TI III: Operating & Communication Systems

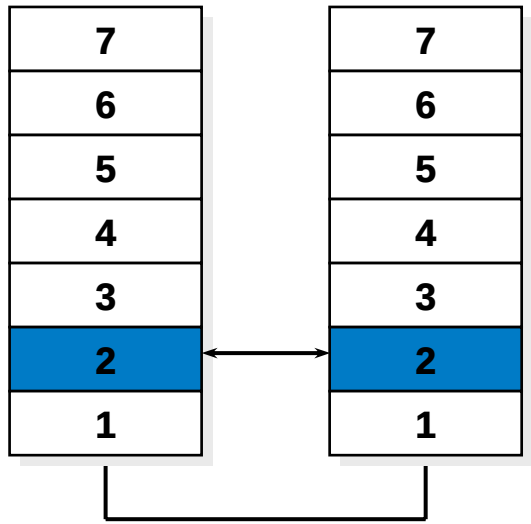
Host-to-Network II

Data Link Layer

Framing, flow control

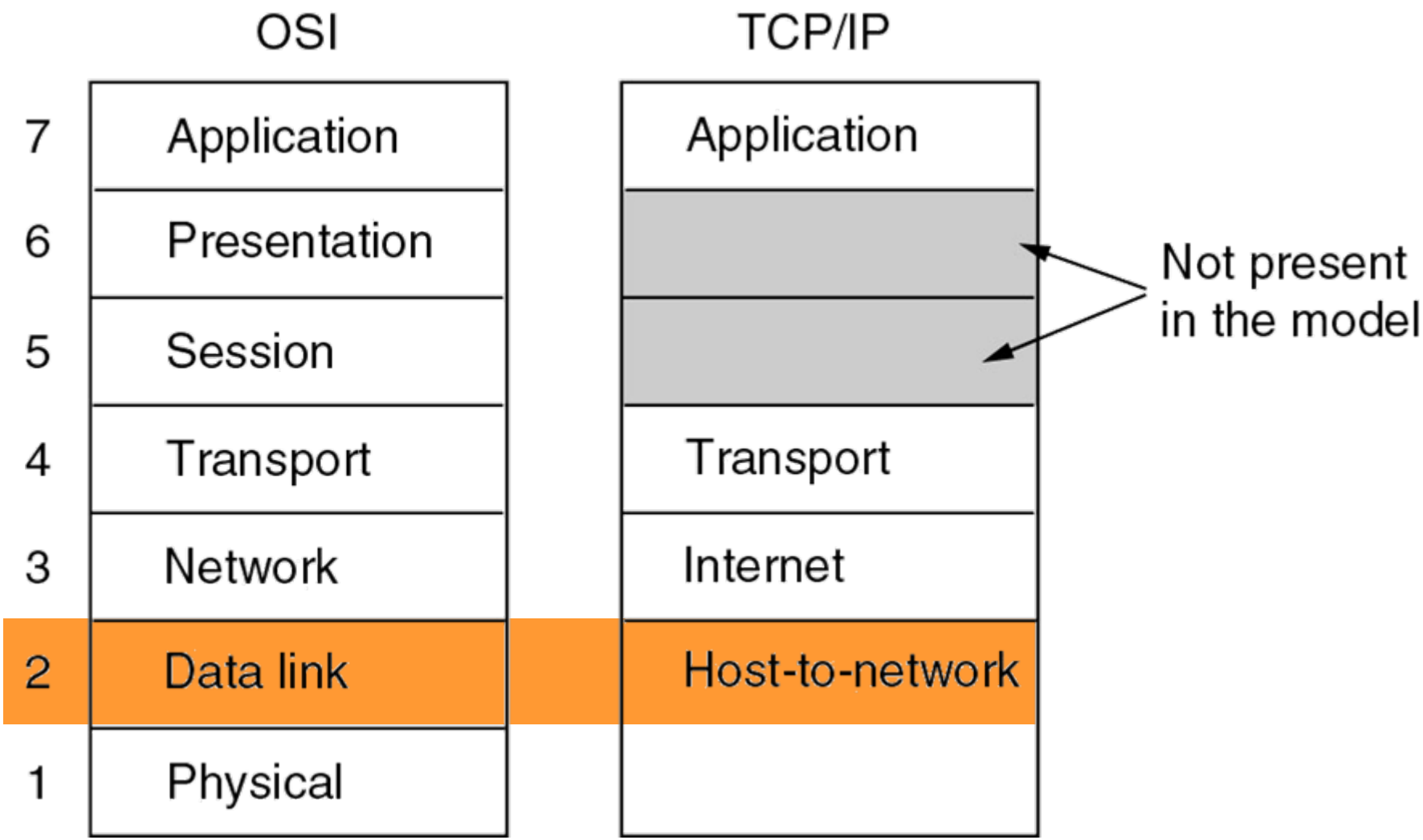
Error detection/correction

PPP



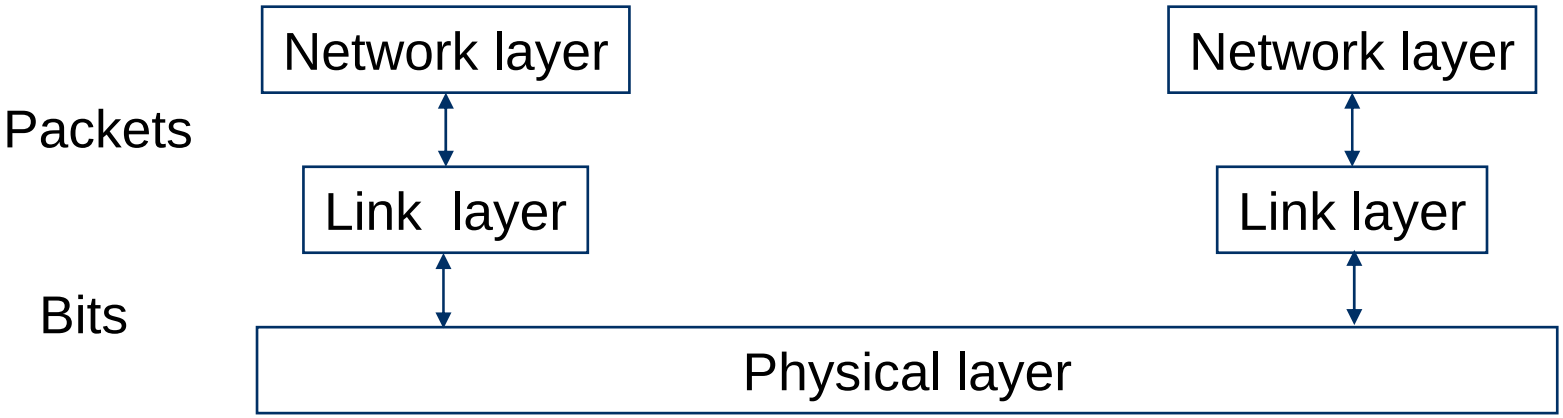
Content (2)

Data Link Layer



The link layer's service I

- Link layer sits on top of the physical layer
  - Can thus use a bit stream transmission service
  - But: this service might have incorrect bits
- Expectations of the higher layer (networking layer)
  - Wants to use either a packet service or, sometimes, a bit stream service (rather unusual)
  - Does not really want to be bothered by errors
  - Does not really want to care about issues at the other end



The link layer's service II

