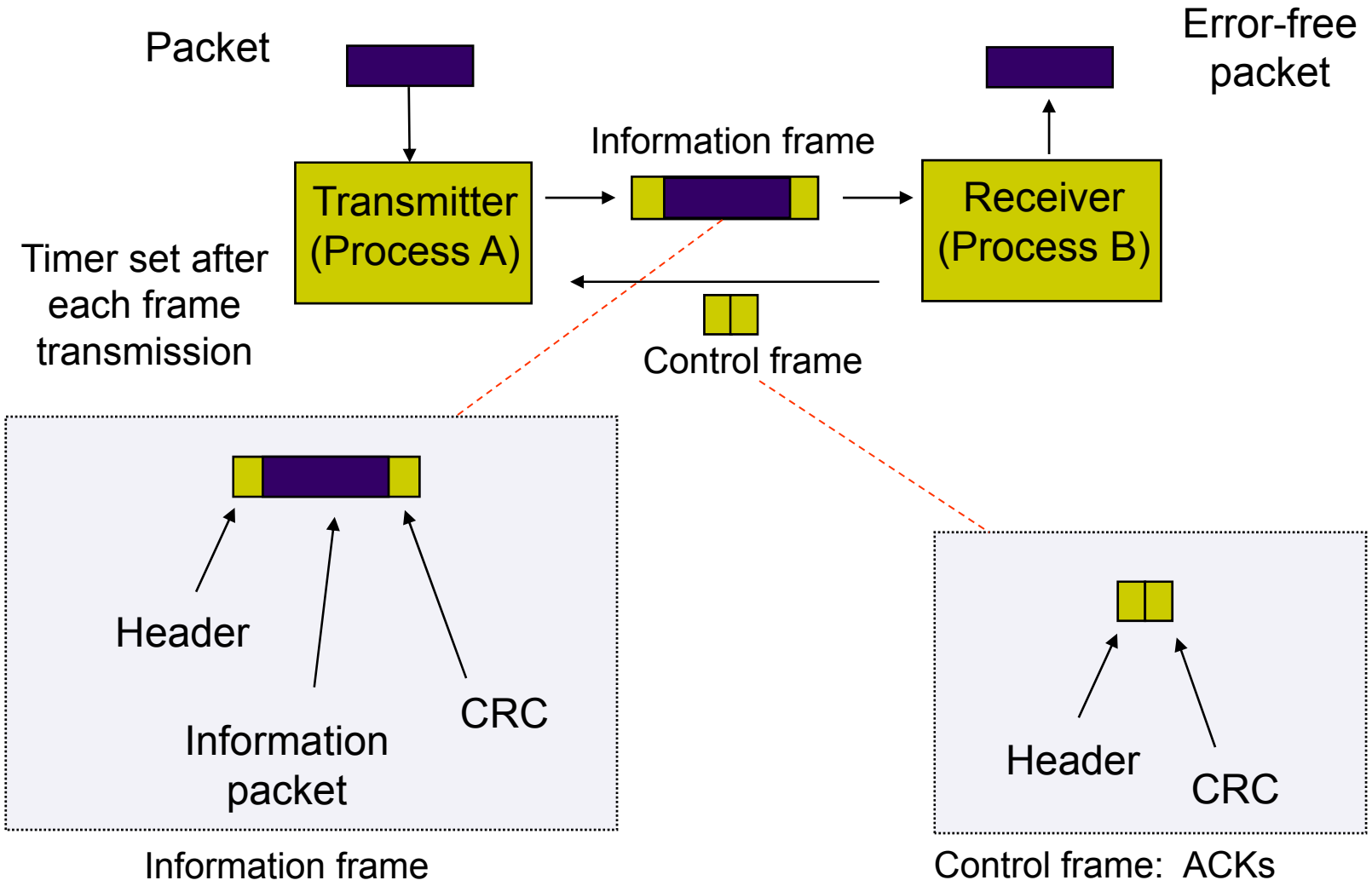
Automatic Repeat Request (ARQ)



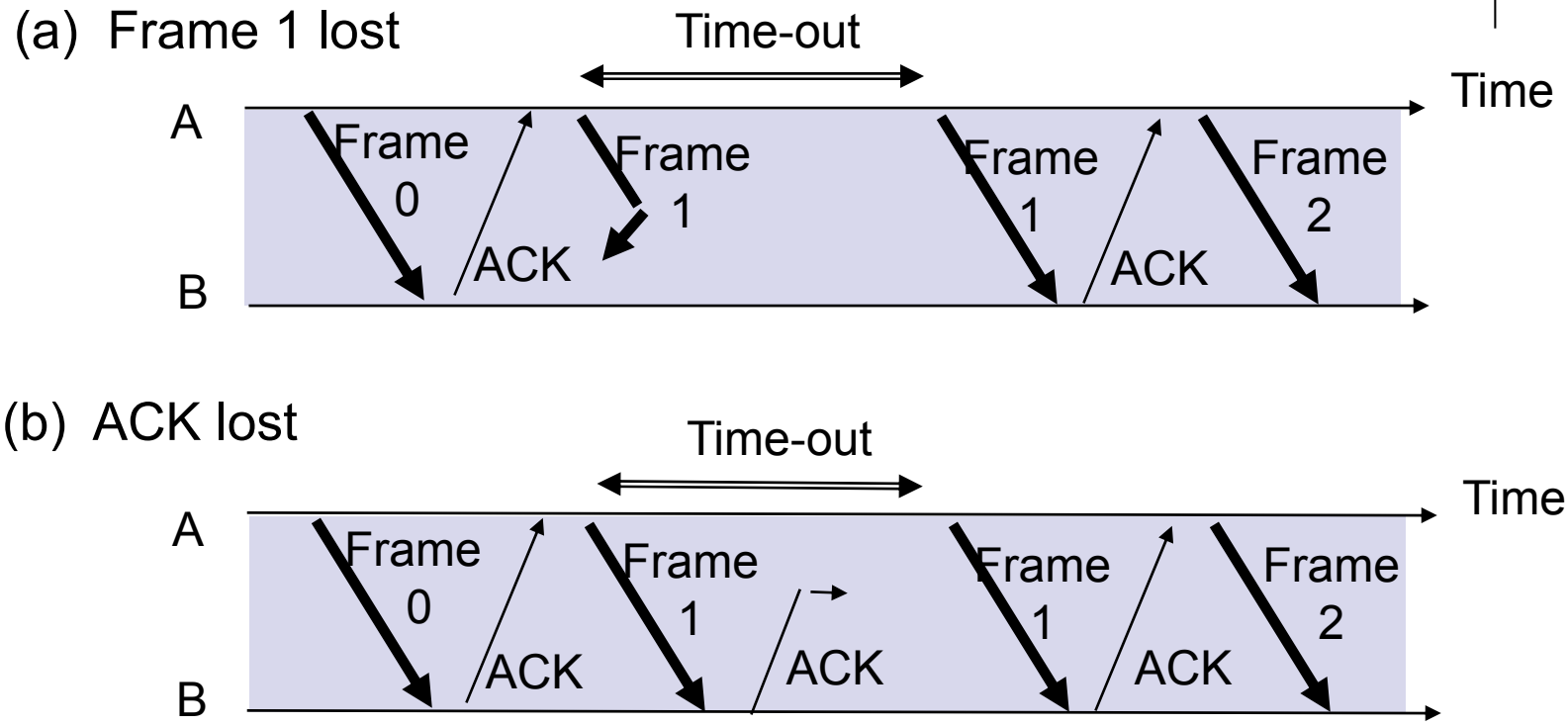
- *Purpose*: to ensure a sequence of information packets is delivered in order and without errors or duplications despite transmission errors & losses
- We will look at:
  - Stop-and-Wait ARQ
  - Go-Back N ARQ
  - Selective Repeat ARQ
- Basic elements of ARQ:
  - *Error-detecting code* with high error coverage
  - *ACKs* (positive acknowledgments
  - *NAKs* (negative acknowlegments)
  - *Timeout mechanism*

Stop-and-Wait ARQ

Transmit a frame, wait for ACK



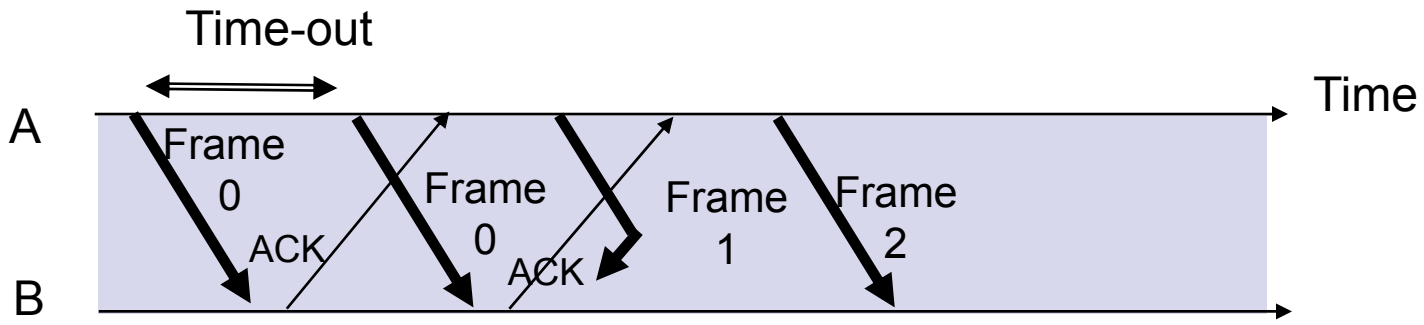
Need for Sequence Numbers



- In cases (a) & (b) the transmitting station A acts the same way
- But in case (b) the receiving station B accepts frame 1 twice
- Question: How is the receiver to know the second frame is also frame 1?
- Answer: ***Add frame sequence number in header***
- S_{last} is sequence number of most recent transmitted frame

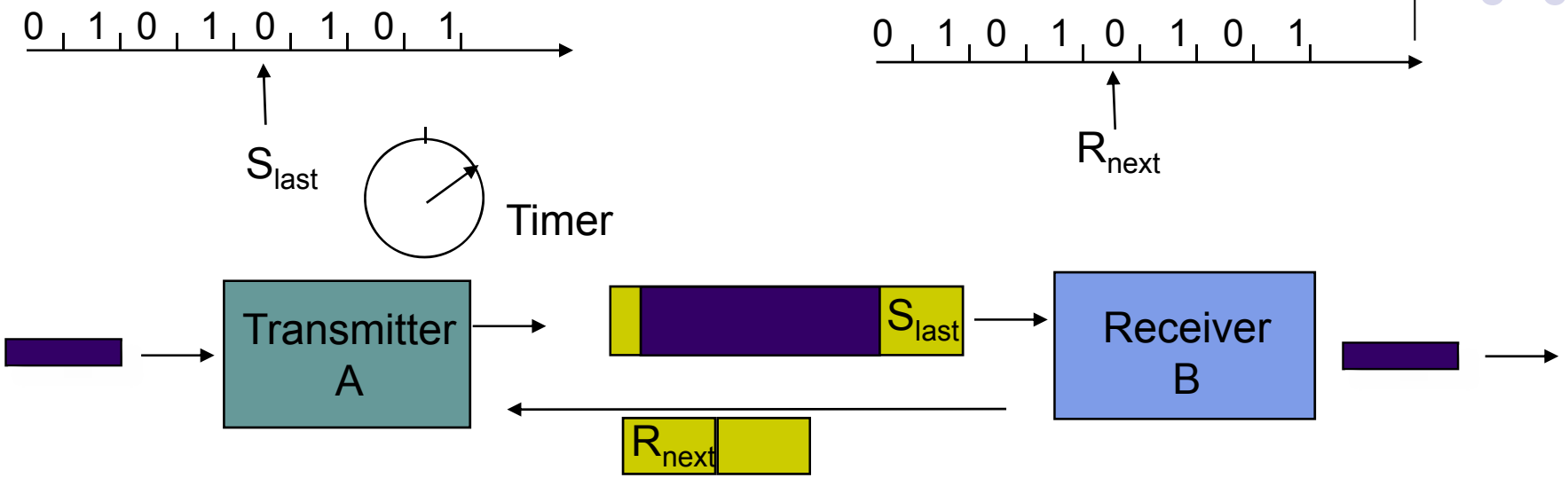
Sequence Numbers

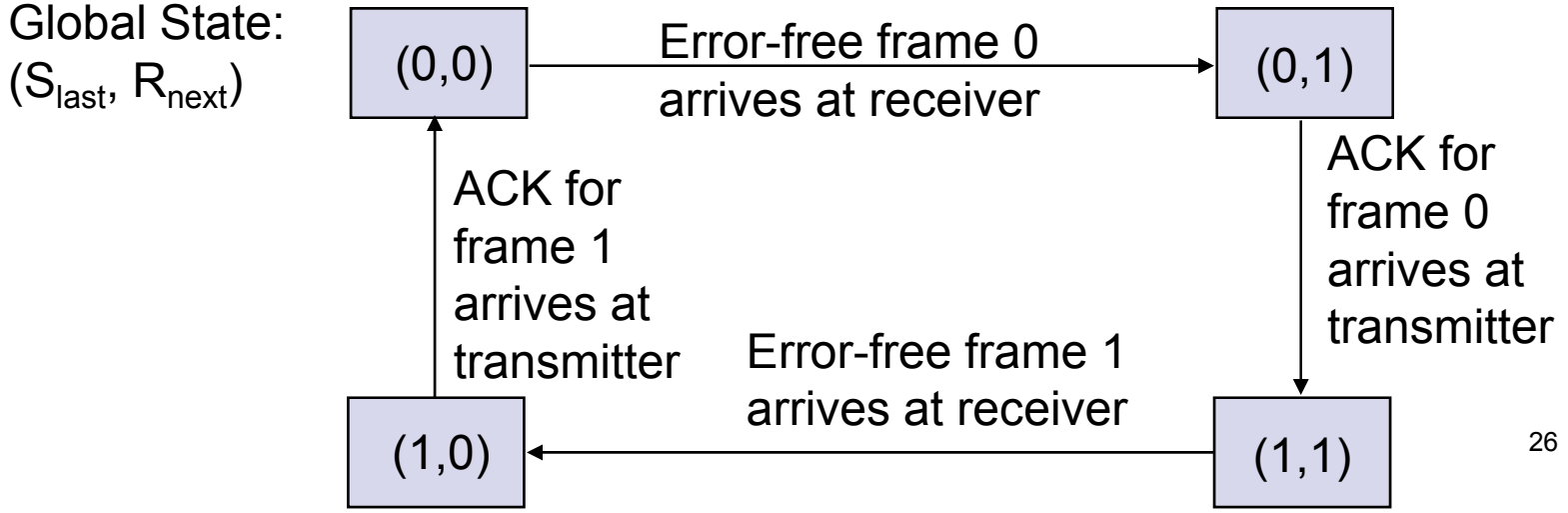
(c) Premature Time-out



- The transmitting station A misinterprets duplicate ACKs
- Incorrectly assumes second ACK acknowledges Frame 1
- Question: How is the receiver to know second ACK is for frame 0?
- Answer: ***Add frame sequence number in ACK header***
- R_{next} is sequence number of next frame expected by the receiver
- Implicitly acknowledges receipt of all prior frames

1-Bit Sequence Numbering Suffices





Stop-and-Wait ARQ



Transmitter

Ready state

- Await request from higher layer for packet transfer
- When request arrives, transmit frame with updated S_{last} and CRC
- Go to Wait State

Wait state

- Wait for ACK or timer to expire; block requests from higher layer
- If timeout expires
  - retransmit frame and reset timer
- If ACK received:
  - If sequence number is incorrect or if errors detected: ignore ACK
  - If sequence number is correct ($R_{next} = S_{last} + 1$): accept frame, go to Ready state

Receiver

Always in Ready State

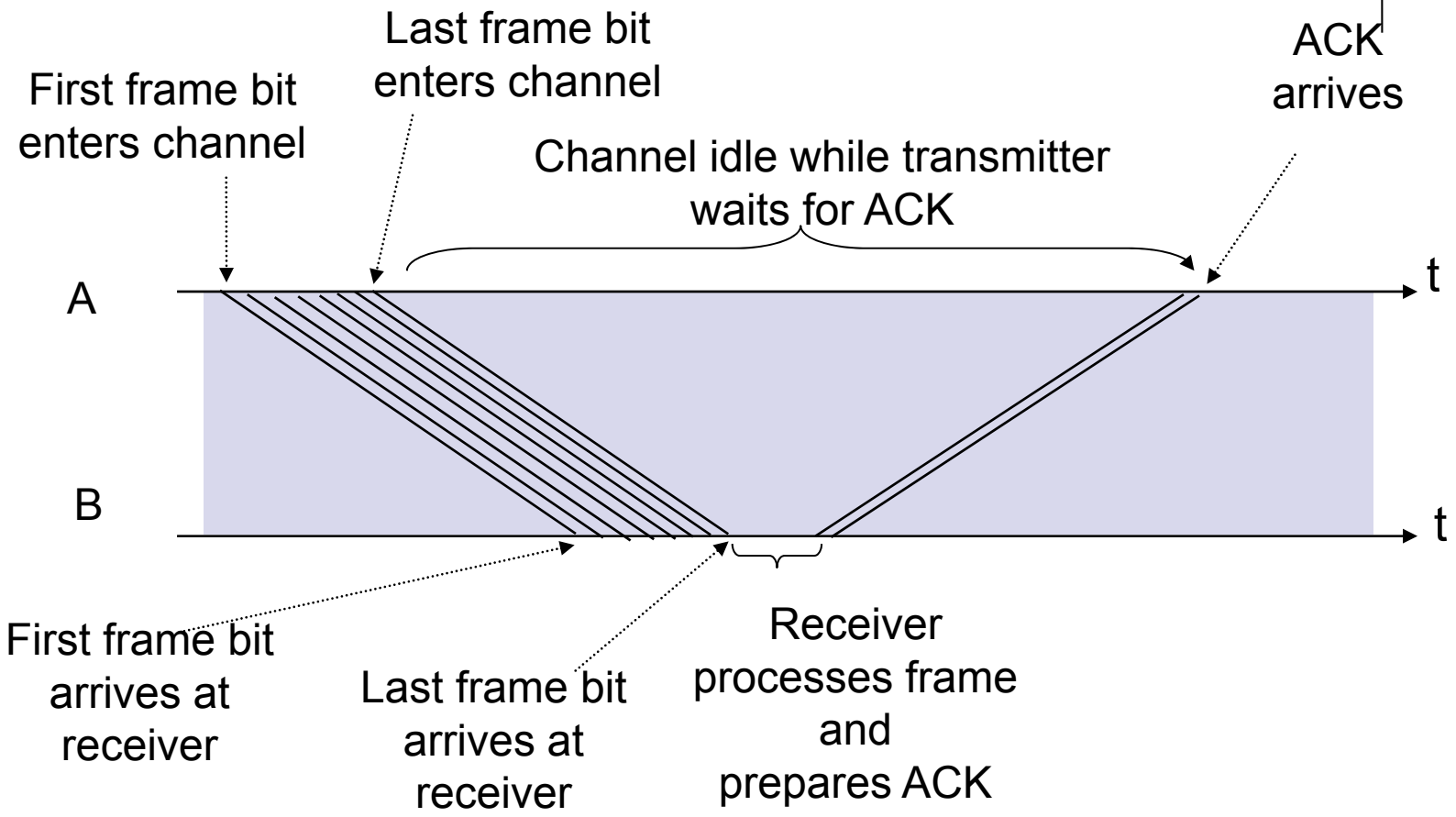
- Wait for arrival of new frame
- When frame arrives, check for errors
- If no errors detected and sequence number is correct ($S_{last}=R_{next}$), then
  - accept frame,
  - update R_{next},
  - send ACK frame with R_{next},
  - deliver packet to higher layer
- If no errors detected and wrong sequence number
  - discard frame
  - send ACK frame with R_{next}
- If errors detected
  - discard frame

Applications of Stop-and-Wait ARQ



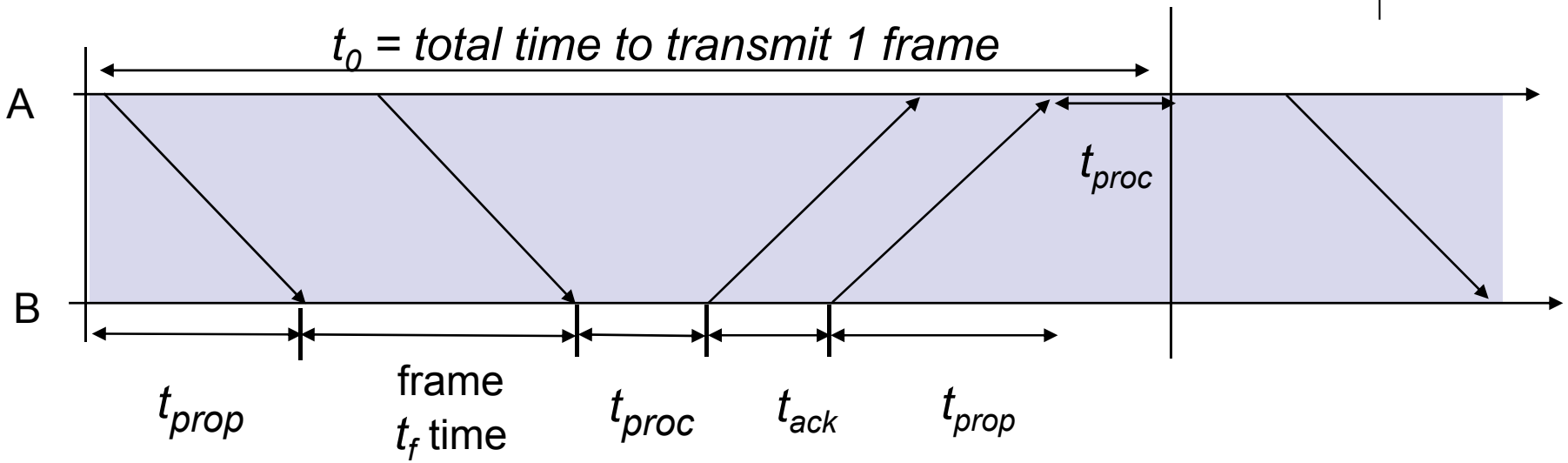
- IBM *Binary Synchronous Communications protocol* (Bisync): character-oriented data link control
- *Xmodem*: modem file transfer protocol
- *Trivial File Transfer Protocol* (RFC 1350): simple protocol for file transfer over UDP

Stop-and-Wait Efficiency



- 10000 bit frame @ 1 Mbps takes 10 ms to transmit
- If wait for ACK = 1 ms, then efficiency = 10/11= 91%
- If wait for ACK = 20 ms, then efficiency =10/30 = 33%

Stop-and-Wait Model



$$t_0 = 2t_{prop} + 2t_{proc} + t_f + t_{ack}$$

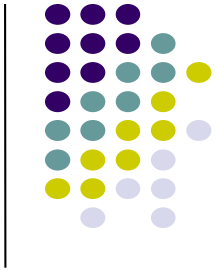
$$= 2t_{prop} + 2t_{proc} + \frac{n_f}{R} + \frac{n_a}{R}$$

bits/info frame

bits/ACK frame

channel transmission rate

S&W Efficiency on Error-free channel



Effective transmission rate:

bits for header & CRC

$$R_{eff}^{0} = \frac{\text{number of information bits delivered to destination}}{\text{total time required to deliver the information bits}} = \frac{n_f - n_o}{t_0},$$

Transmission efficiency:

$$\eta_0 = \frac{R_{eff}}{R} = \frac{\dfrac{n_f - n_o}{t_0}}{R} = \frac{1 - \dfrac{n_o}{n_f}}{1 + \dfrac{n_a}{n_f} + \dfrac{2(t_{prop} + t_{proc})R}{n_f}}.$$

Effect of frame overhead

Effect of ACK frame

Effect of ***Delay-Bandwidth Product***

Example: Impact of Delay-Bandwidth Product



n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

2xDelayxBW Efficiency	1 ms 200 km	10 ms 2000 km	100 ms 20000 km	1 sec 200000 km
1 Mbps	10^3 88%	10^4 49%	10^5 9%	10^6 1%
1 Gbps	10^6 1%	10^7 0.1%	10^8 0.01%	10^9 0.001%

Stop-and-Wait does not work well for very high speeds or long propagation delays

S&W Efficiency in Channel with Errors



- Let 1 – P_f = probability frame arrives w/o errors
- Avg. # of transmissions to first correct arrival is then 1/ (1–P_f)
- "If 1-in-10 get through without error, then avg. 10 tries to success"
- Avg. Total Time per frame is then $t_0/(1 - P_f)$

$$\eta_{SW} = \frac{R_{eff}}{R} = \frac{\frac{n_f - n_o}{t_0}(1-P_f)}{R} = \frac{1-\frac{n_o}{n_f}}{1+\frac{n_a}{n_f}+\frac{2(t_{prop}+t_{proc})R}{n_f}}(1-P_f)$$

Effect of frame loss

Example: Impact Bit Error Rate



n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

Find efficiency for random bit errors with p=0, 10^{-6}, 10^{-5}, 10^{-4}

$$1-P_f = (1-p)^{n_f} \approx e^{-n_f p} \text{ for large } n_f \text{ and small } p$$

$1-P_f$ **Efficiency**	0	10^{-6}	10^{-5}	10^{-4}
1 Mbps & 1 ms	1 88%	0.99 86.6%	0.905 79.2%	0.368 32.2%

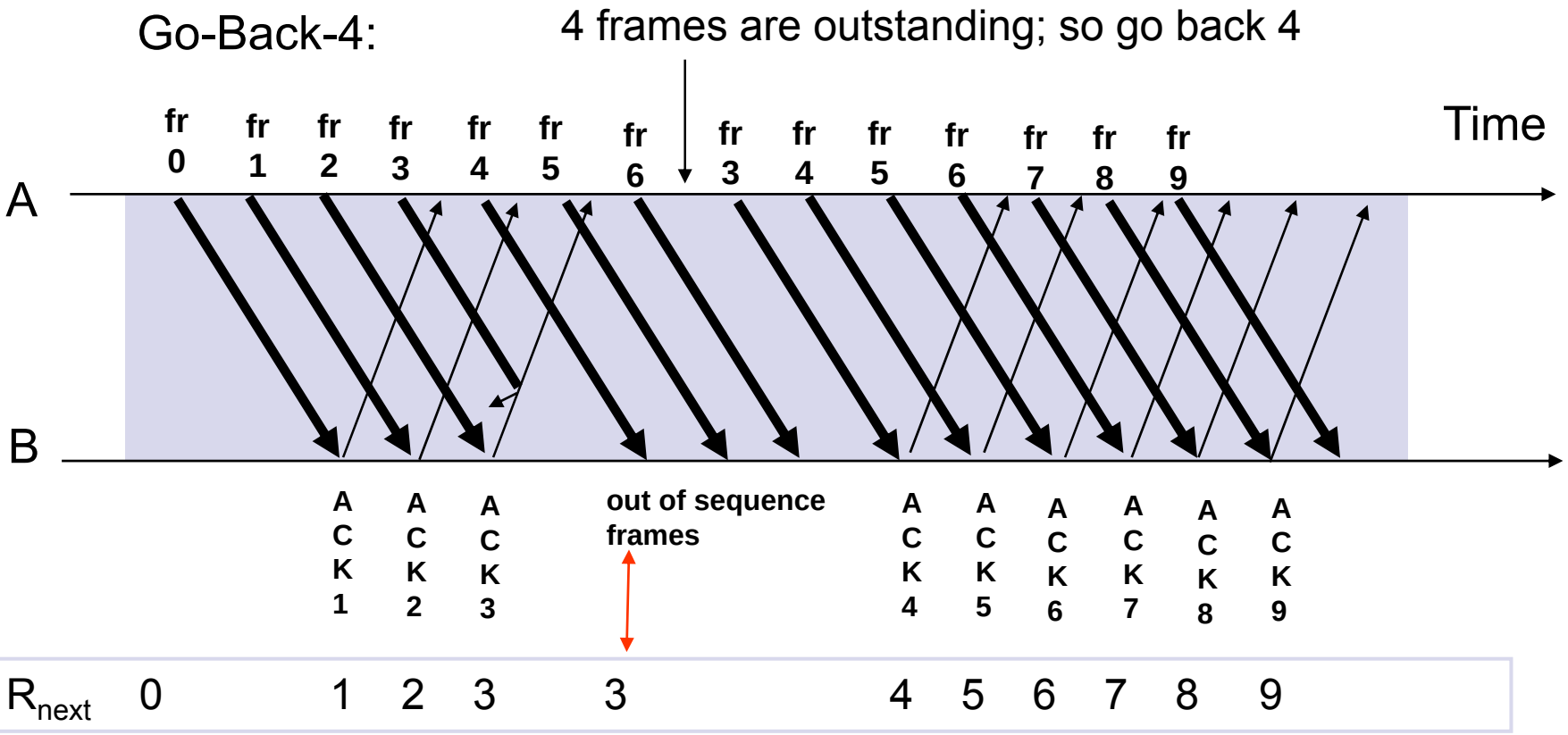
Bit errors impact performance as $n_f p$ *approach 1*

Go-Back-N



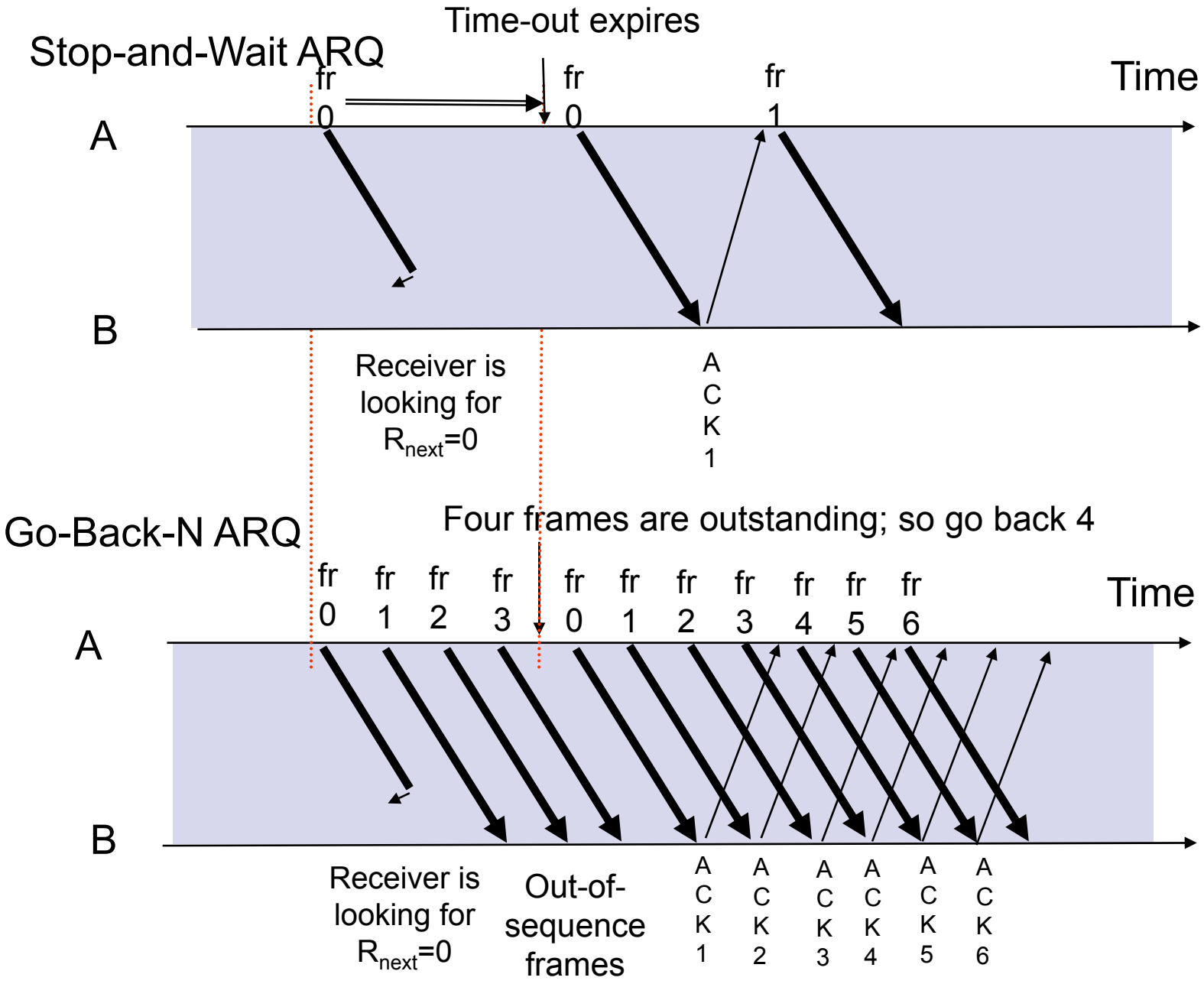
- Improve Stop-and-Wait by not waiting!
- Keep channel busy by continuing to send frames
- Allow a window of up to W_s outstanding frames
- Use *m*-bit sequence numbering
- If ACK for oldest frame arrives before window is exhausted, we can continue transmitting
- If window is exhausted, pull back and retransmit all outstanding frames
- Alternative: Use timeout

Go-Back-N ARQ



- Frame transmission are *pipelined* to keep the channel busy
- Frame with errors and subsequent out-of-sequence frames are ignored
- Transmitter is forced to go back when window of 4 is exhausted

Window size long enough to cover round trip time



Go-Back-N with Timeout

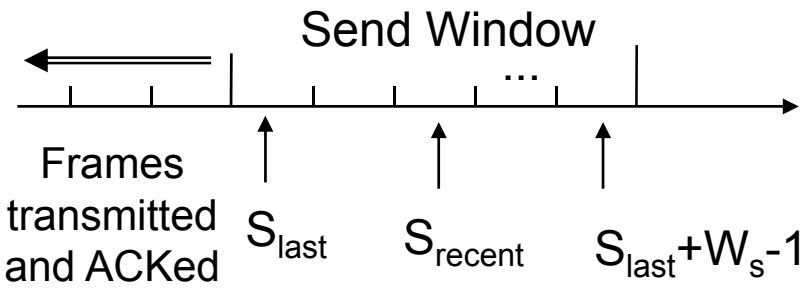


- Problem with Go-Back-N as presented:
  - If frame is lost and source does not have frame to send, then window will not be exhausted and recovery will not commence
- Use a timeout with each frame
  - When timeout expires, resend all outstanding frames

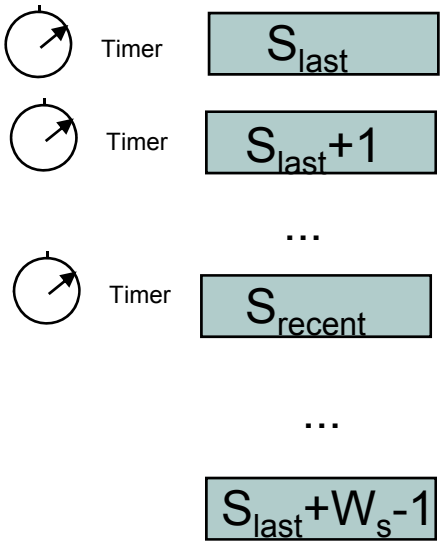
Go-Back-N Transmitter & Receiver



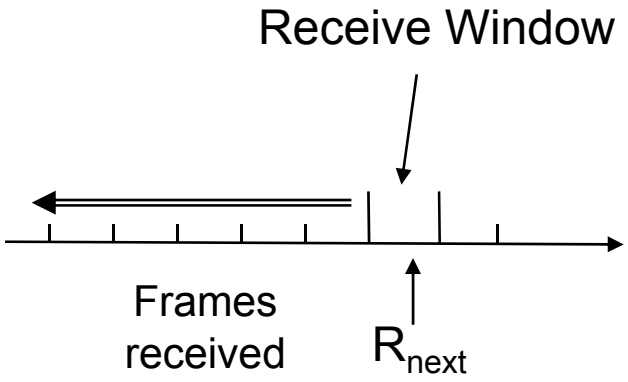
Transmitter



Buffers



Receiver



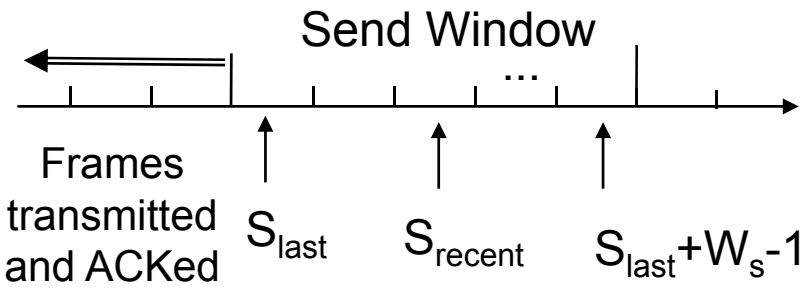
Receiver will only accept a frame that is error-free and that has sequence number R_{next}

When such frame arrives R_{next} is incremented by one, so the *receive window slides forward* by one