- Transparent communication between two directly connected nodes
- Framing of a physical bit stream into a structure of frames/packets
- Error control: Detection and correction
- Connection setup and release
- Acknowledgement-based protocols
- Flow control

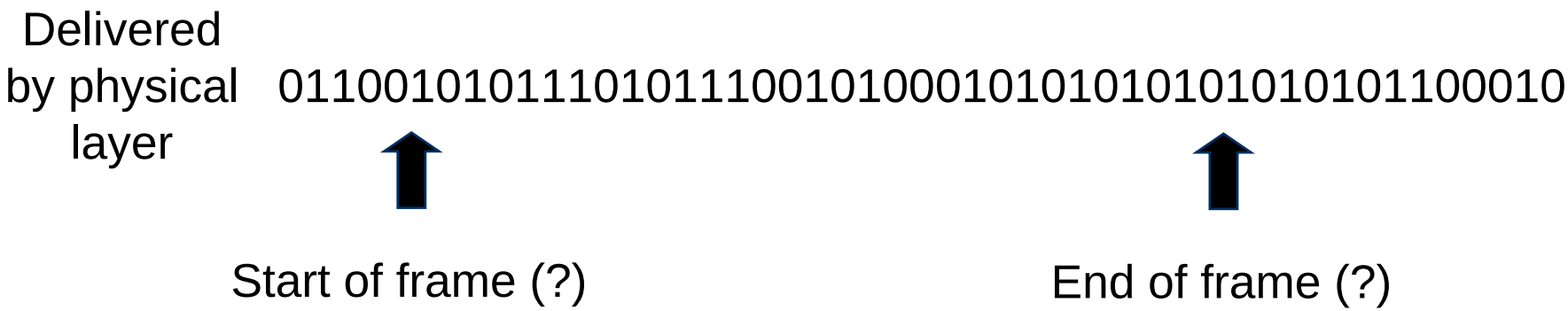
Basic link layer functions – Framing

- How to turn a physical layer's bit stream abstraction into individual, well demarcated ***frames***
  - Usually necessary to provide error control – not obvious how to do that over a bit stream abstraction
  - Frames and packets are really the same thing, only a convention to talk about "frames" in the link layer context

- In addition: Fragmentation & reassembly if network layer packets are longer than link layer packets