Sliding Window Operation



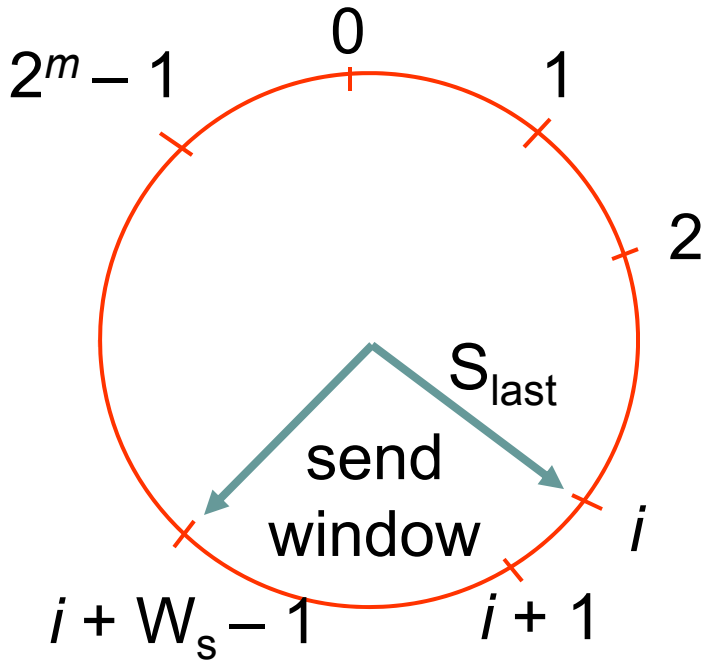
Transmitter



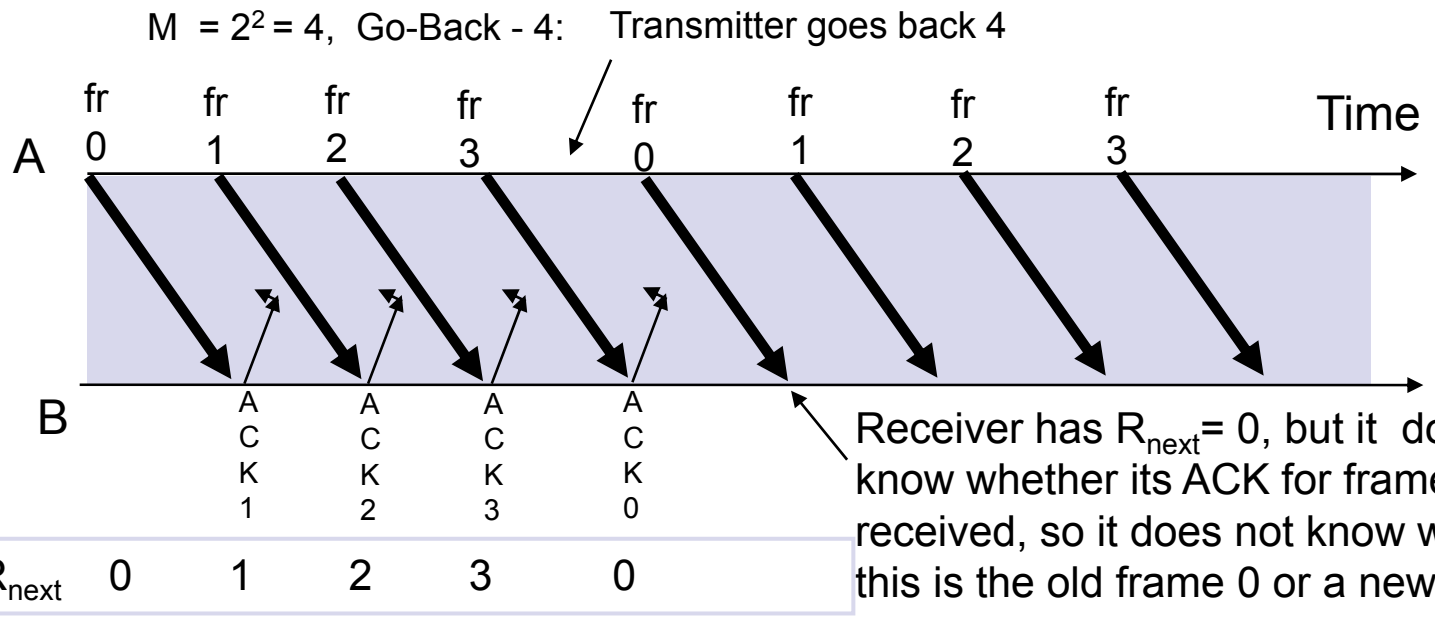
Transmitter waits for error-free ACK frame with sequence number S_{last}

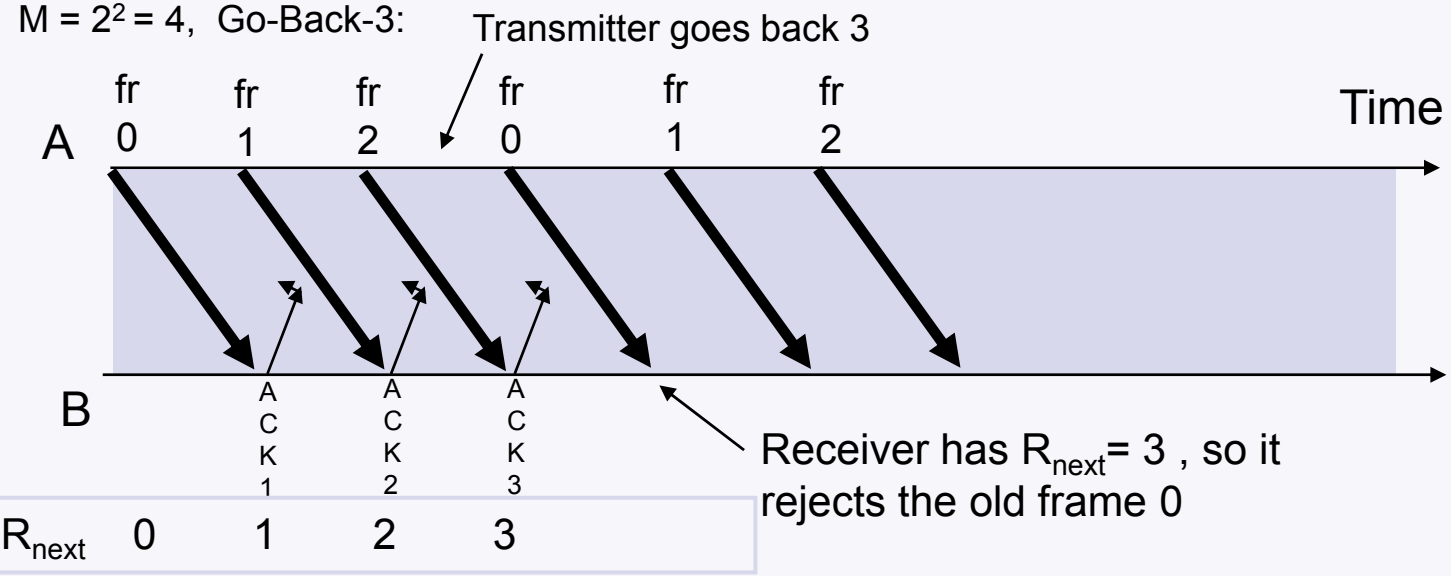
When such ACK frame arrives, S_{last} is incremented by one, and the *send window slides forward* by one

m-bit Sequence Numbering

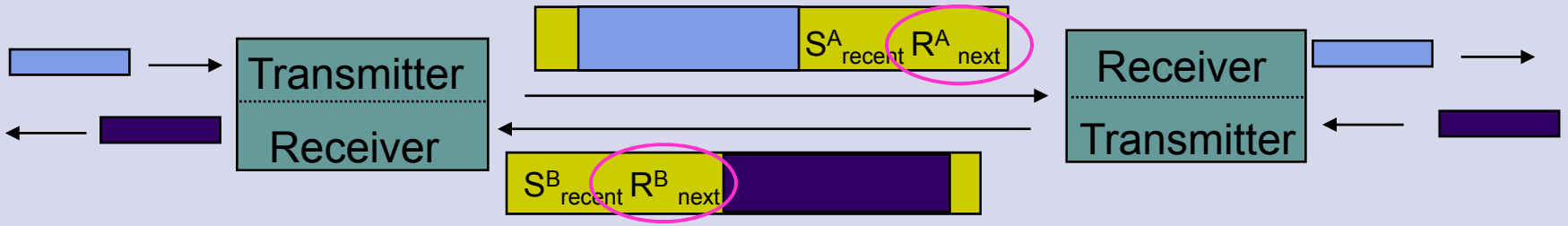


Maximum Allowable Window Size is $W_s = 2^m - 1$

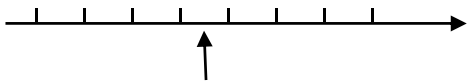




ACK Piggybacking in Bidirectional GBN

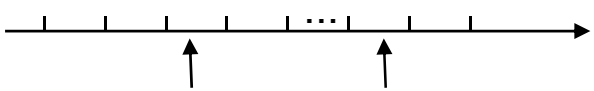


"A" Receive Window



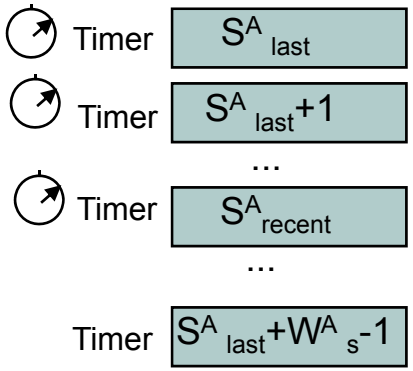
R^A_{next}

"A" Send Window

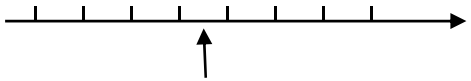


S^A_{last} $S^A_{last}+W^A_s-1$

Buffers

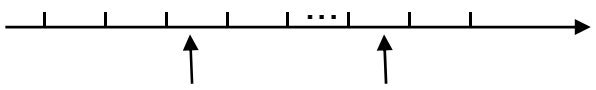


"B" Receive Window



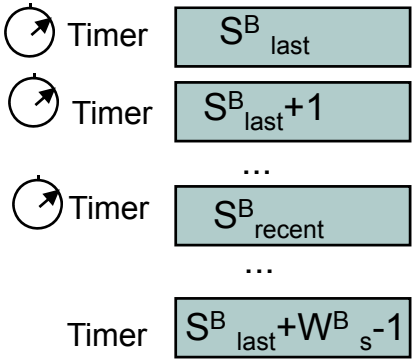
R^B_{next}

"B" Send Window



S^B_{last} $S^B_{last}+W^B_s-1$

Buffers



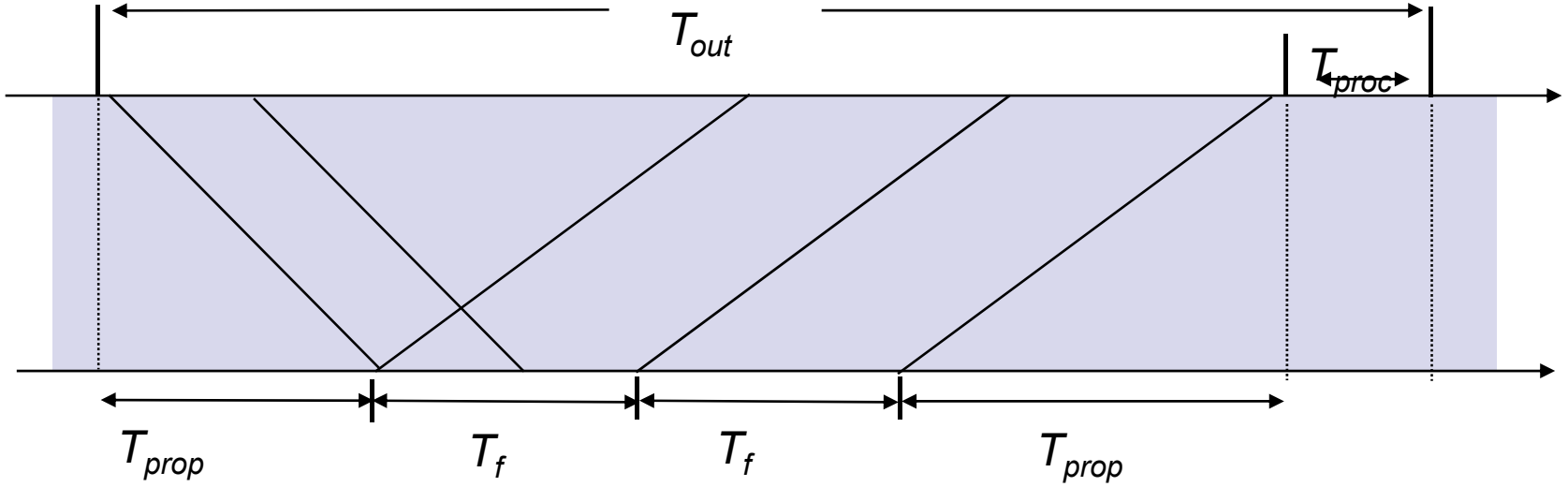
Note: Out-of-sequence error-free frames discarded after R_{next} examined

Applications of Go-Back-N ARQ



- *HDLC* (High-Level Data Link Control): bit-oriented data link control
- *V.42 modem*: error control over telephone modem links

Required Timeout & Window Size



- Timeout value should allow for:
  - Two propagation times + 1 processing time: $2\ T_{prop} + T_{proc}$
  - A frame that begins transmission right before our frame arrives T_f
  - Next frame carries the ACK, T_f
- W_s should be large enough to keep channel busy for T_{out}

Required Window Size for Delay-Bandwidth Product



Frame = 1250 bytes =10,000 bits, *R* = 1 Mbps		
$2(t_{prop} + t_{proc})$	**2 x Delay x BW**	**Window**
1 ms	1000 bits	1
10 ms	10,000 bits	2
100 ms	100,000 bits	11
1 second	1,000,000 bits	101

Efficiency of Go-Back-N



- GBN is completely efficient, if W_s large enough to keep channel busy, and if channel is error-free
- Assume P_f frame loss probability, then time to deliver a frame is:
  - t_f if first frame transmission succeeds $(1 - P_f)$
  - $T_f + W_s t_f/(1-P_f)$ if the first transmission does not succeed P_f

$$t_{GBN} = t_f(1-P_f) + P_f\{t_f + \frac{W_s t_f}{1-P_f}\} = t_f + P_f \frac{W_s t_f}{1-P_f} \quad \text{and}$$

$$\eta_{GBN} = \frac{\frac{n_f - n_o}{t_{GBN}}}{R} = \frac{1-\frac{n_o}{n_f}}{1+(W_s-1)P_f}(1-P_f)$$

Delay-bandwidth product determines W_s

Example: Impact Bit Error Rate on GBN



n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

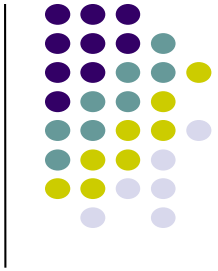
Compare S&W with GBN efficiency for random bit errors with p = 0, 10^{-6}, 10^{-5}, 10^{-4} and R = 1 Mbps & 100 ms

1 Mbps x 100 ms = 100000 bits = 10 frames → Use W_s = 11

Efficiency	0	10^{-6}	10^{-5}	10^{-4}
S&W	8.9%	8.8%	8.0%	3.3%
GBN	98%	88.2%	45.4%	4.9%

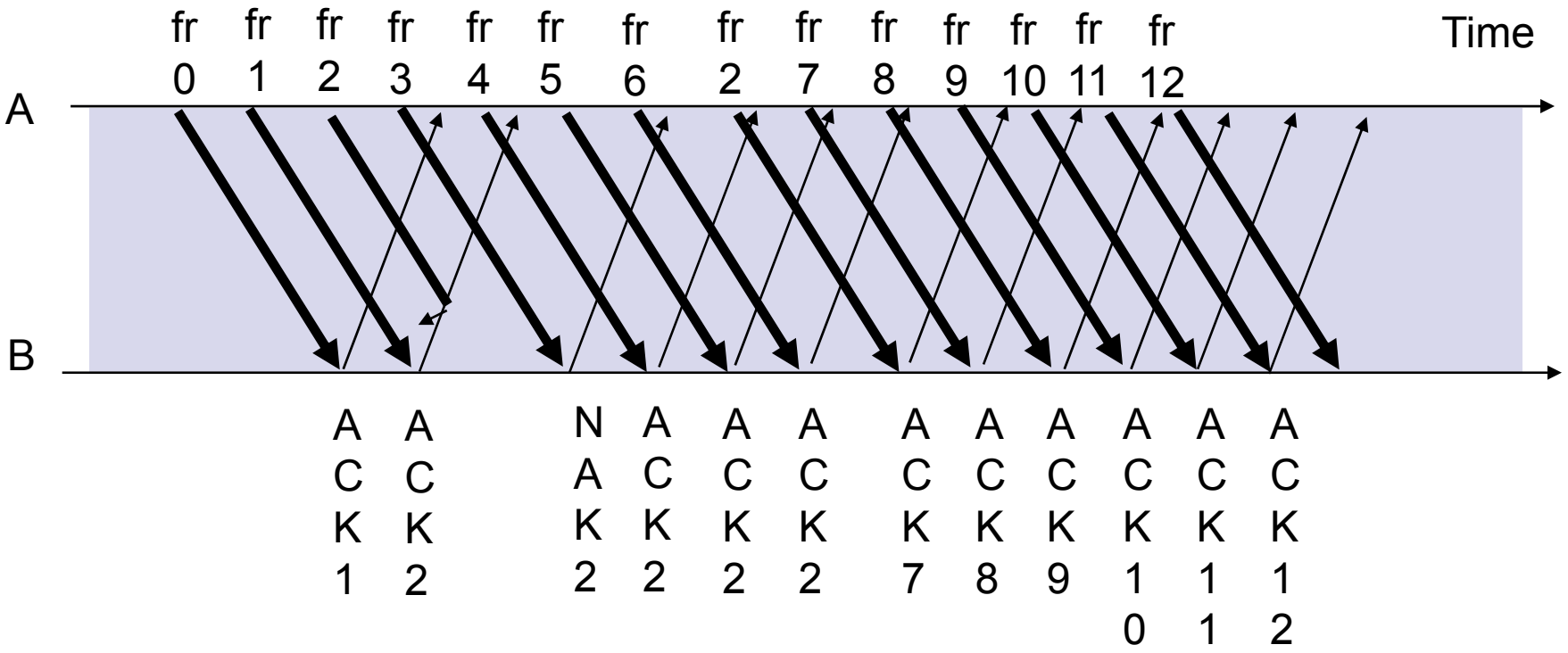
- *Go-Back-N significant improvement over Stop-and-Wait for large delay-bandwidth product*
- *Go-Back-N becomes inefficient as error rate increases*

Selective Repeat ARQ

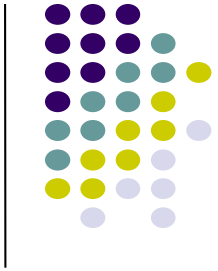


- Go-Back-N ARQ inefficient because *multiple* frames are resent when errors or losses occur
- Selective Repeat retransmits *only an individual frame*
  - Timeout causes individual corresponding frame to be resent
  - NAK causes retransmission of oldest un-acked frame
- Receiver maintains a *receive window* of sequence numbers that can be accepted
  - Error-free, but out-of-sequence frames with sequence numbers within the receive window are buffered
  - Arrival of frame with R_{next} causes window to slide forward by 1 or more