Framing

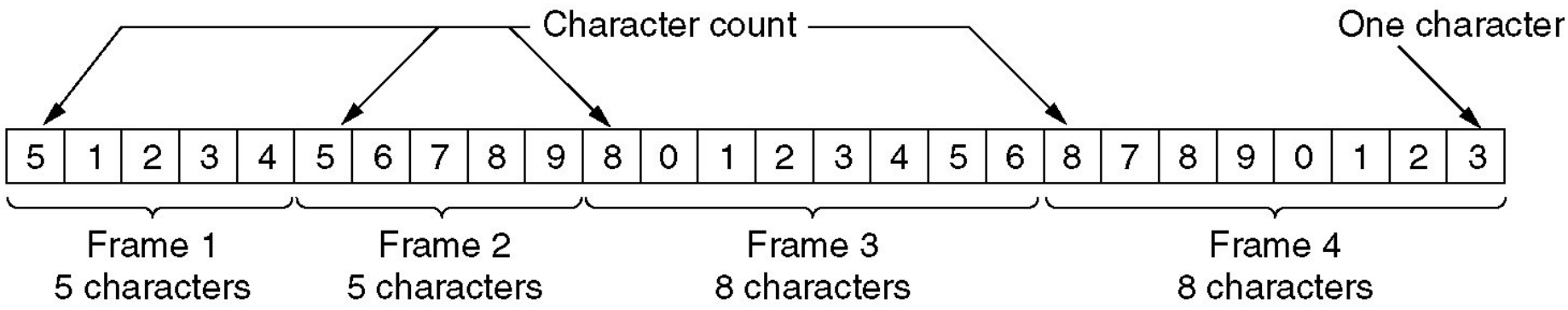
- How to turn a bit stream into a sequence of frames?
  - More precisely: how does a receiver know when a frame starts and when it finishes?



- Note: Physical layer might try to detect and deliver bits when the sender is not actually transmitting anything
  - Receiver still tries to get any information from the physical medium

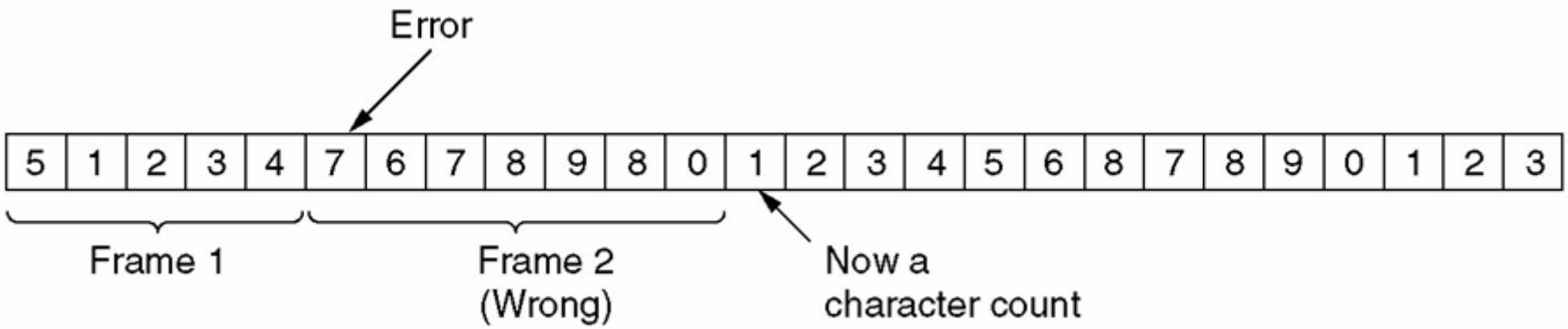
Framing by character/byte counting I

- Idea: Announce the number of bits (bytes, characters) in a frame to the receiver
  - Put this information into the frame
  - Has to be at the beginning of a frame – a frame header



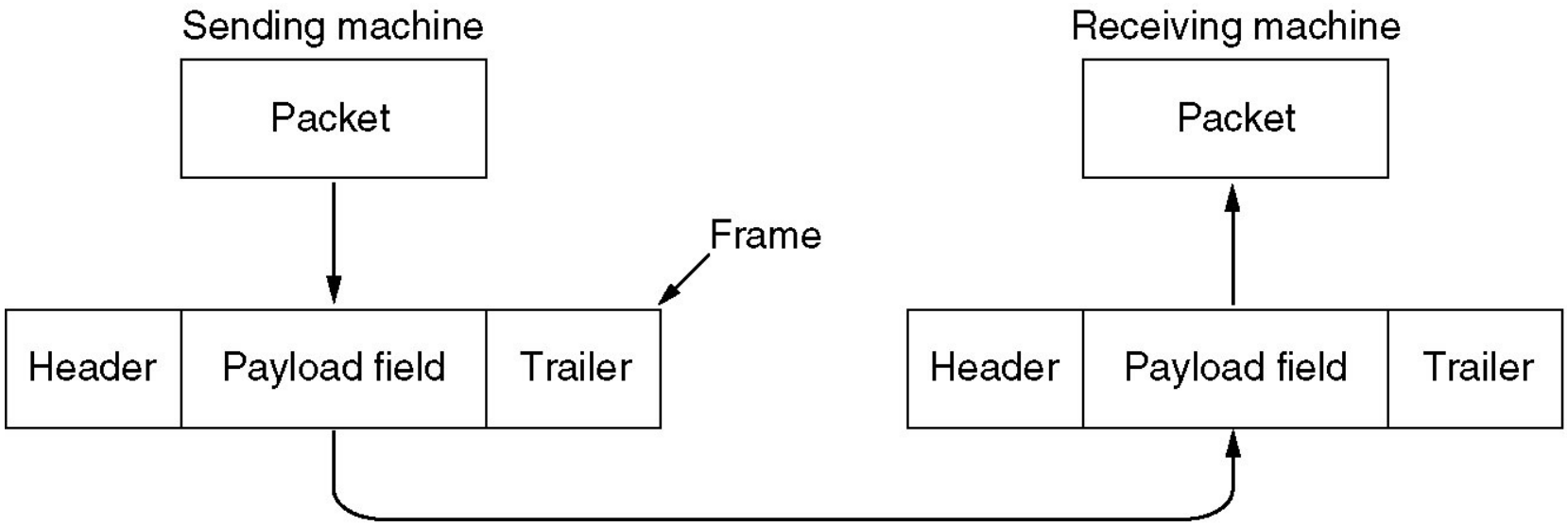
Framing by character/byte counting II

- Problem: What happens if the *count* information itself is damaged during transmission?
  - Receiver will loose frame synchronization and produce different sequence of frames than original one



Basic technique: Put control data into a header

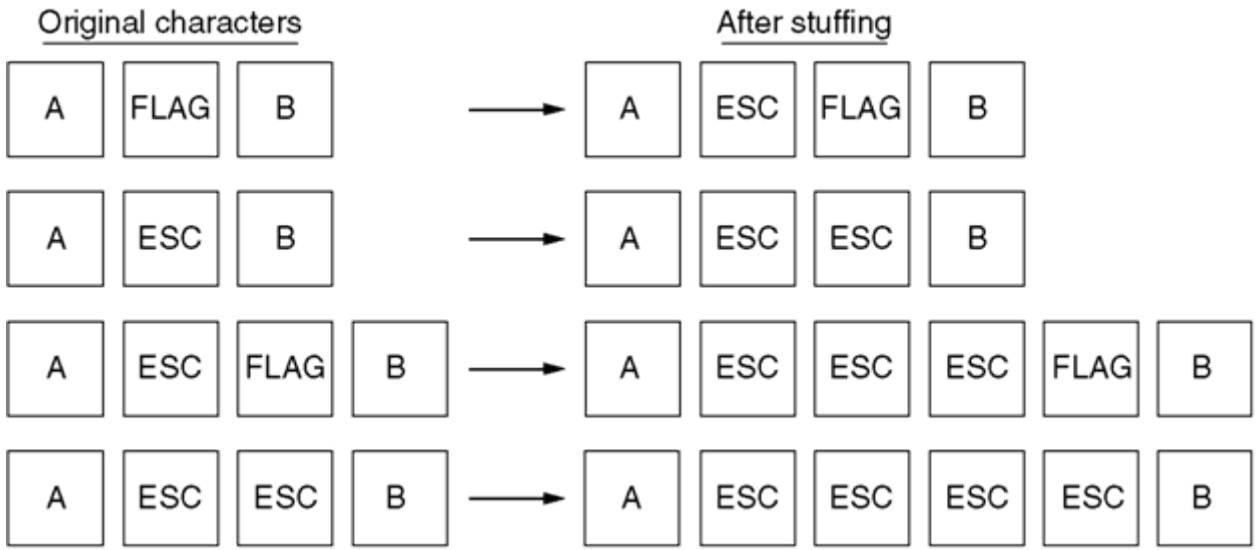
- Albeit "character count" is not a good framing technique, it illustrates an important technique: headers
  - If sender has to communicate administrative or control data to receiver, it can be added to the payload, the actual packet content
  - Usually at the start of the packet; sometimes at the end (a trailer)
  - Receiver uses headers to learn about sender's intention
  - Same thing works for packet headers as well



Framing by flag bytes/byte stuffing

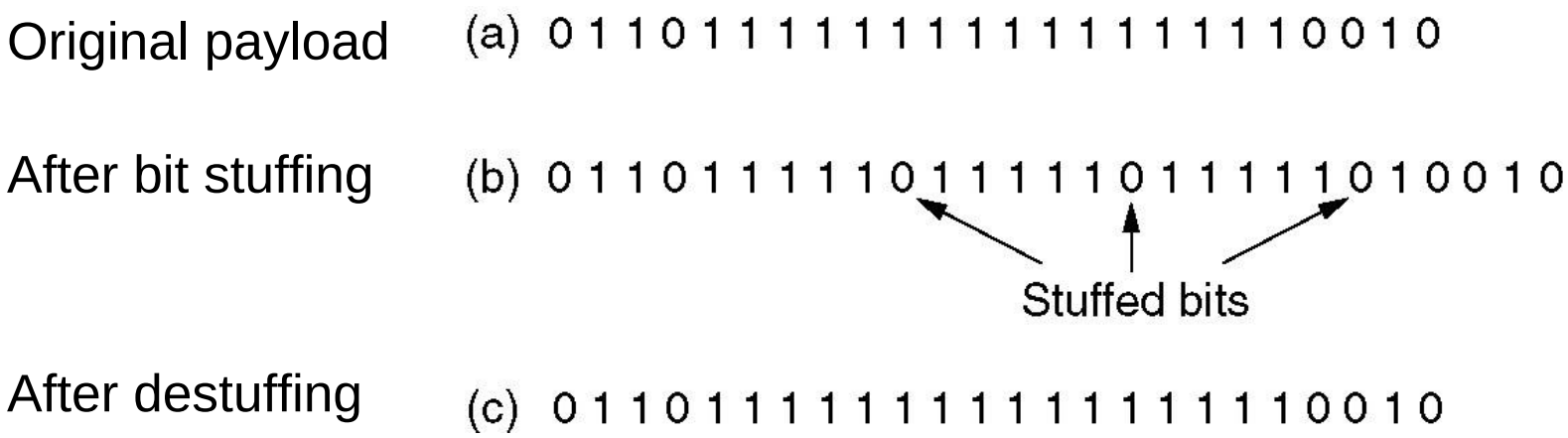
- Use dedicated flag bytes to demarcate start/stop of a frame