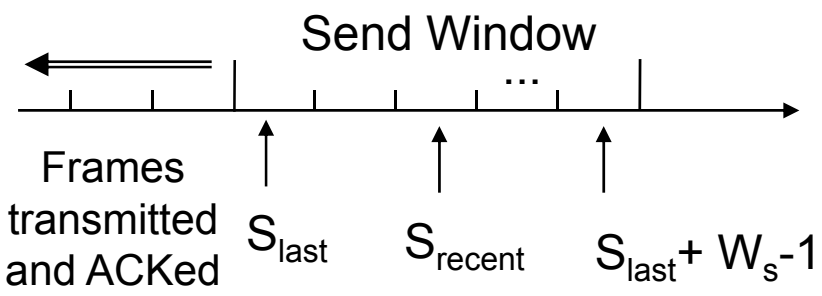
Selective Repeat ARQ

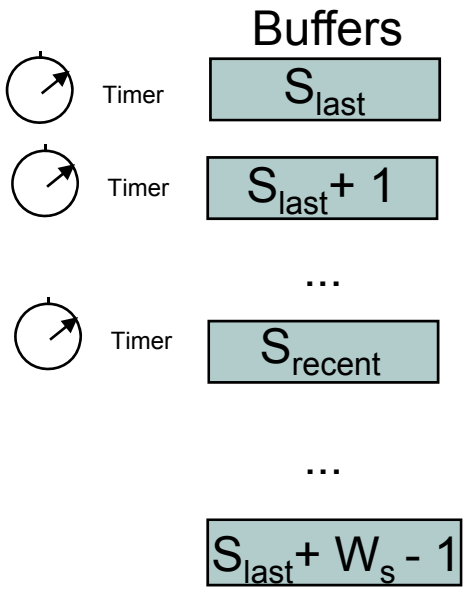


Selective Repeat ARQ

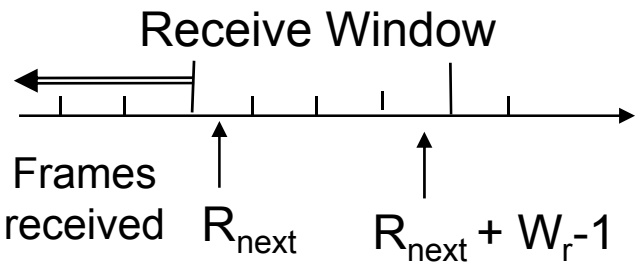


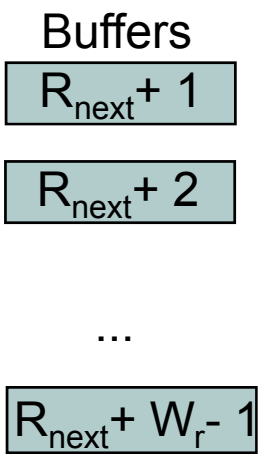
Transmitter





Receiver

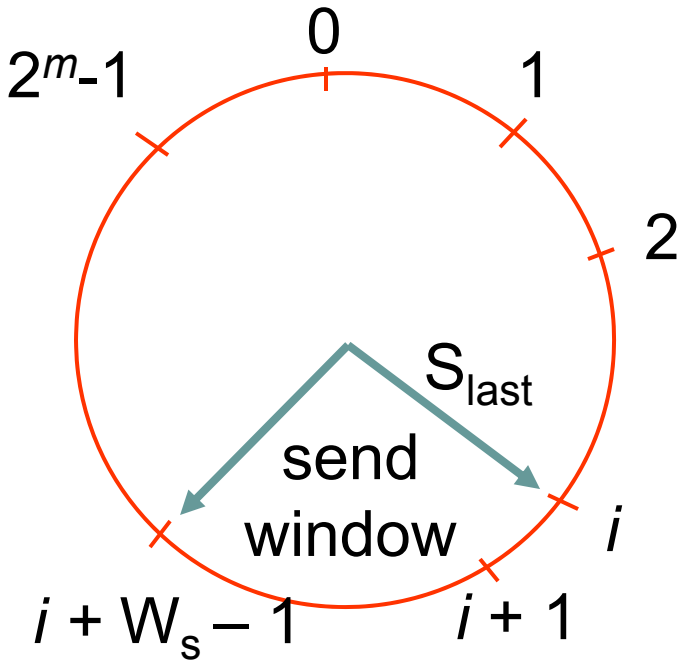




max Seq # accepted

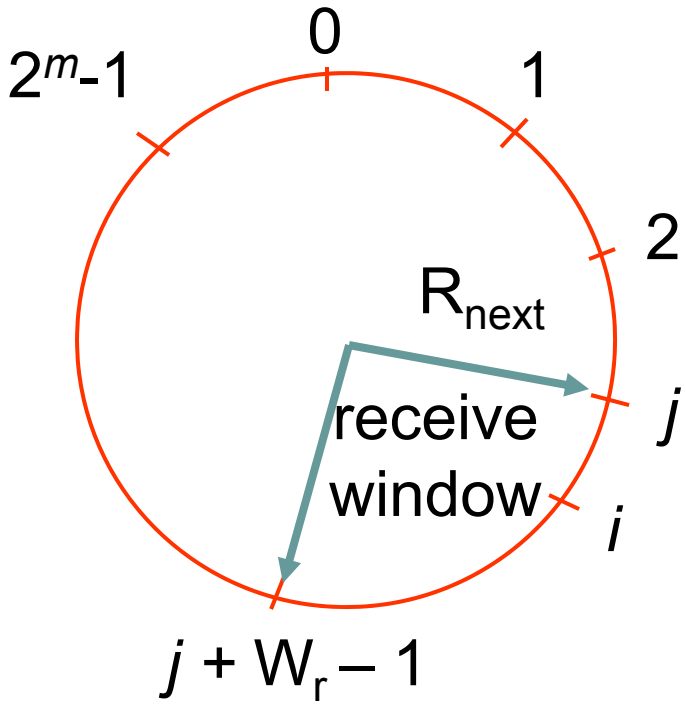
Send & Receive Windows

Transmitter



Moves k forward when ACK arrives with $R_{next} = S_{last} + k$
$k = 1, \ldots, W_s\text{-}1$

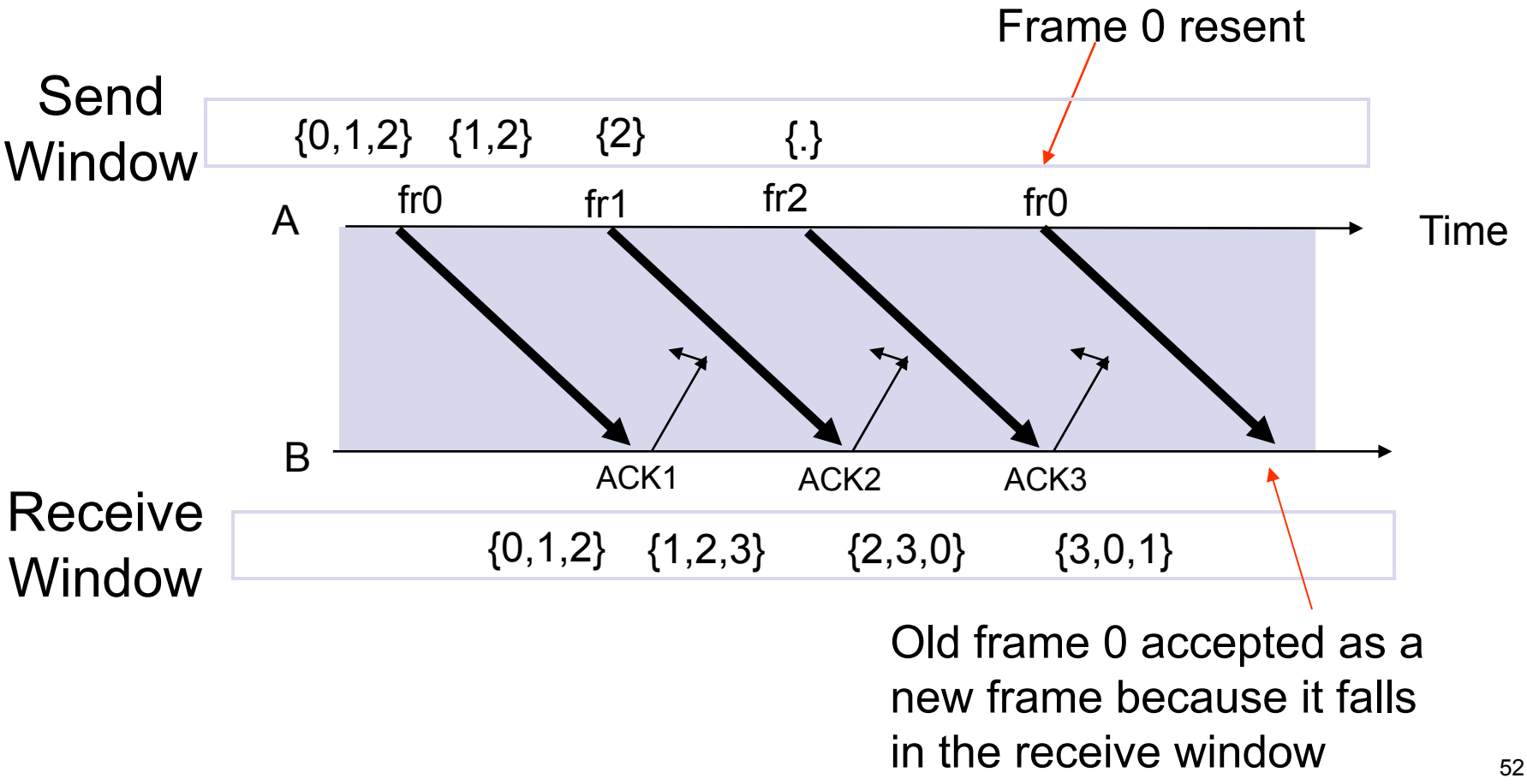
Receiver



Moves forward by 1 or more when frame arrives with Seq. # = R_{next}

What size W_s and W_r allowed?

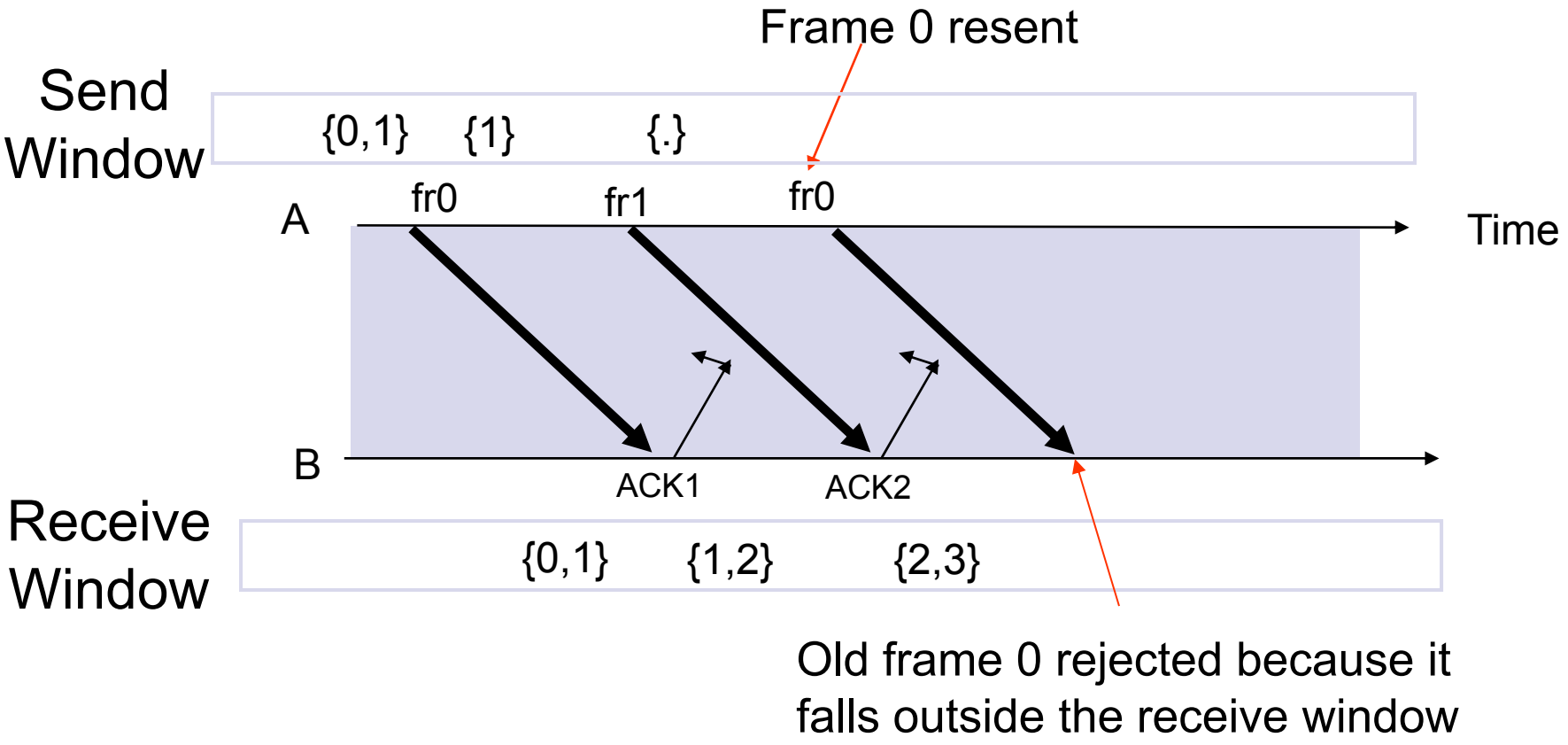
- Example: $M=2^2=4$, $W_s=3$, $W_r=3$



$W_s + W_r = 2^m$ is maximum allowed



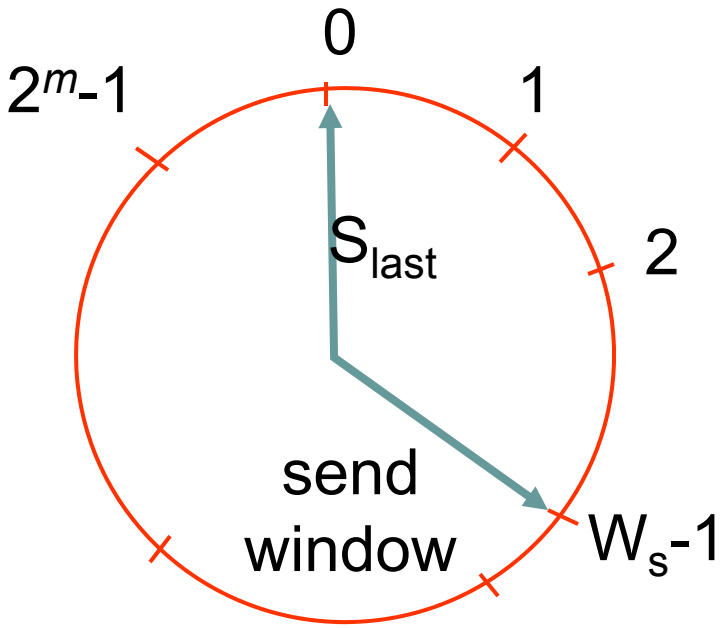
- Example: $M=2^2=4$, $W_s=2$, $W_r=2$

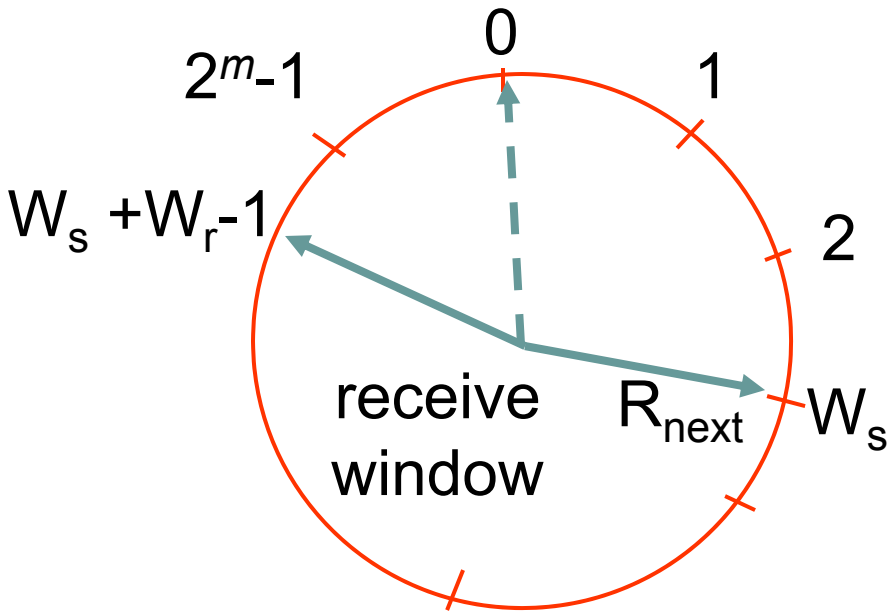


Why $W_s + W_r = 2^m$ works

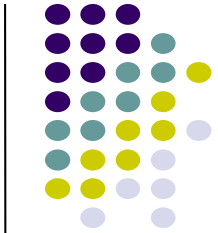
- Transmitter sends frames 0 to Ws-1; send window empty
- All arrive at receiver
- All ACKs lost
- Transmitter resends frame 0

- Receiver window starts at {0, …, W_r}
- Window slides forward to {W_s,…,W_s+W_r-1}
- Receiver rejects frame 0 because it is outside receive window





Applications of Selective Repeat ARQ



- *TCP* (Transmission Control Protocol): transport layer protocol uses variation of selective repeat to provide reliable stream service
- *Service Specific Connection Oriented Protocol*: error control for signaling messages in ATM networks

Efficiency of Selective Repeat



- Assume P_f frame loss probability, then number of transmissions required to deliver a frame is:
  - $t_f / (1-P_f)$

$$\eta_{SR} = \frac{\frac{n_f - n_o}{t_f/(1-P_f)}}{R} = (1-\frac{n_o}{n_f})(1-P_f)$$

Example: Impact Bit Error Rate on Selective Repeat



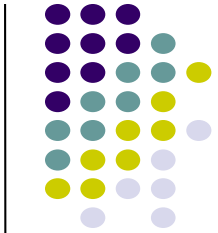
n_f=1250 bytes = 10000 bits, n_a=n_o=25 bytes = 200 bits

Compare S&W, GBN & SR efficiency for random bit errors with p=0, 10^{-6}, 10^{-5}, 10^{-4} and R= 1 Mbps & 100 ms

Efficiency	0	10^{-6}	10^{-5}	10^{-4}
S&W	8.9%	8.8%	8.0%	3.3%
GBN	98%	88.2%	45.4%	4.9%
SR	98%	97%	89%	36%

- *Selective Repeat outperforms GBN and S&W, but efficiency drops as error rate increases*

Comparison of ARQ Efficiencies



Assume n_a and n_o are negligible relative to n_f, and $L = 2(t_{prop}+t_{proc})R/n_f = (W_s-1)$, then

Selective-Repeat:

$$\eta_{SR} = (1-P_f)(1-\frac{n_o}{n_f}) \approx (1-P_f)$$

Go-Back-N:

For $P_f \approx 0$, SR & GBN same

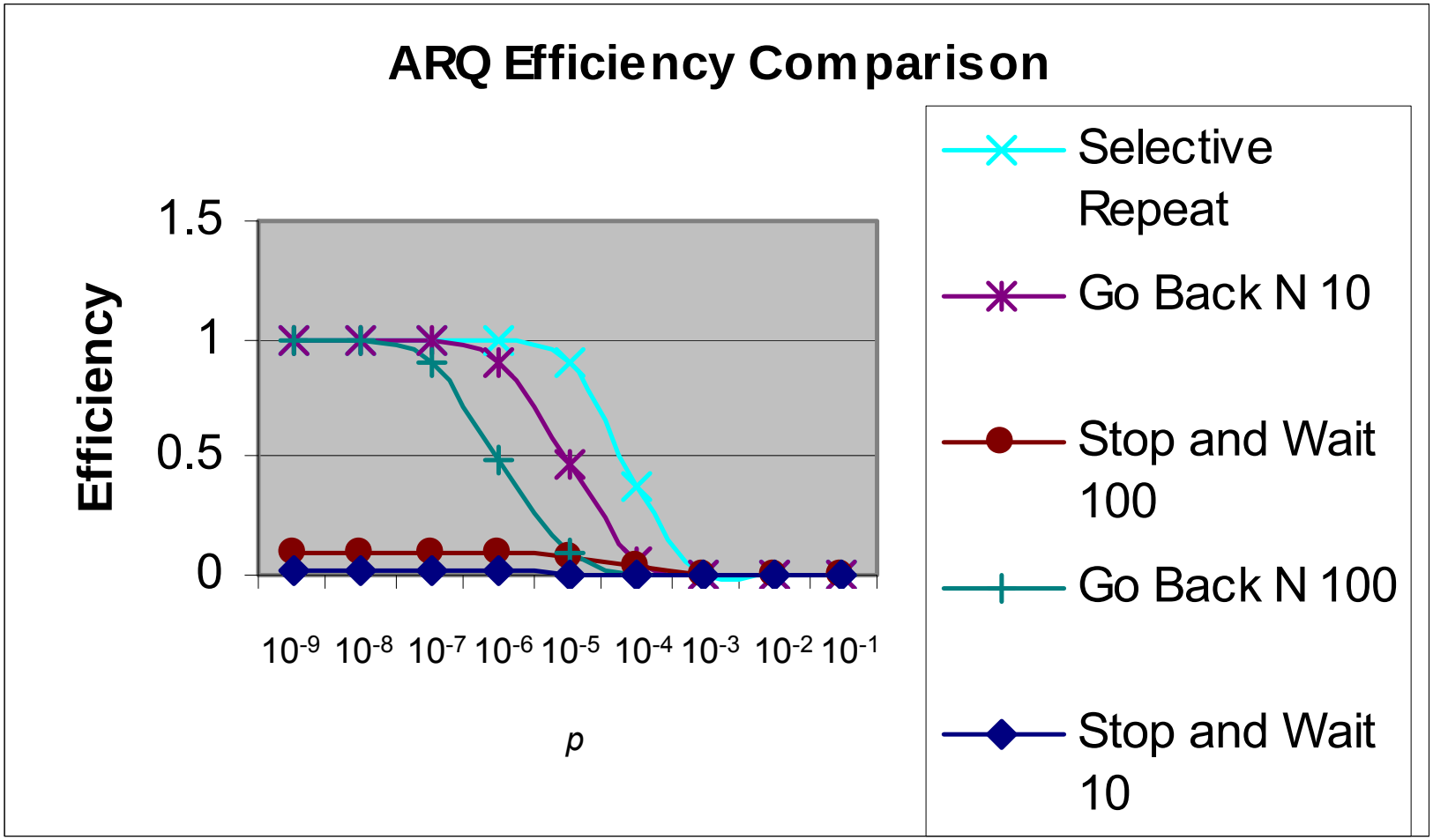
$$\eta_{GBN} = \frac{1-P_f}{1+(W_S-1)P_f} = \frac{1-P_f}{1+LP_f}$$

For $P_f \rightarrow 1$, GBN & SW same

Stop-and-Wait:

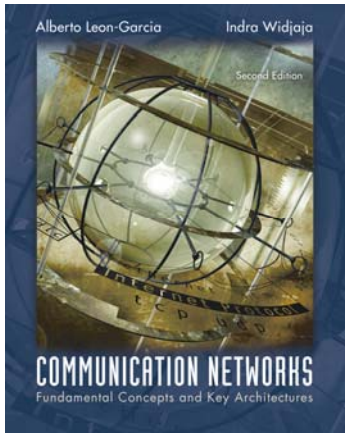
$$\eta_{SW} = \frac{(1-P_f)}{1+\frac{n_a}{n_f}+\frac{2(t_{prop}+t_{proc})R}{n_f}} \approx \frac{1-P_f}{1+L}$$

ARQ Efficiencies



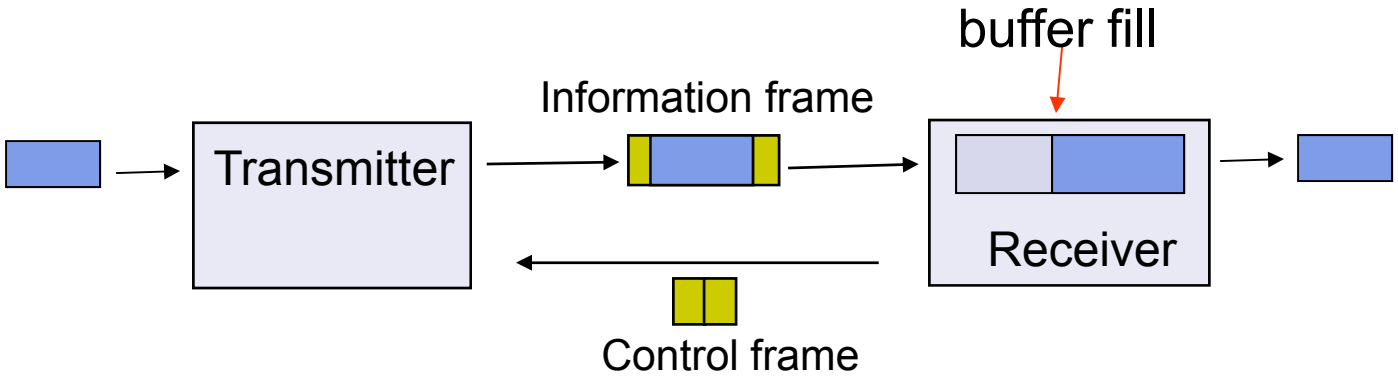
Delay-Bandwidth product = 10, 100

Chapter 5
Peer-to-Peer Protocols and Data Link Layer



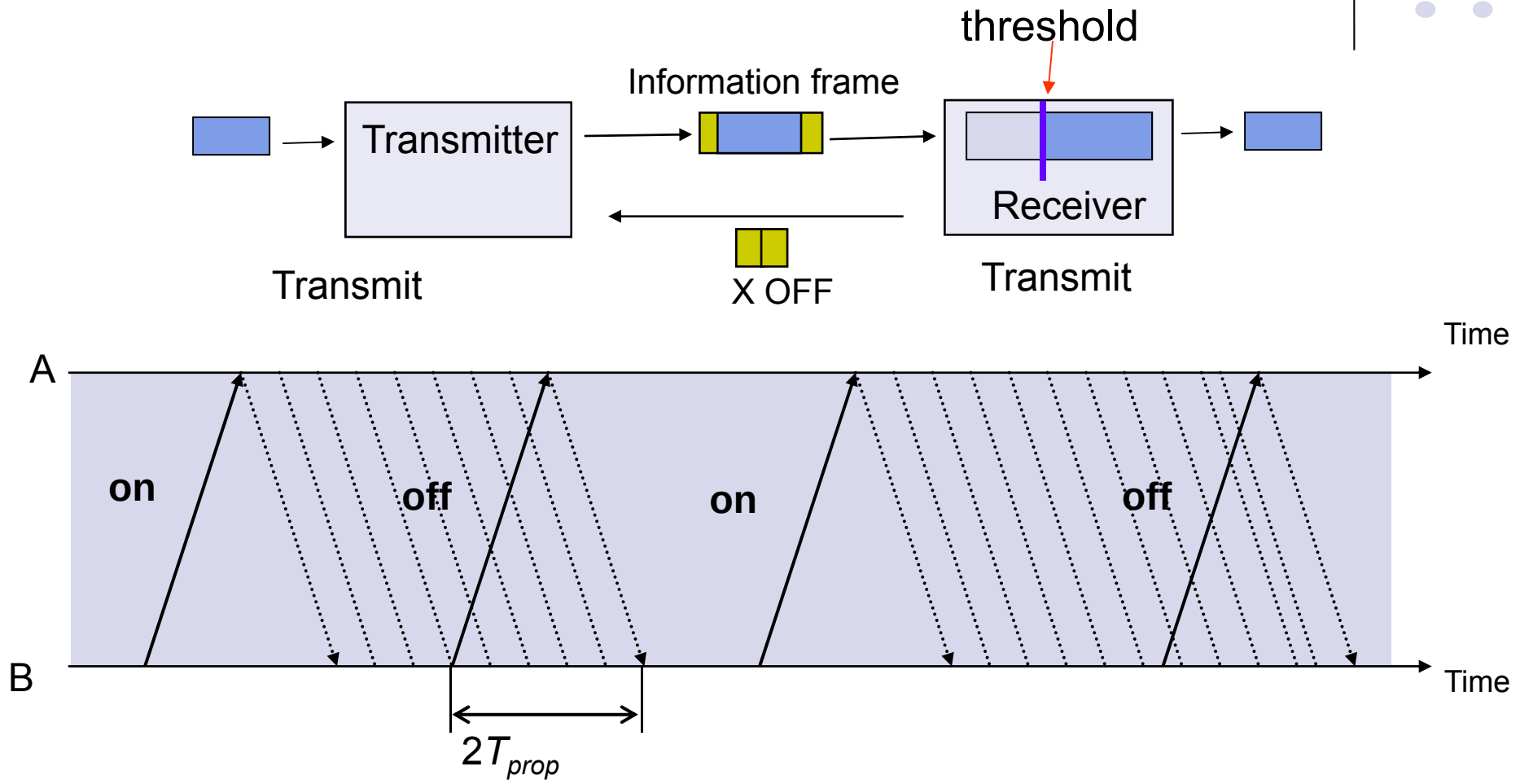
Flow Control

Flow Control



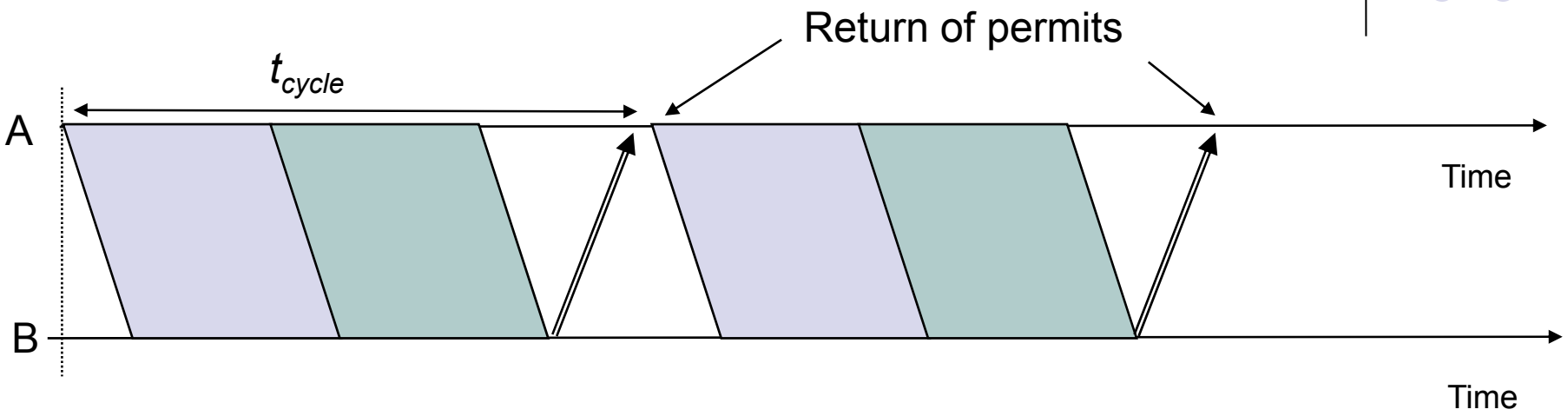
- Receiver has limited buffering to store arriving frames
- Several situations cause buffer overflow
  - Mismatch between sending rate & rate at which user can retrieve data
  - Surges in frame arrivals
- *Flow control* prevents buffer overflow by regulating rate at which source is allowed to send information

X ON / X OFF



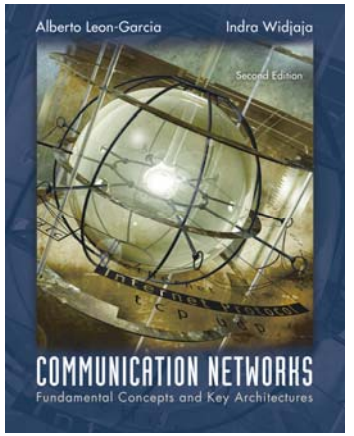
Threshold must activate OFF signal while 2 T_{prop} R bits still remain in buffer

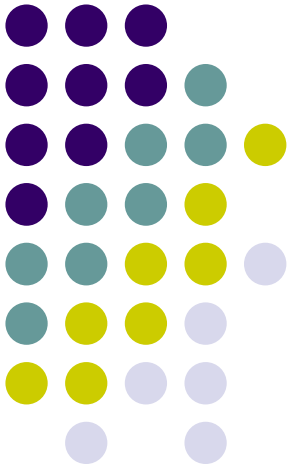
Window Flow Control



- Sliding Window ARQ method with W_s equal to buffer available
  - Transmitter can never send more than W_s frames
- ACKs that slide window forward can be viewed as permits to transmit more
- Can also pace ACKs as shown above
  - Return permits (ACKs) at end of cycle regulates transmission rate
- Problems using sliding window for both error & flow control
  - Choice of window size
  - Interplay between transmission rate & retransmissions
  - TCP separates error & flow control

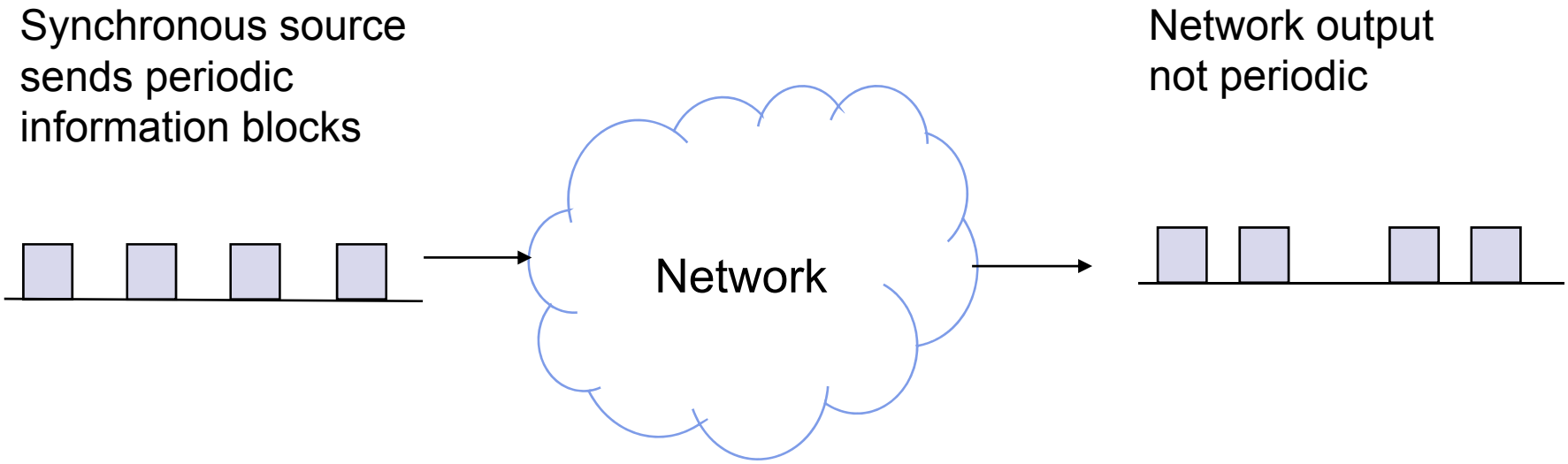
Chapter 5
Peer-to-Peer Protocols and Data Link Layer





Timing Recovery

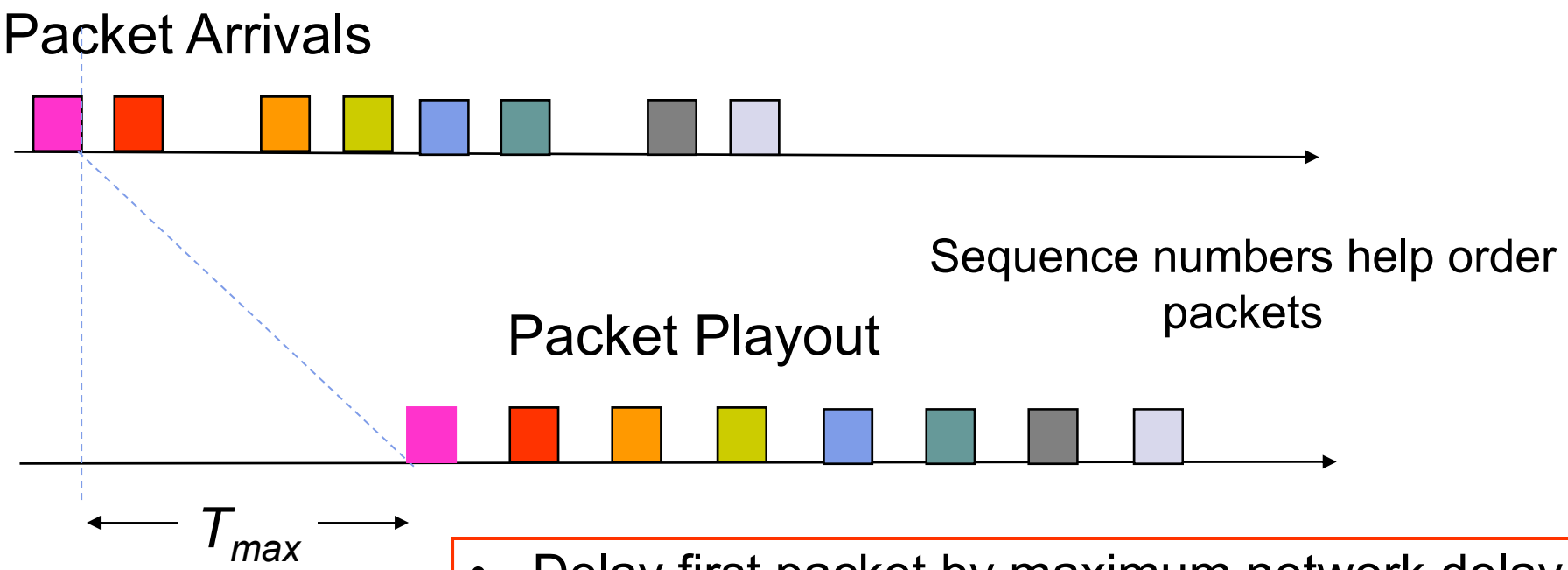
Timing Recovery for Synchronous Services



- Applications that involve voice, audio, or video can generate a synchronous information stream
- Information carried by equally-spaced fixed-length packets
- Network multiplexing & switching introduces random delays
  - Packets experience variable transfer delay
  - Jitter (variation in interpacket arrival times) also introduced
- Timing recovery re-establishes the synchronous nature of the stream