- What happens if the flag byte appears in the payload?
  - Escape it with a special control character – byte stuffing
  - If that appears, escape it as well



Framing by flag bit patterns/ bit stuffing

- Byte stuffing is closely tied to characters/bytes as fundamental unit – often not appropriate
- Use same idea, but stick with the bit stream abstraction of the physical layer
  - Use a bit pattern instead of a flag byte – often, 01111110
    - Actually, it IS a flag byte
  - Use bit stuffing
    - Whenever sender sends five 1's in a row, it automatically adds a zero into the bit stream – except in the flag pattern
    - Receiver throws away ("destuffs") any 0 after five 1's

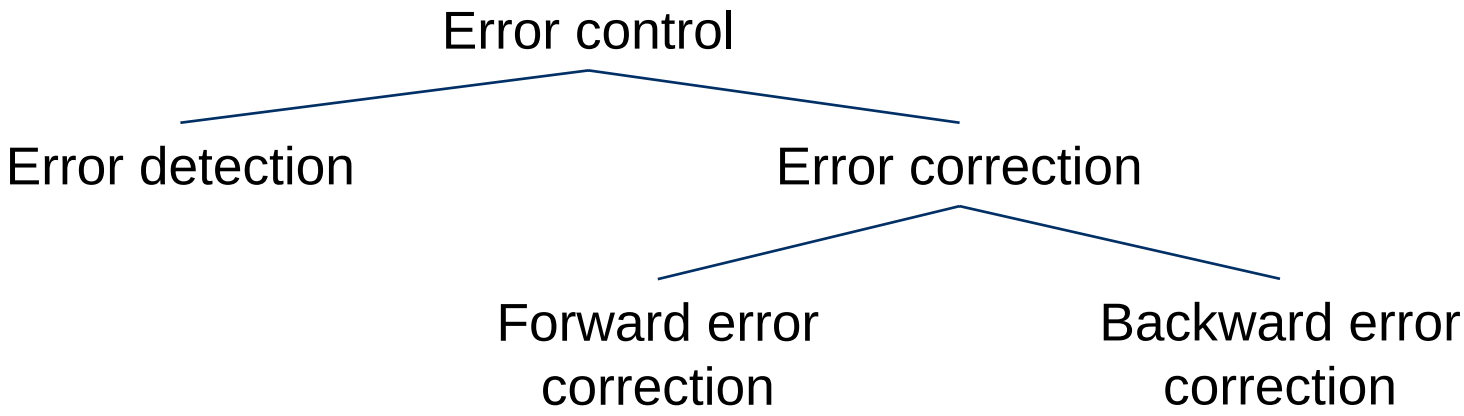


Framing by coding violations

- Suppose the physical layer's encoding rules "bits → signals" still provide some options to play with
  - Not all possible combinations that the physical layer can express are used to express bit patterns
  - Example: Manchester encoding – only low/high and high/low is used
- When "violating" these encoding rules, data can be transmitted – e.g., the start and end of a frame
  - Example: Manchester – use high/high or low/low
    - This drops the self-clocking feature of Manchester, but clock synchronization is sufficiently good to hold for a short while
- Powerful and simple scheme – used e.g. by Ethernet networks
  - But raises questions regarding bandwidth efficiency

Basic link layer functions – Error control

- If desired by the network layer (usually is)
- Usually build on top of frames
- Error detection – are there incorrect bits?
- Error correction – repair any mistakes that have happened?
  - Forward error correction – invest effort ***before*** error happened; try to hide it from higher layers ➔ FEC
  - Backward error correction – invest effort ***after*** error happened; try to repair it ➔ ARQ (Automatic Repeat reQuest)



Error detection: Cyclic Redundancy Check (CRC)

- CRC can check arbitrary, unstructured sequence of bits
- CRC attaches Frame Check Sequence (FCS) as trailer to the checked data
  - Typically calculated ba a feedback shift register in hardware

Attached frame check sequence to error checked data

- Construction of the FCS
  - Treat bit sequence as polynomial with bit determining the coefficients (e.g., 10010 = $1*x^5+0*x^4+0*x^3+0*x^2+1*x^1+0*x^0$)
  - Expand the polynomial with as many 0s as the degree of the generator polynomial is
  - Divide the expanded bit sequence (i.e. the polynomial) by the generator polynomial
  - The FCS is the remainder of the division
  - The receiver again divides the received bit sequence (including the FCS) by the generator polynomial.
  - If no error occurred the must be no remainder

CRC example (sender)

- Transmitted payload: 110011
- Generator polynomial: $x^4 + x^3 + 1$ ⇨ translate into sequence of coefficients: 11001
  - Addition or subtraction equal a simple bitwise XOR
  - Special arithmetic for polynomials modulo 2
- Length of FCS = degree of generator polynomial = 4

- Calculation of FCS:

```
11 0011 0000 ÷ 1 1001 = 10 0001
11 001
00 0001 0000
      1 1001
        1001 = remainder
```

- Transmitted bit sequence: 11 0011 1001.

CRC example (receiver)

- Reception of a correct bit sequence:

```
11 0011 1001 ÷ 1 1001 = 10 0001
11 001
00 0001 1001
      1 1001
      0 0000  = remainder
```

  - No remainder, thus the received bits *should* be error free

- Reception of a erroneous bit sequence:

```
11 1111 1000 ÷ 1 1001 = 10 1001
11 001
00 1101 1
   1100 1
   0001 0000
      1 1001
      0 1001  = remainder ≠ 0
```

  - There is a remainder unequal 0, thus there was definitely a transmission error

Performance of CRC

A CRC can detect the following errors:

- All single bit errors
- All double bit errors (if $(x^k + 1)$ is not dividable by the generator polynomial for $k \leq$ frame length)
- All errors affecting an odd number of bits (if (x+1) is a factor of the generator polynomial)
- All error bursts of length $\leq$ degree of generator polynomial

Internationally standardized generator polynomials

- CRC-12 $= x^{12} + x^{11} + x^3 + x^2 + x + 1$
- CRC-16 $= x^{16} + x^{15} + x^2 + 1$
- CRC-CCITT $= x^{16} + x^{12} + x^5 + 1$

CRC-16 and CRC-CCITT detect

- All single and double errors
- All errors affecting an odd number of bits
- All error bursts of length ≤ 16
- 99,997 % of all error bursts of length 17
- 99,998 % of all error bursts of length ≥ 18

Forward Error Correction (FEC) I

- CRC uses redundancy for error detection only. FEC uses redundancy to correct errors.
- A simple FEC scheme: repeat data several times, then use a majority decision.
- More advanced: XOR (see example), Reed-Solomon, …

- Example:
  - Send the following packets:

    0101 - P1
    1111 - P2
    0000 - P3
  - Calculate a fourth packet using XOR:

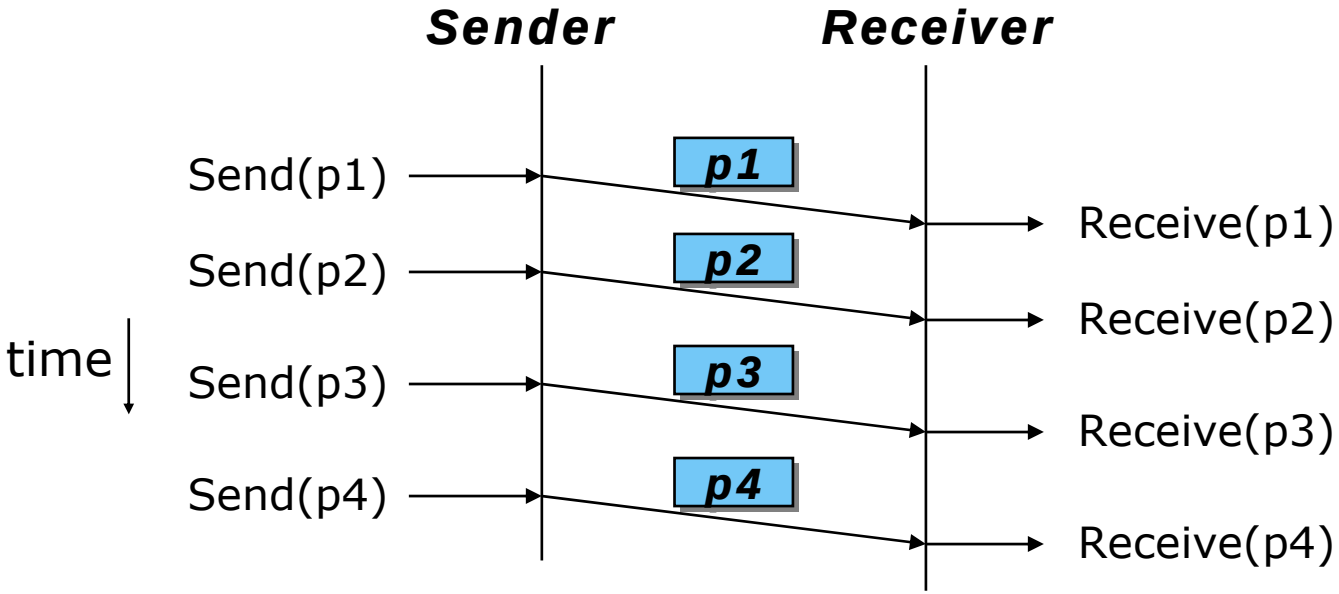
    1010 - P4
  - Send all four packets to the receiver.

Forward Error Correction (FEC) II

- For the receiver it is sufficient to receive any three out of the four packets.
- Reconstruct the missing packet based on the received packets:
  - P1 is lost:
    1111
    0000
    1010
    0101 -> P1
  - P2 is lost:
    0101
    0000
    1010
    1111 -> P2
  - P3 is lost:
    0101
    1111
    1010
    0000 -> P3
- However, the receiver still has to know which packet was lost ...