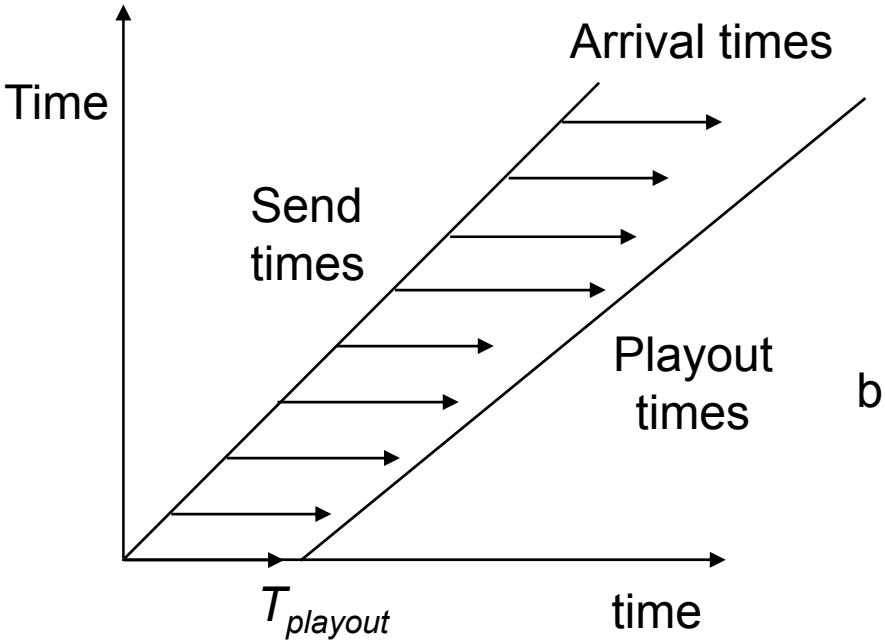
Introduce Playout Buffer



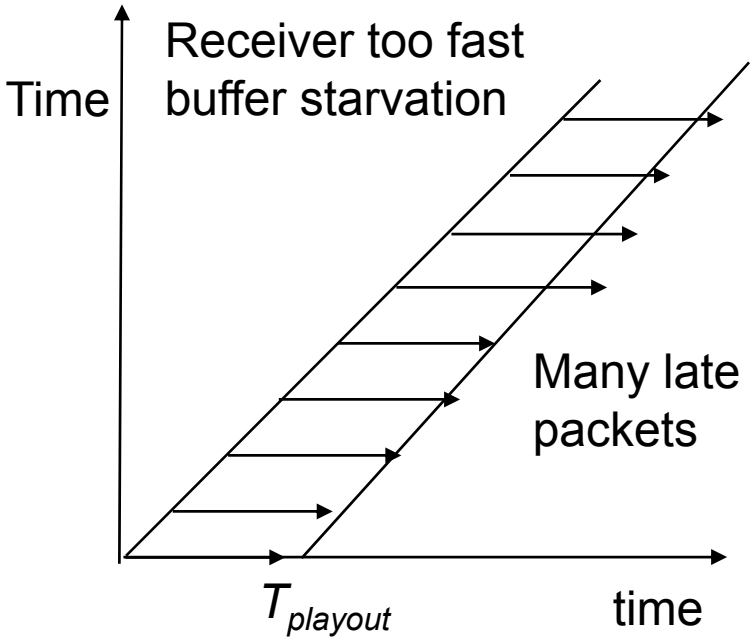


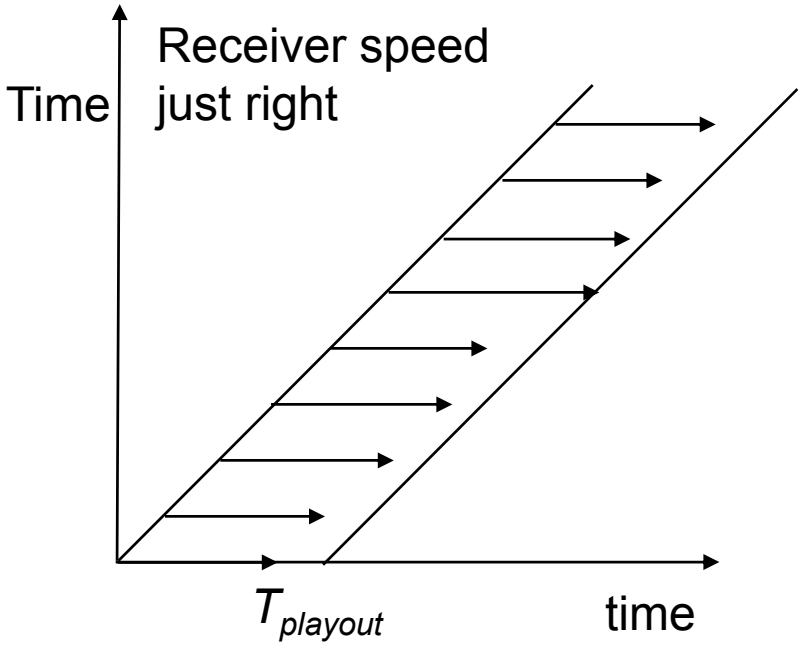
- Delay first packet by maximum network delay
- All other packets arrive with less delay
- Playout packet uniformly thereafter

Playout clock must be synchronized to transmitter clock

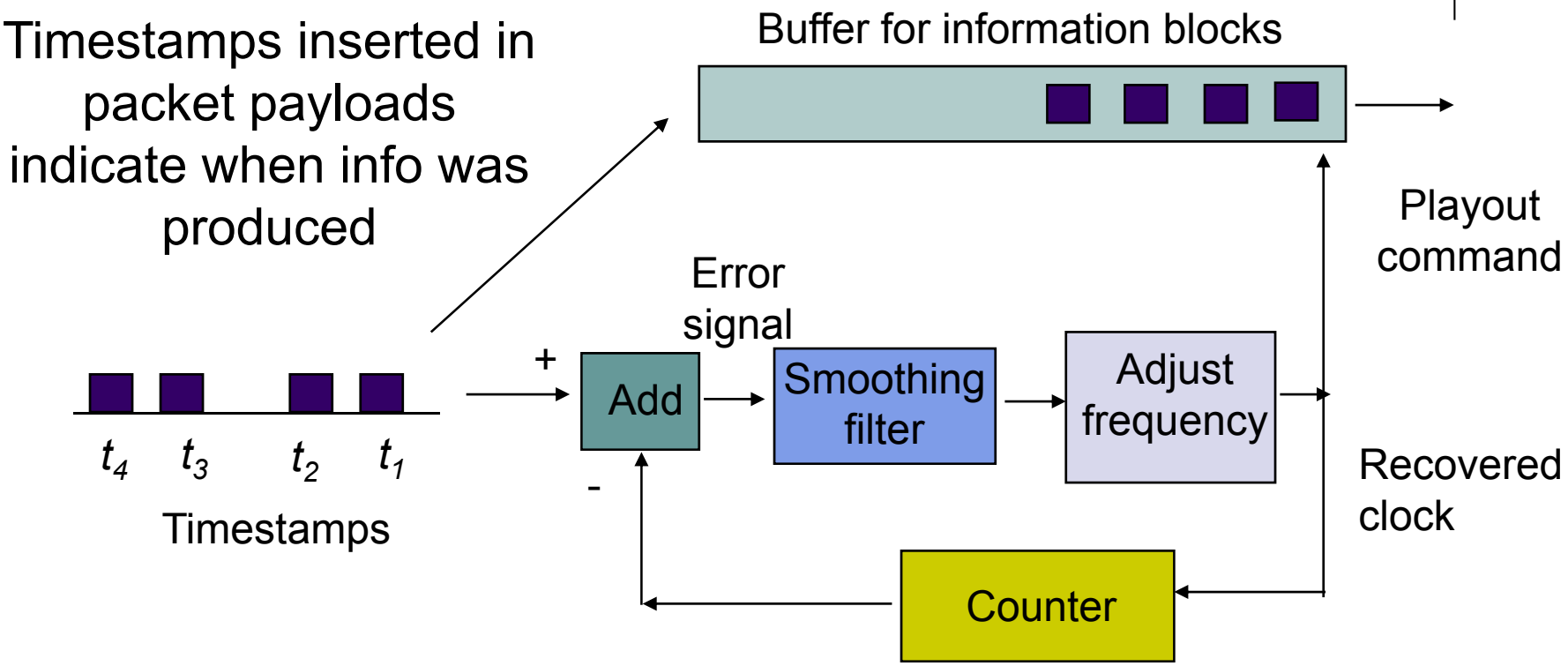


Receiver too slow; buffer fills and overflows



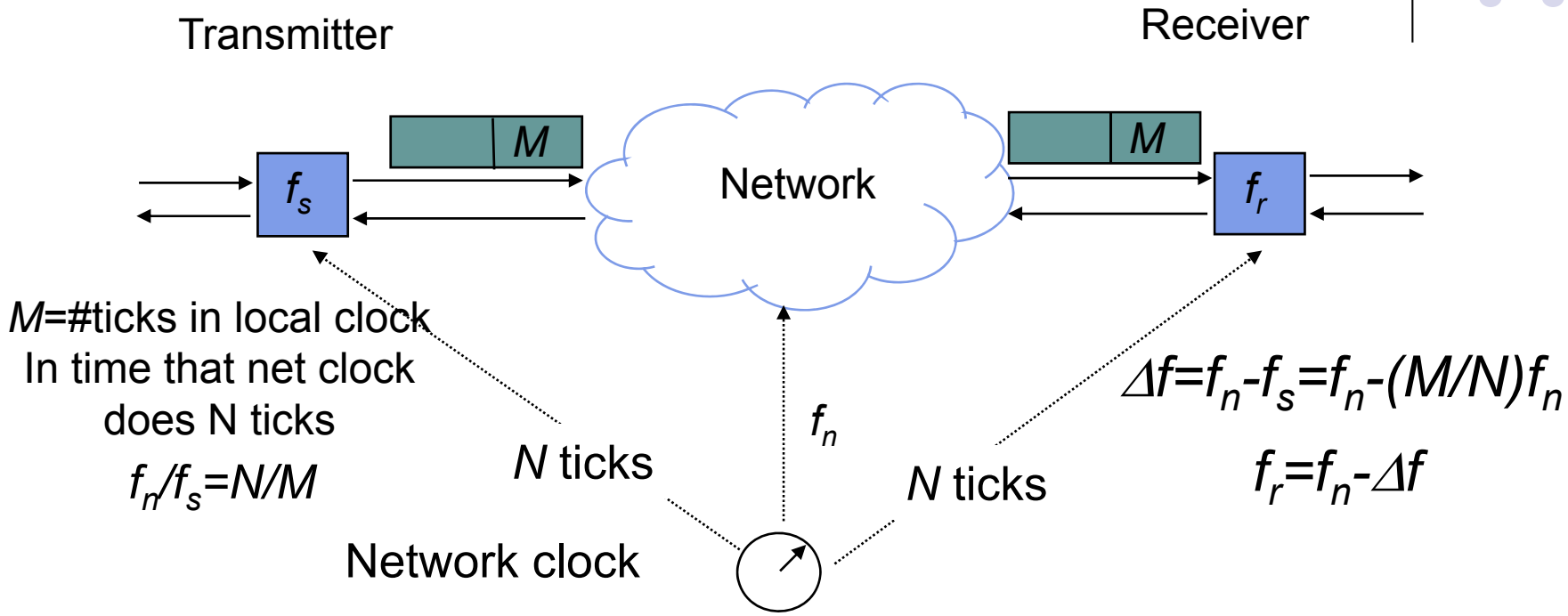


Clock Recovery



- Counter attempts to replicate transmitter clock
- Frequency of counter is adjusted according to arriving timestamps
- Jitter introduced by network causes fluctuations in buffer & in local clock

Synchronization to a Common Clock



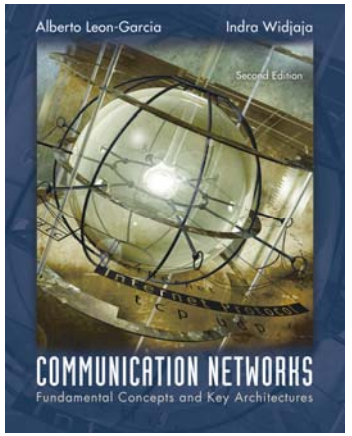
- Clock recovery simple if a common clock is available to transmitter & receiver
  - E.g. SONET network clock; Global Positioning System (GPS)
- Transmitter sends Δf of its frequency & network frequency
- Receiver adjusts network frequency by Δf
- Packet delay jitter can be removed completely

Example: Real-Time Protocol

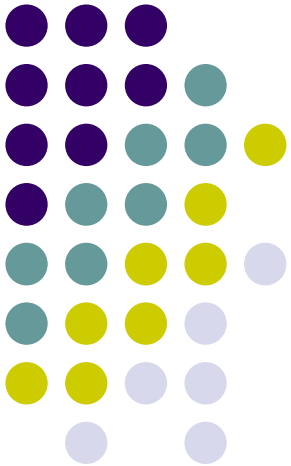


- RTP (RFC 1889) designed to support real-time applications such as voice, audio, video
- RTP provides means to carry:
  - Type of information source
  - Sequence numbers
  - Timestamps
- Actual timing recovery must be done by higher layer protocol
  - MPEG2 for video, MP3 for audio

Chapter 5
Peer-to-Peer Protocols and Data Link Layer



TCP Reliable Stream Service & Flow Control



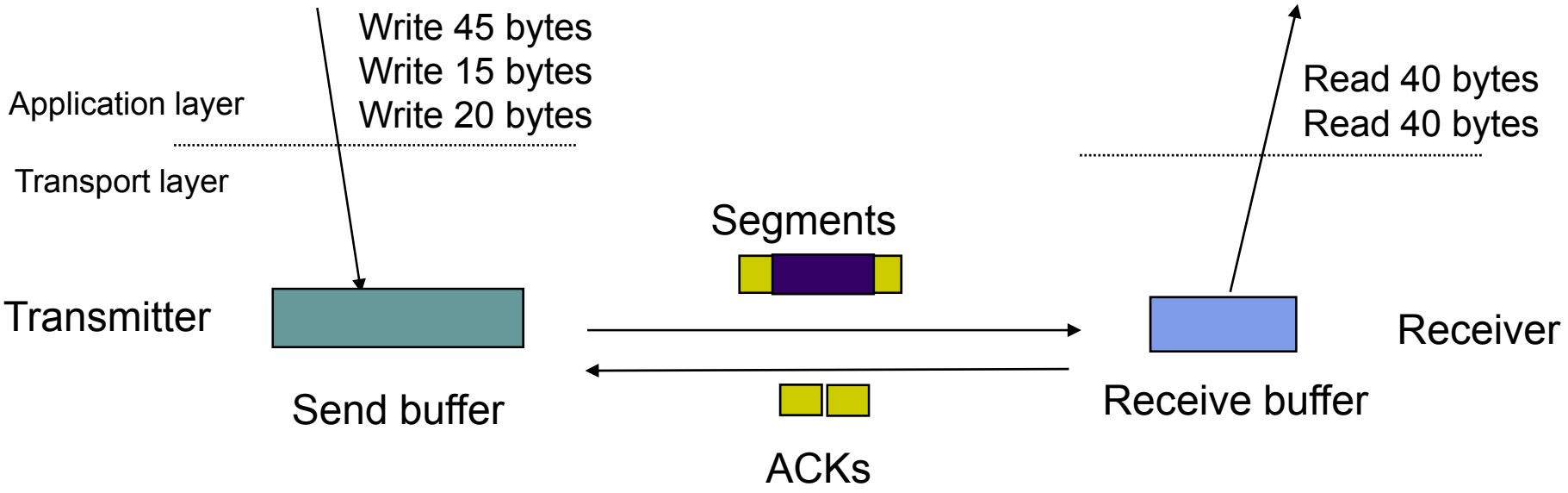
TCP Reliable Stream Service



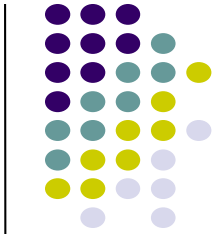
TCP transfers byte stream in order, without errors or duplications

Application Layer writes bytes into send buffer through socket

Application Layer reads bytes from receive buffer through socket

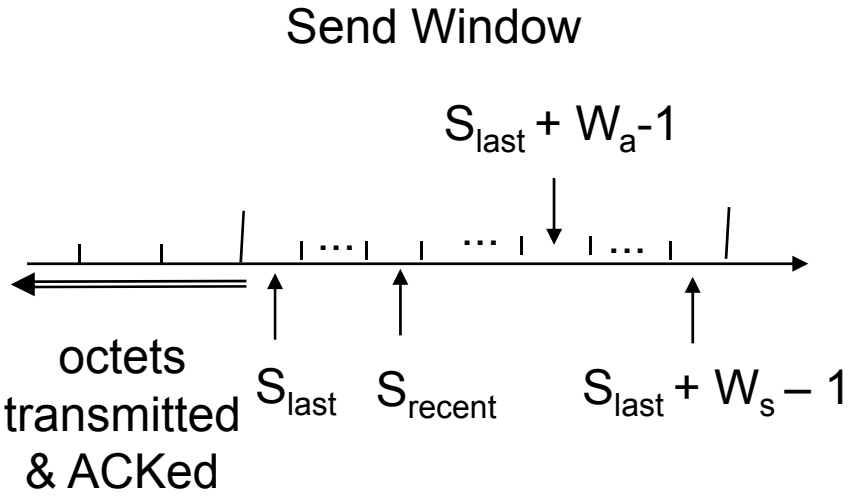


TCP ARQ Method



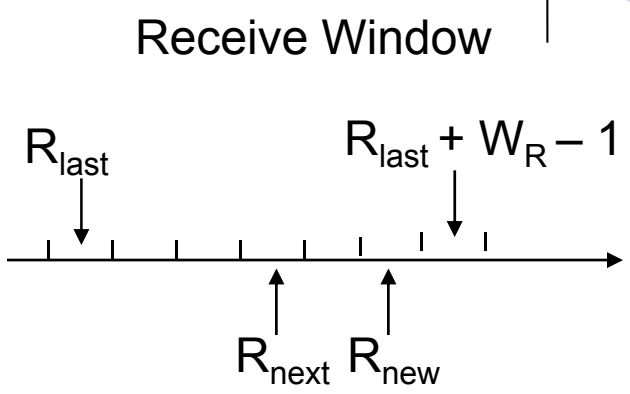
- TCP uses *Selective Repeat ARQ*
  - Transfers byte stream without preserving boundaries
- Operates over best effort service of IP
  - Packets can arrive with errors or be lost
  - Packets can arrive out-of-order
  - Packets can arrive after very long delays
  - Duplicate segments must be detected & discarded
  - Must protect against segments from previous connections
- Sequence Numbers
  - Seq. # is number of first byte in segment payload
  - Very long Seq. #s (32 bits) to deal with long delays
  - Initial sequence numbers negotiated during connection setup (to deal with very old duplicates)
  - Accept segments within a receive window

Transmitter



S_{last} oldest unacknowledged byte
S_{recent} highest-numbered transmitted byte
$S_{last}+W_a-1$ highest-numbered byte that can be transmitted
$S_{last}+W_s-1$ highest-numbered byte that can be accepted from the application