Data transmission in an ideal world

- Sender/receiver are always ready to send/receive
- Receiver can handle amount of incoming data
- No errors occur that cannot be handled by FEC

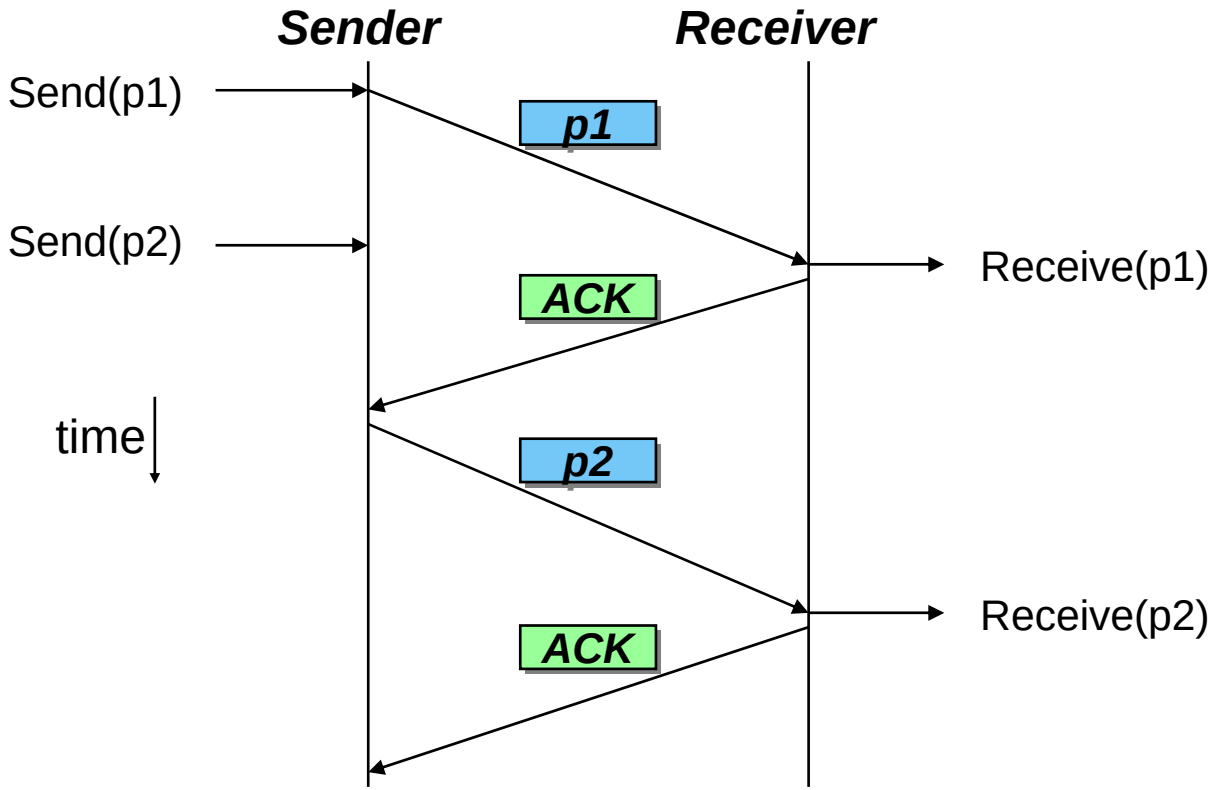


- What happens if packets are lost?
- What happens if the sender floods the receiver?
  - Imagine a web server sending data to a mobile phone…

A first very simple solution: Stop-and-wait

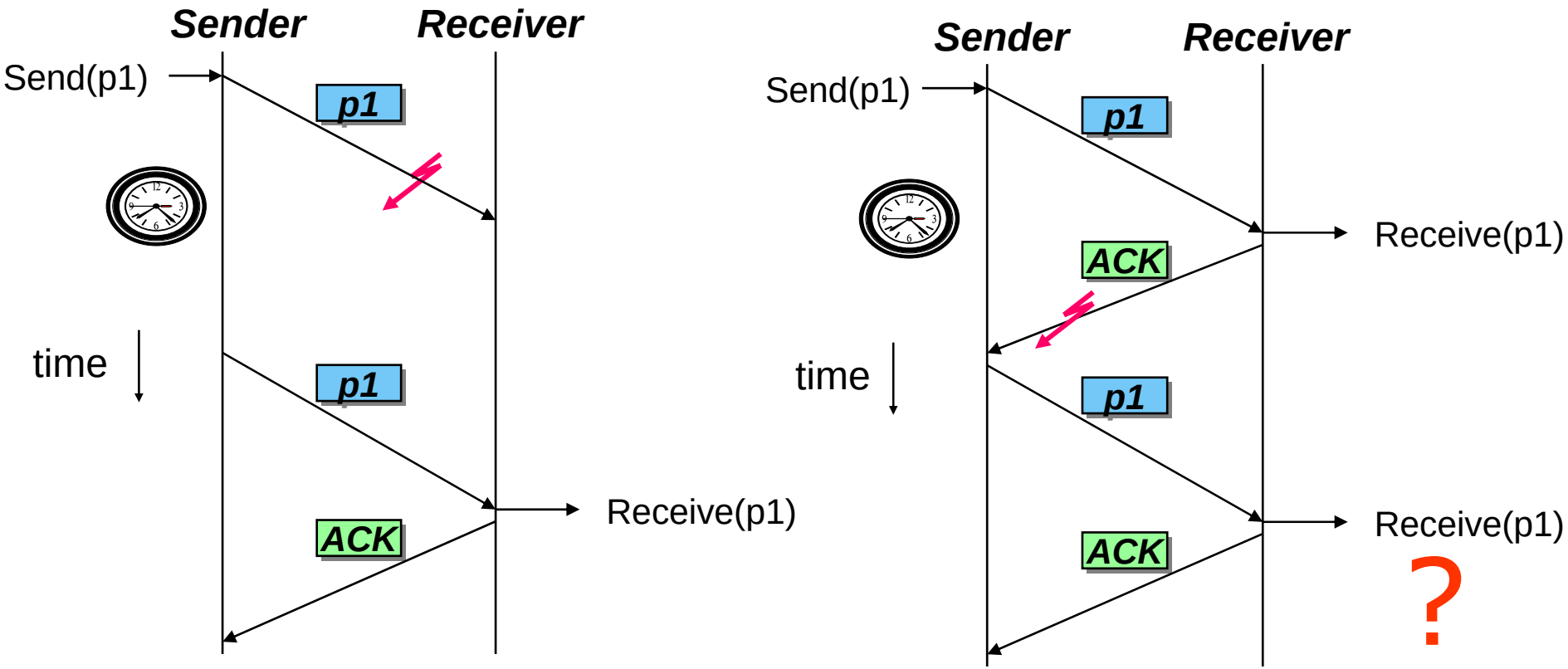
- Concentrate on a single packet
- Receiver acknowledges the correct reception of the packet
- The sender has to wait for that acknowledgement before continuing with the next packet