Receiver



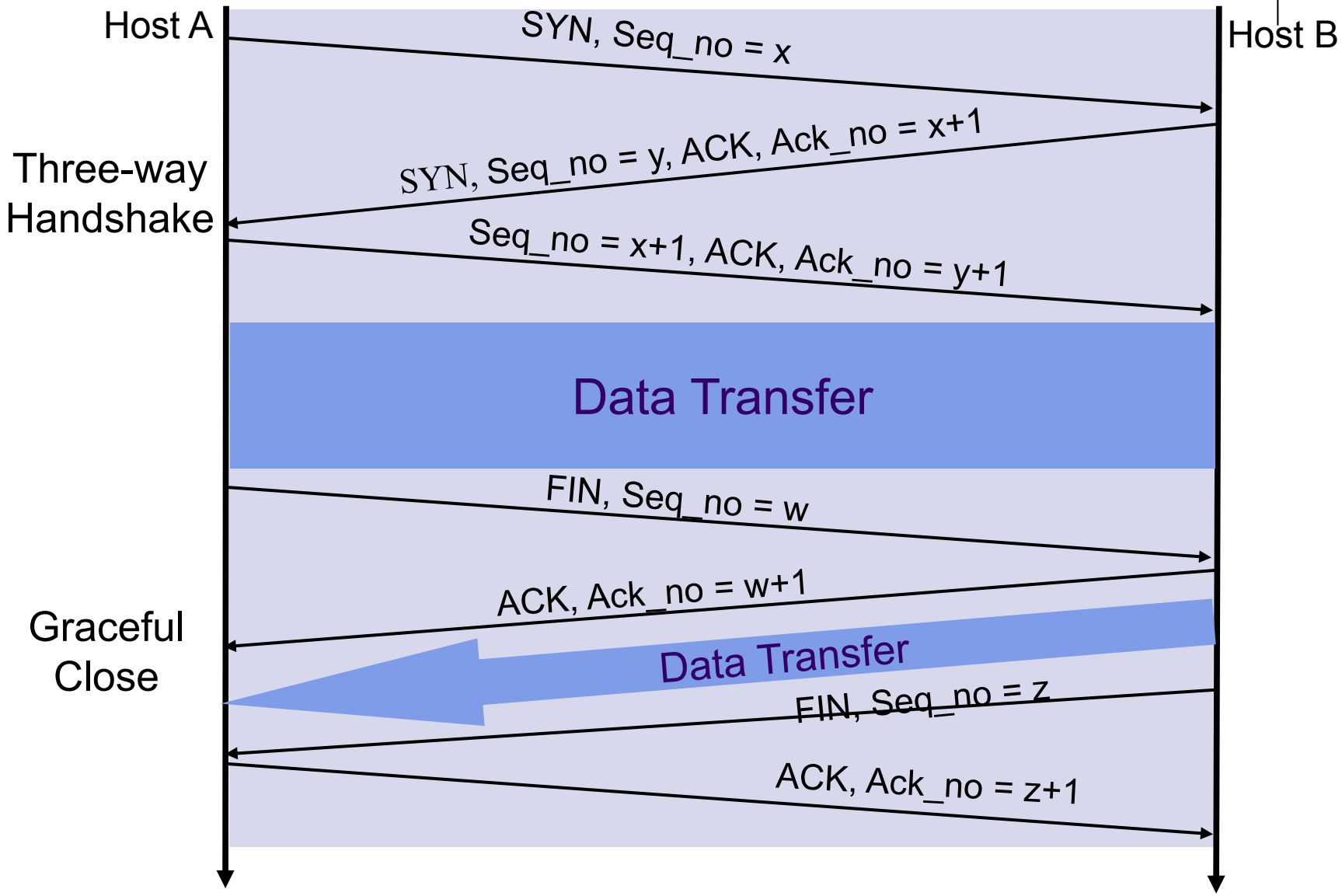
R_{last} highest-numbered byte not yet read by the application
R_{next} next expected byte
R_{new} highest numbered byte received correctly
$R_{last}+W_R-1$ highest-numbered byte that can be accommodated in receive buffer

TCP Connections

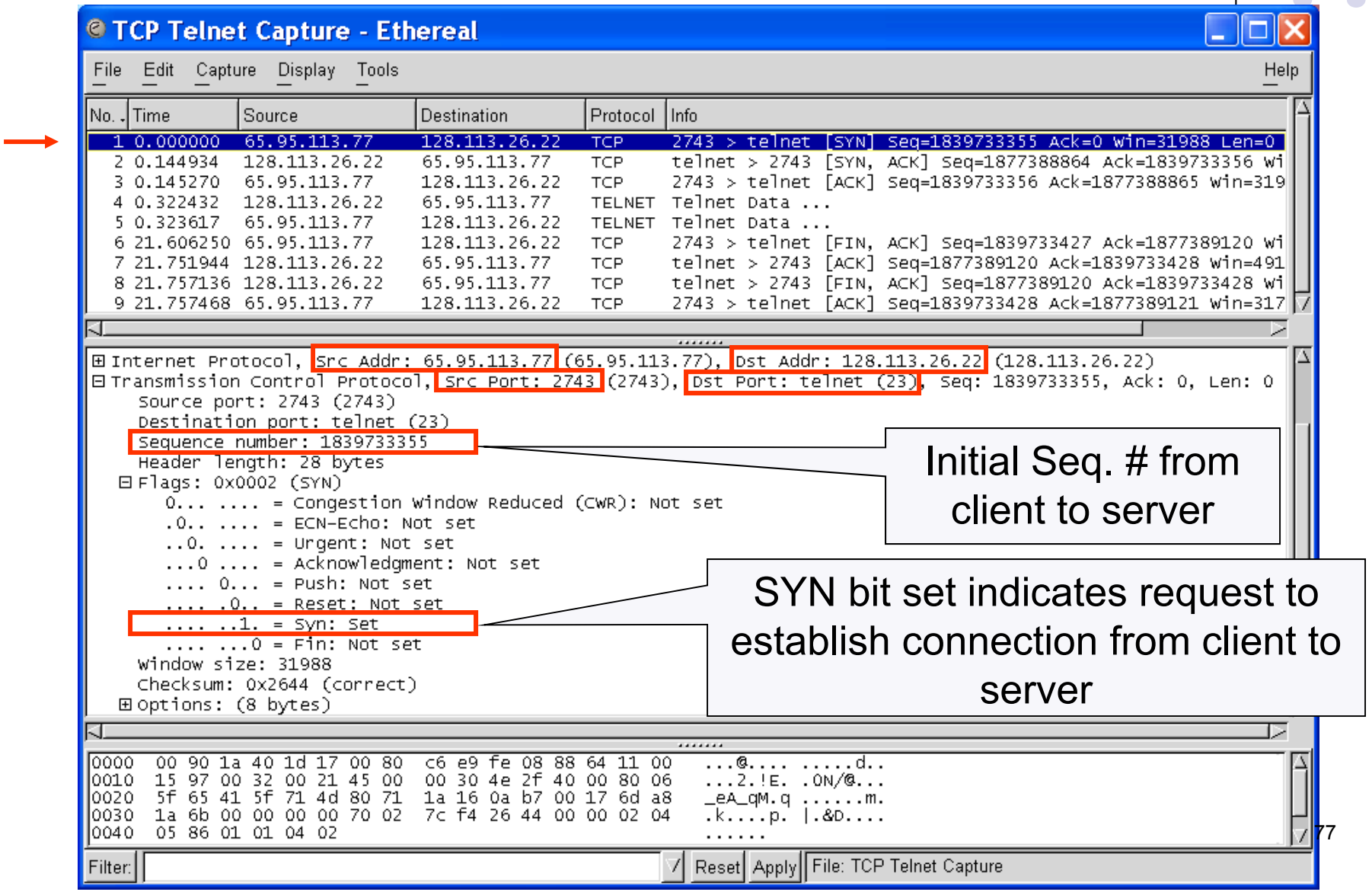


- TCP Connection
  - One connection each way
  - Identified uniquely by Send IP Address, Send TCP Port #, Receive IP Address, Receive TCP Port #
- Connection Setup with Three-Way Handshake
  - Three-way exchange to negotiate initial Seq. #'s for connections in each direction
- Data Transfer
  - Exchange segments carrying data
- Graceful Close
  - Close each direction separately

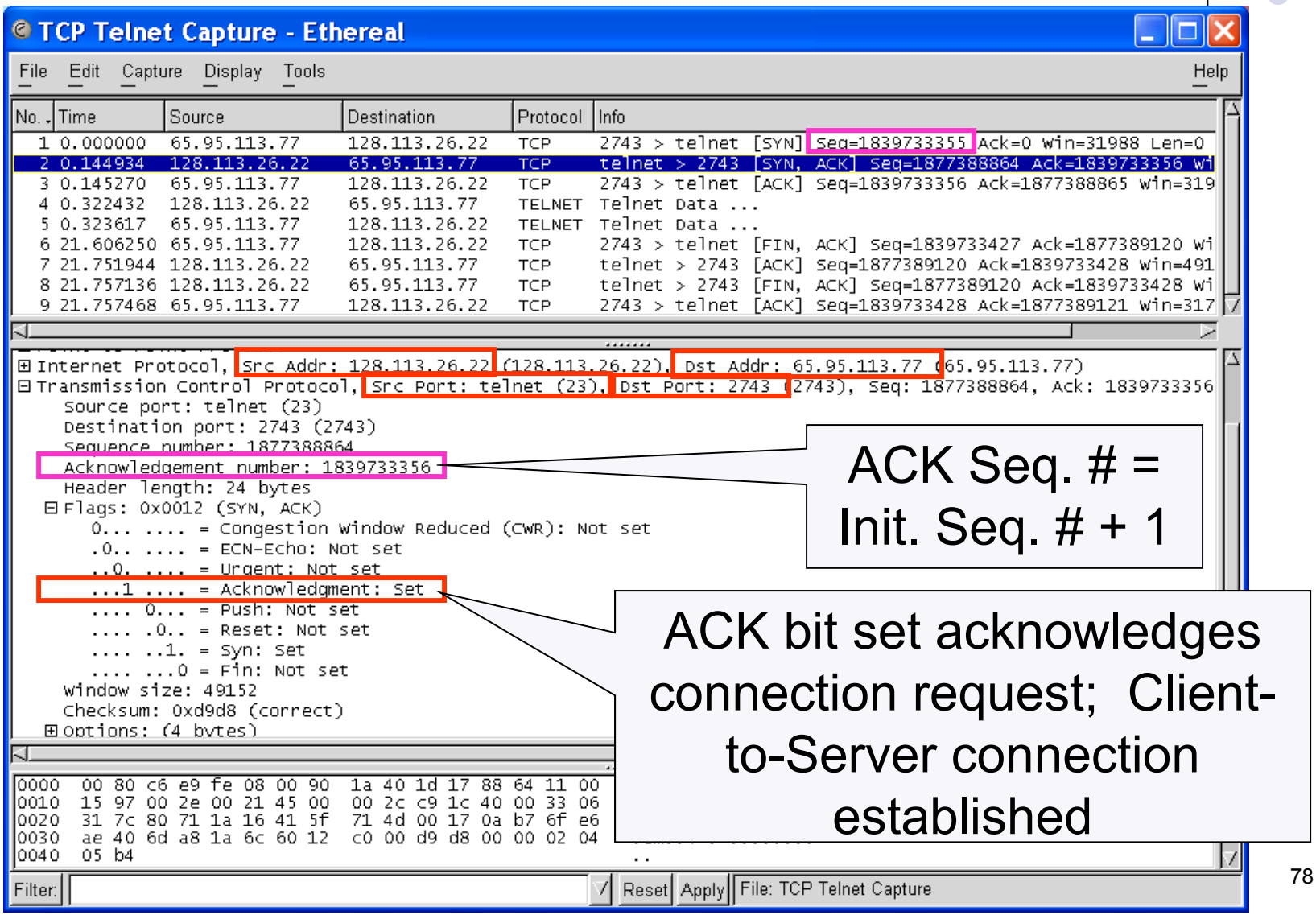
Three Phases of TCP Connection



1st Handshake: Client-Server Connection Request



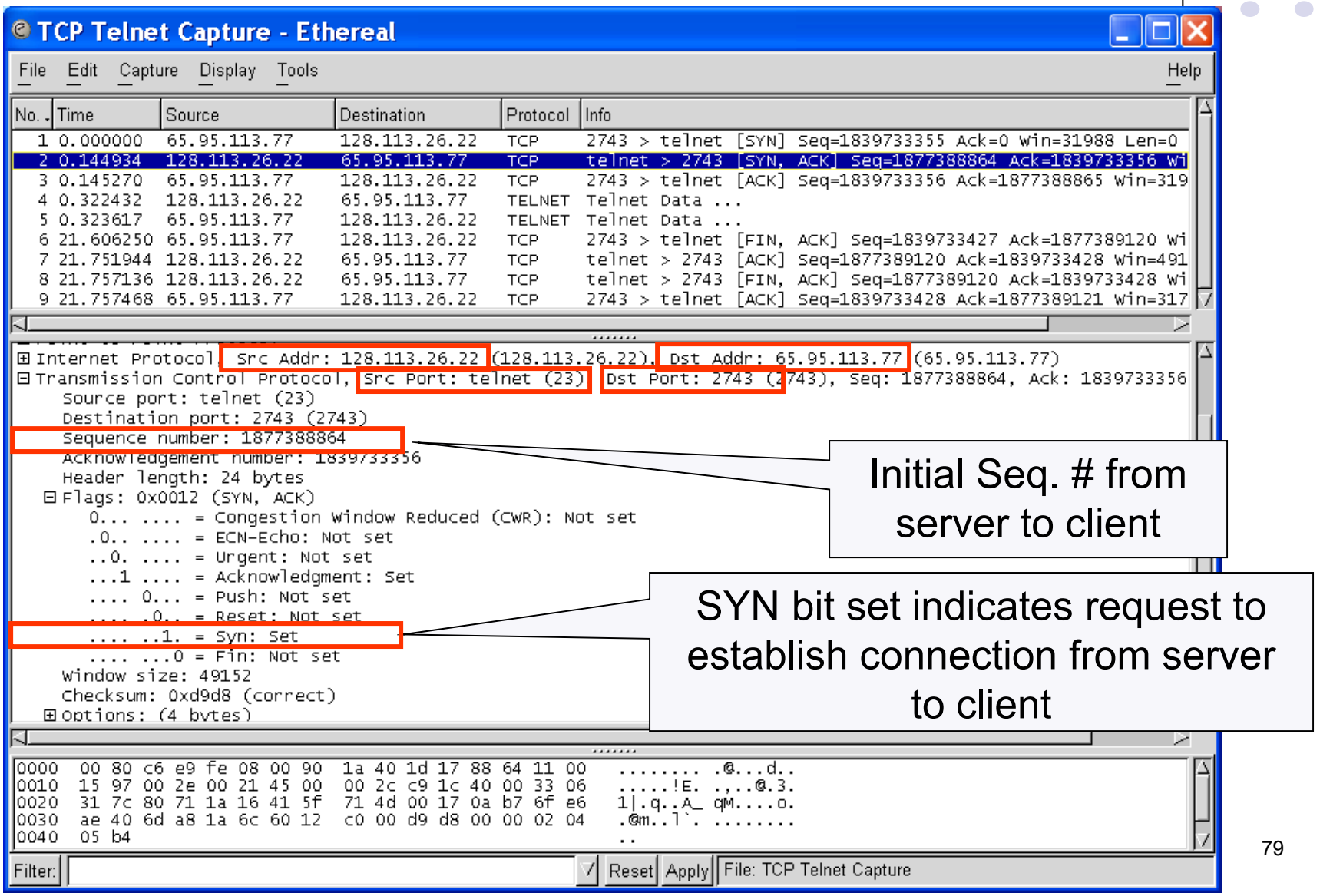
2nd Handshake: ACK from Server



ACK Seq. # = Init. Seq. # + 1

ACK bit set acknowledges connection request; Client-to-Server connection established

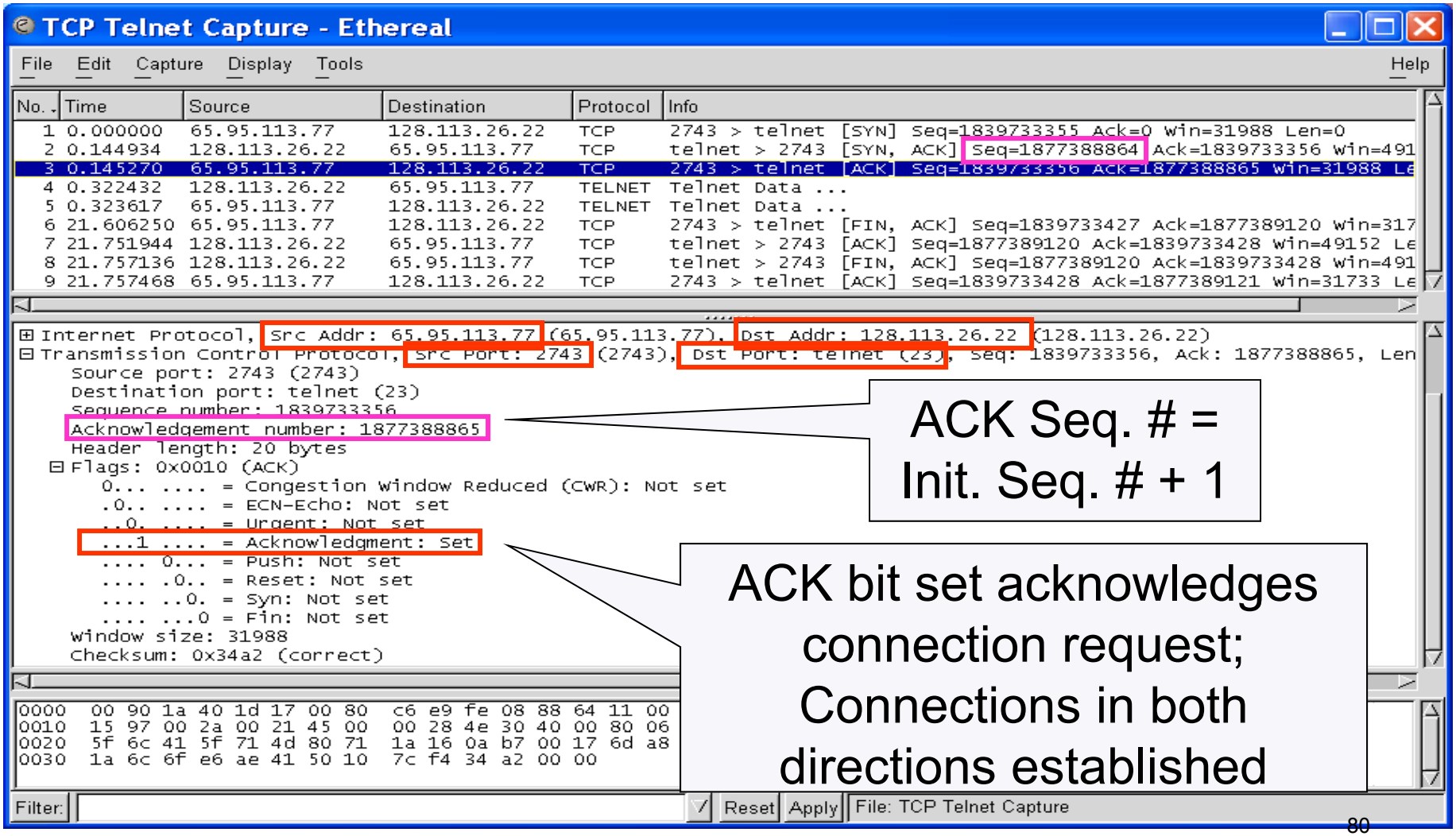
2nd Handshake: Server-Client Connection Request



Initial Seq. # from server to client

SYN bit set indicates request to establish connection from server to client

3rd Handshake: ACK from Client

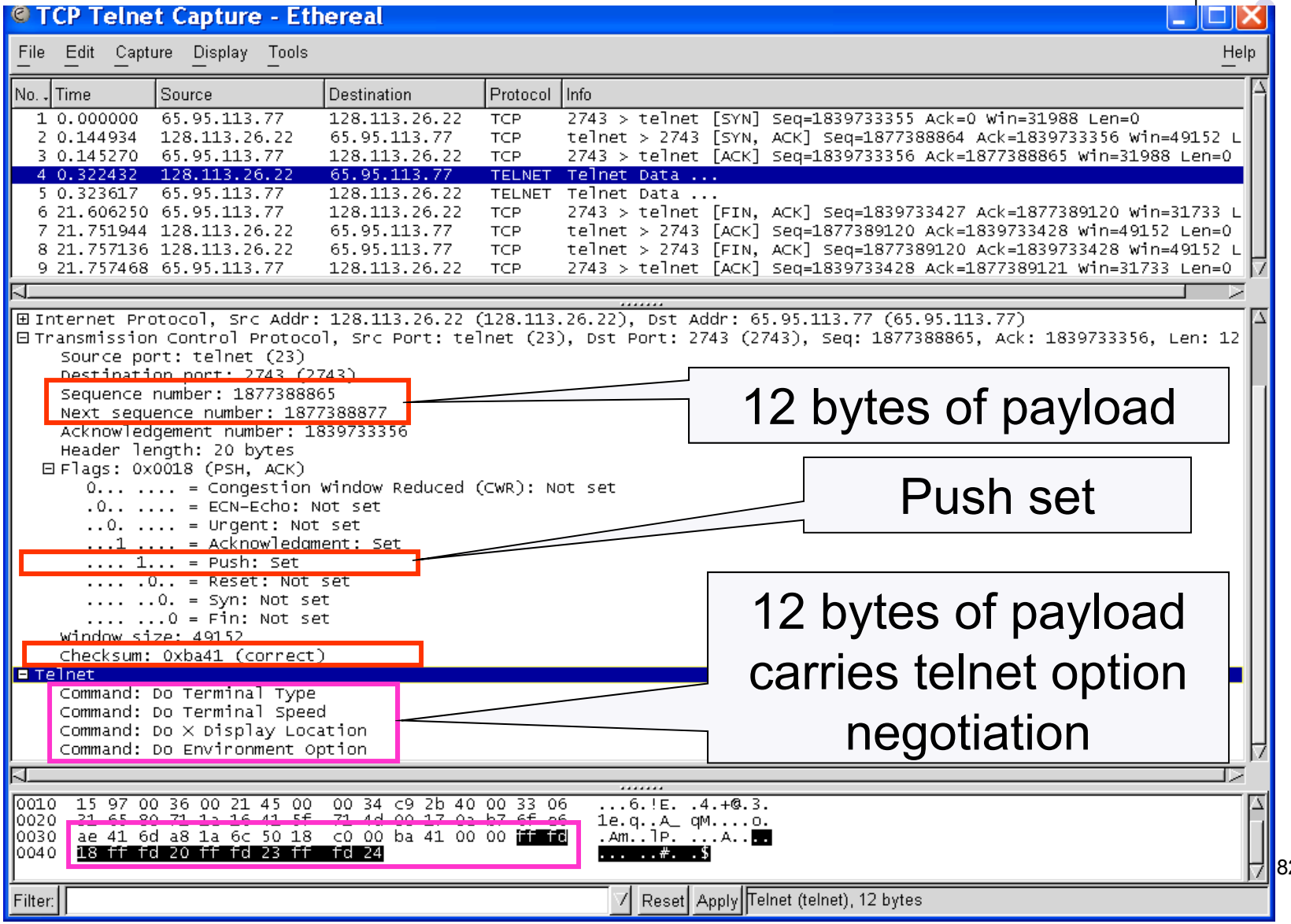


TCP Data Exchange

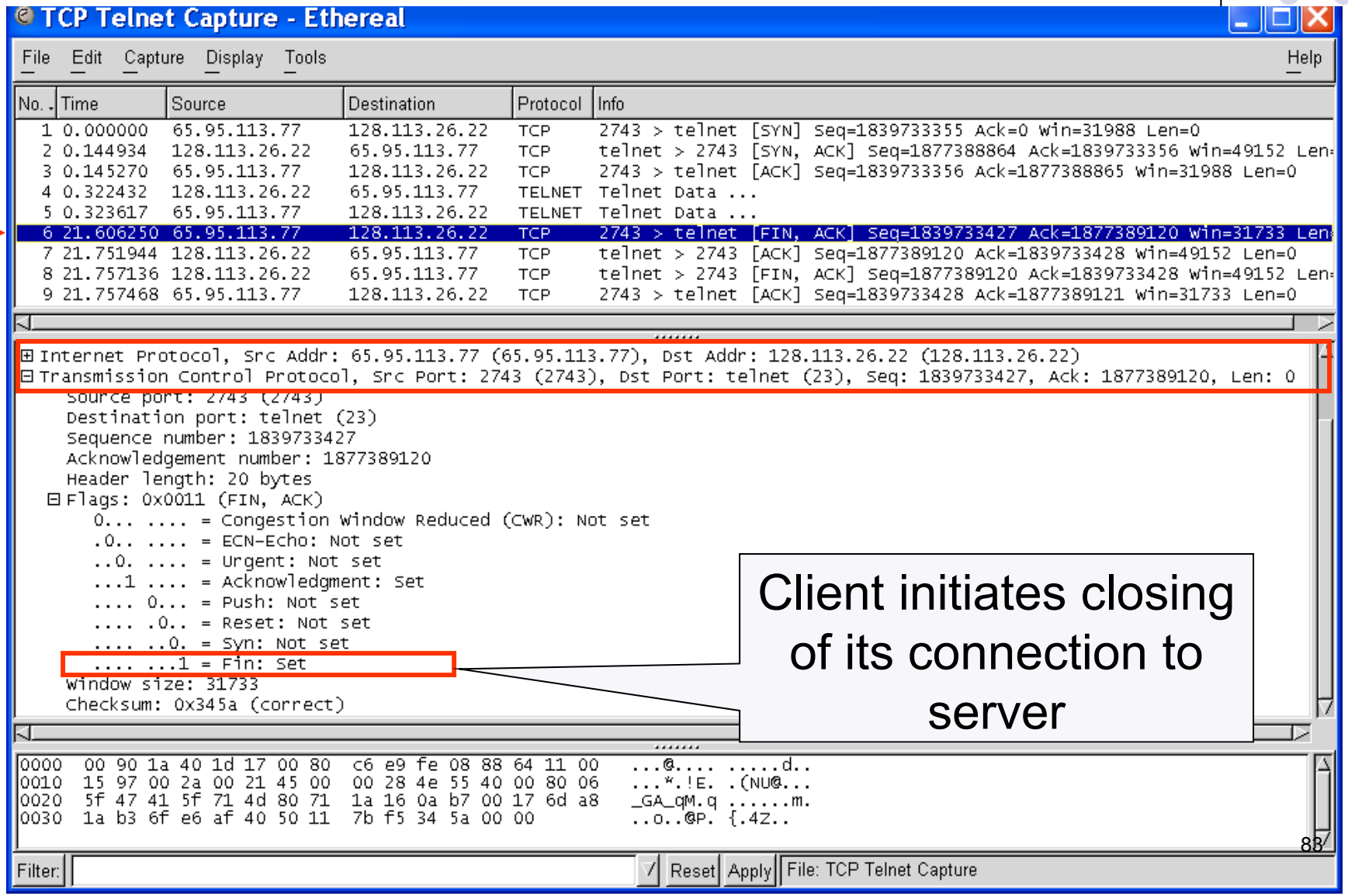


- Application Layers write bytes into buffers
- TCP sender forms segments
  - When bytes exceed threshold or timer expires
  - Upon PUSH command from applications
  - Consecutive bytes from buffer inserted in payload
  - Sequence # & ACK # inserted in header
  - Checksum calculated and included in header
- TCP receiver
  - Performs selective repeat ARQ functions
  - Writes error-free, in-sequence bytes to receive buffer

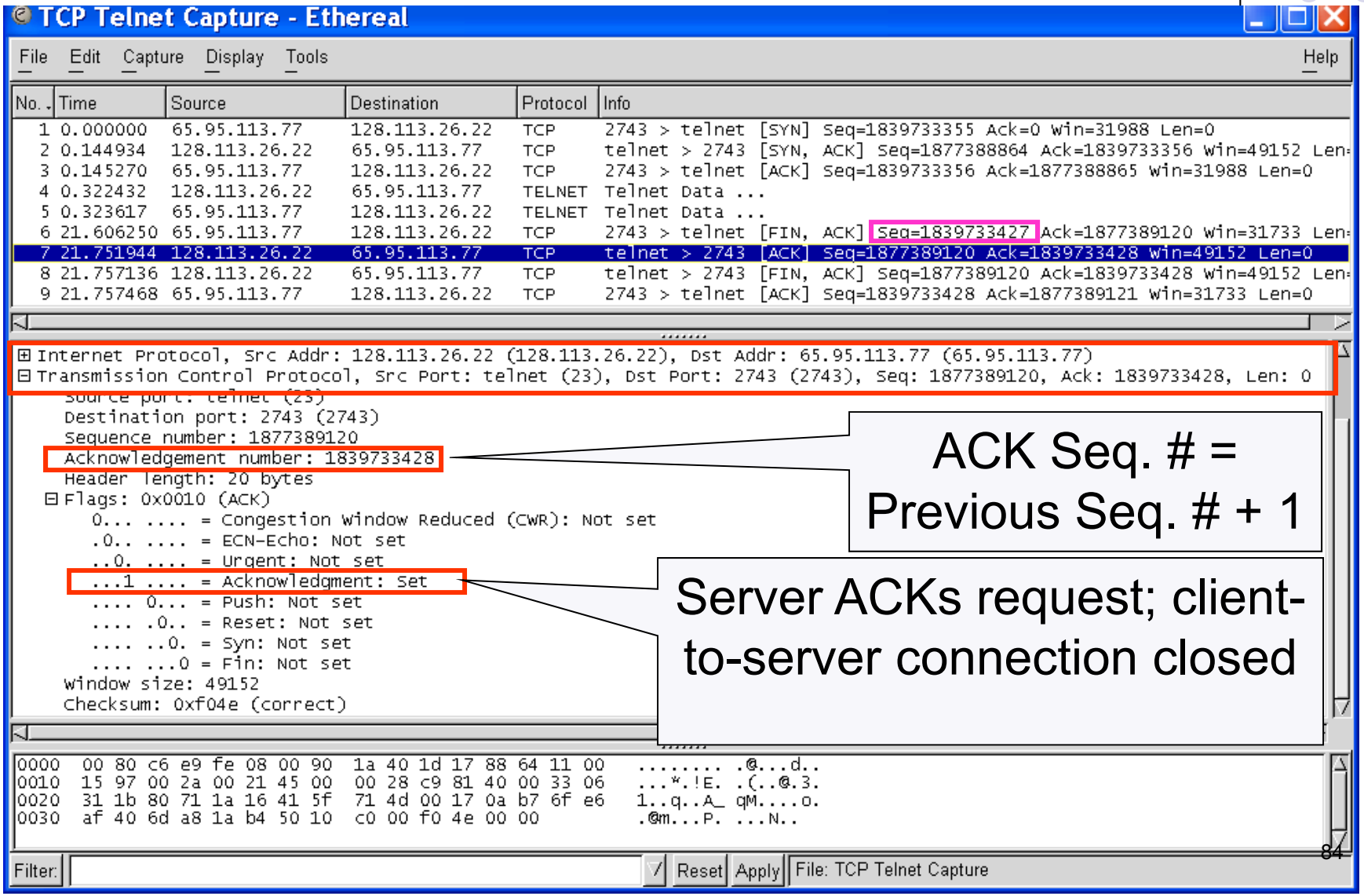
Data Transfer: Server-to-Client Segment



Graceful Close: Client-to-Server Connection



Graceful Close: Client-to-Server Connection



Flow Control

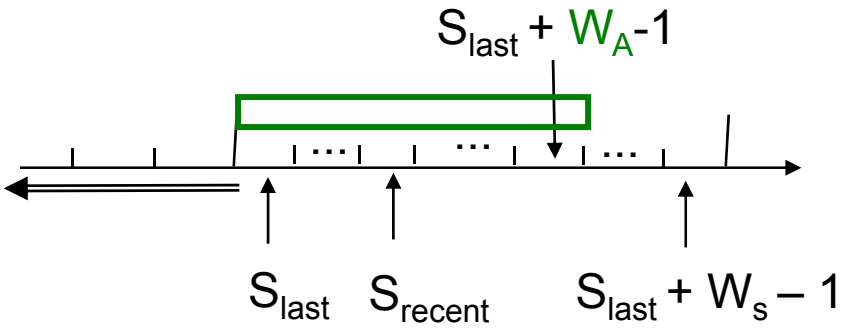
- TCP receiver controls rate at which sender transmits to prevent buffer overflow
- TCP receiver advertises a window size specifying number of bytes that can be accommodated by receiver

$$W_A = W_R - (R_{new} - R_{last})$$

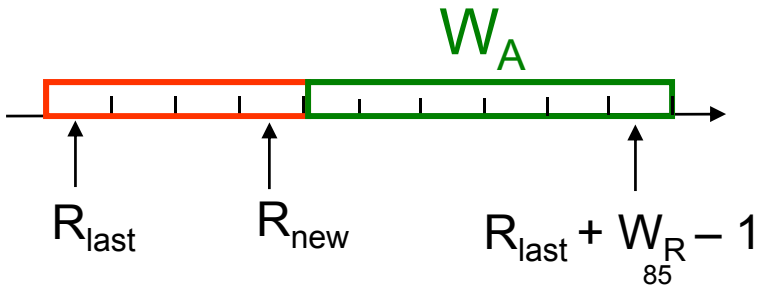
- TCP sender obliged to keep # outstanding bytes below W_A

$$(S_{recent} - S_{last}) \leq W_A$$

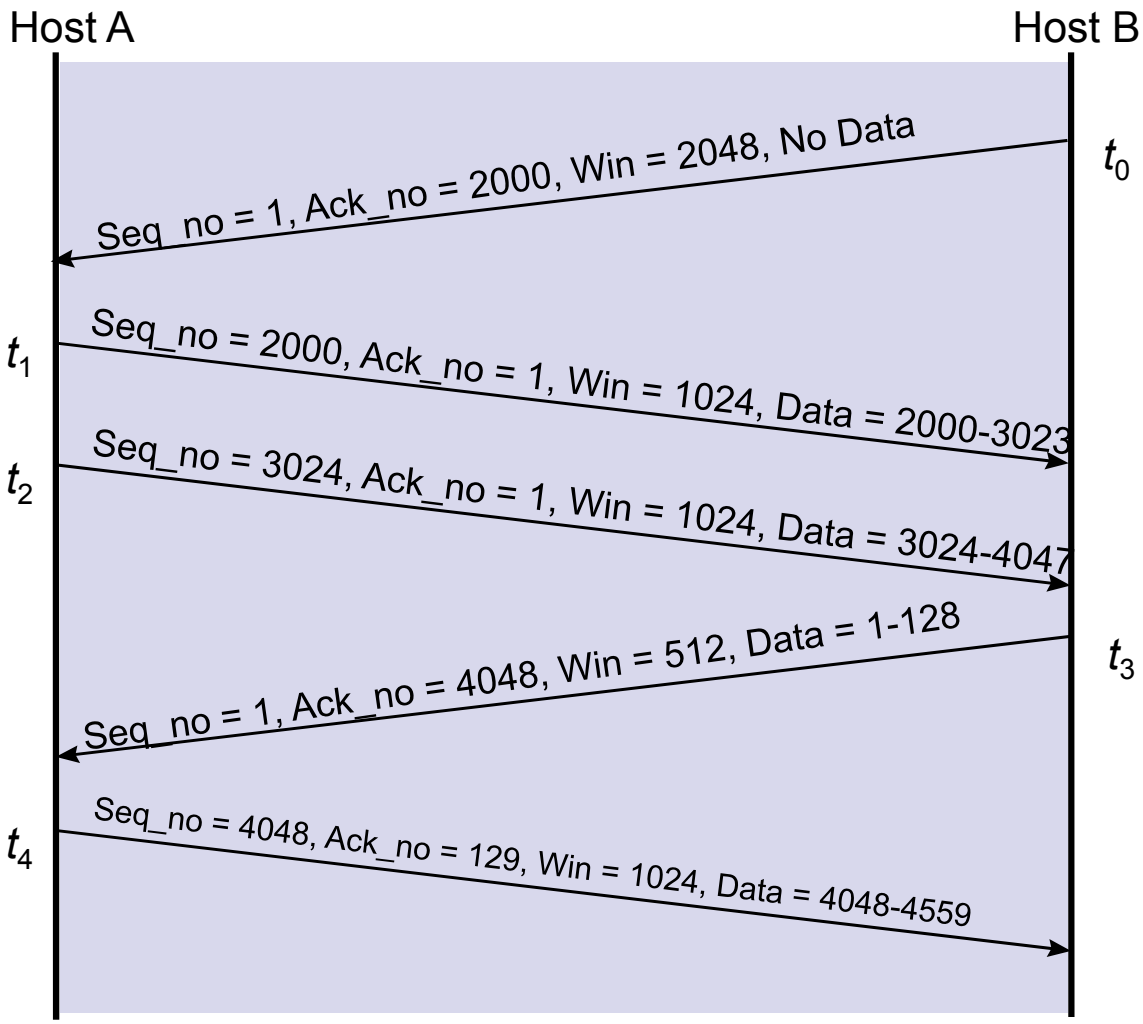
Send Window



Receive Window



TCP window flow control



TCP Retransmission Timeout

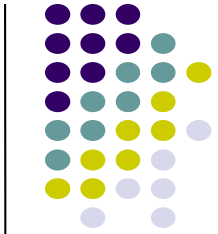


- TCP retransmits a segment after timeout period
  - Timeout too short: excessive number of retransmissions
  - Timeout too long: recovery too slow
  - Timeout depends on RTT: time from when segment is sent to when ACK is received
- Round trip time (RTT) in Internet is highly variable
  - Routes vary and can change in mid-connection
  - Traffic fluctuates
- TCP uses adaptive estimation of RTT
  - Measure RTT each time ACK received: τ_n

$$t_{RTT}(\text{new}) = \alpha\, t_{RTT}(\text{old}) + (1 - \alpha)\, \tau_n$$

  - $\alpha = 7/8$ typical

RTT Variability



- Estimate variance σ^2 of RTT variation
- Estimate for timeout:

$$t_{out} = t_{RTT} + k\ \sigma_{RTT}$$

- If RTT highly variable, timeout increase accordingly
- If RTT nearly constant, timeout close to RTT estimate

- Approximate estimation of deviation

$$d_{RTT}(new) = \beta\, d_{RTT}(old) + (1-\beta)\,|\,\tau_n - t_{RTT}\,|$$

$$t_{out} = t_{RTT} + 4\ d_{RTT}$$