  - Thus no overloading of the receiver possible!
  - This is basic flow control



Problems of the simple solution

- What happens if errors occur?
  - Lost packet? Lost acknowledgement? Is there a difference?
- Basic solution: ARQ (Automatic Repeat reQuest)

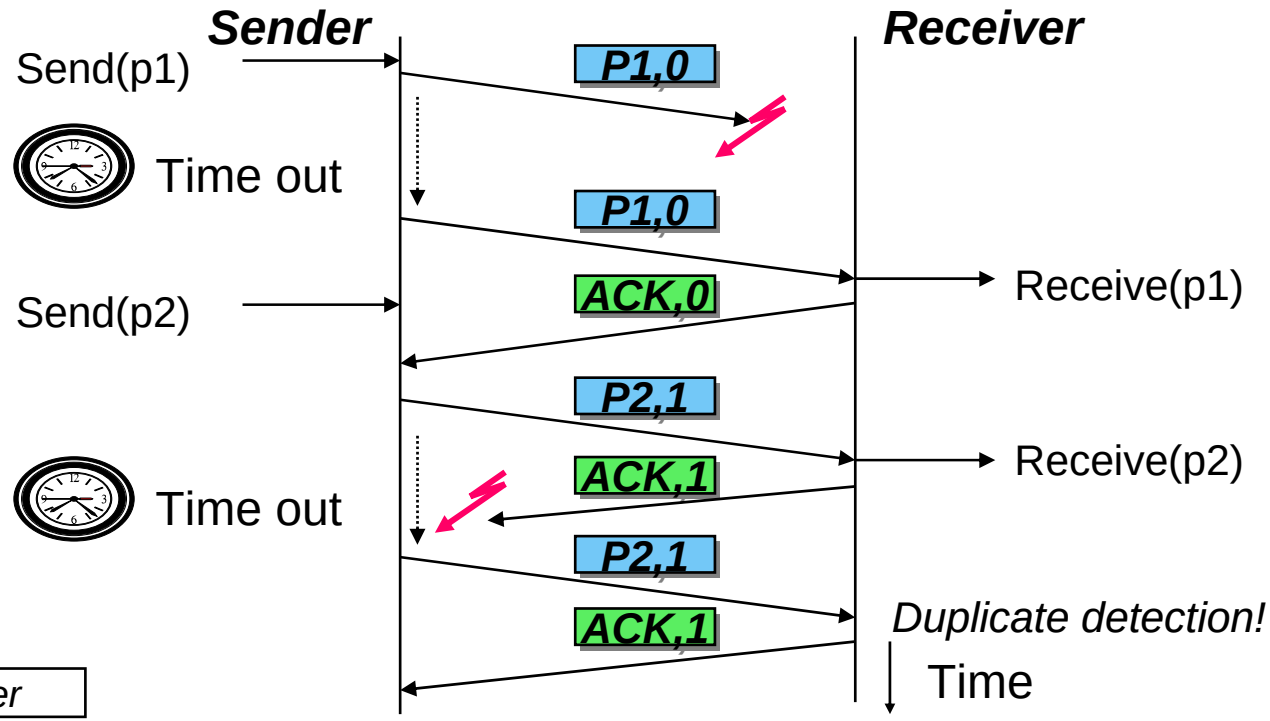


Problem of the second version

- Sender cannot distinguish between a lost packet and a lost acknowledgement ➔ has to re-send the packet
- Receiver cannot distinguish between a new packet and a redundant copy of an old packet ➔ Additional information is needed
- Put a ***sequence number*** in each packet, telling the receiver which packet it is
  - Sequence numbers as ***header information*** in each packet
  - Simplest sequence number: 0 or 1
- Needed in packet & acknowledgement
  - In ACK, one convention: send the sequence number of the last correctly received packet back
    - Also possible: send sequence number of next expected packet
    - Be aware: some protocols count bytes instead of packets

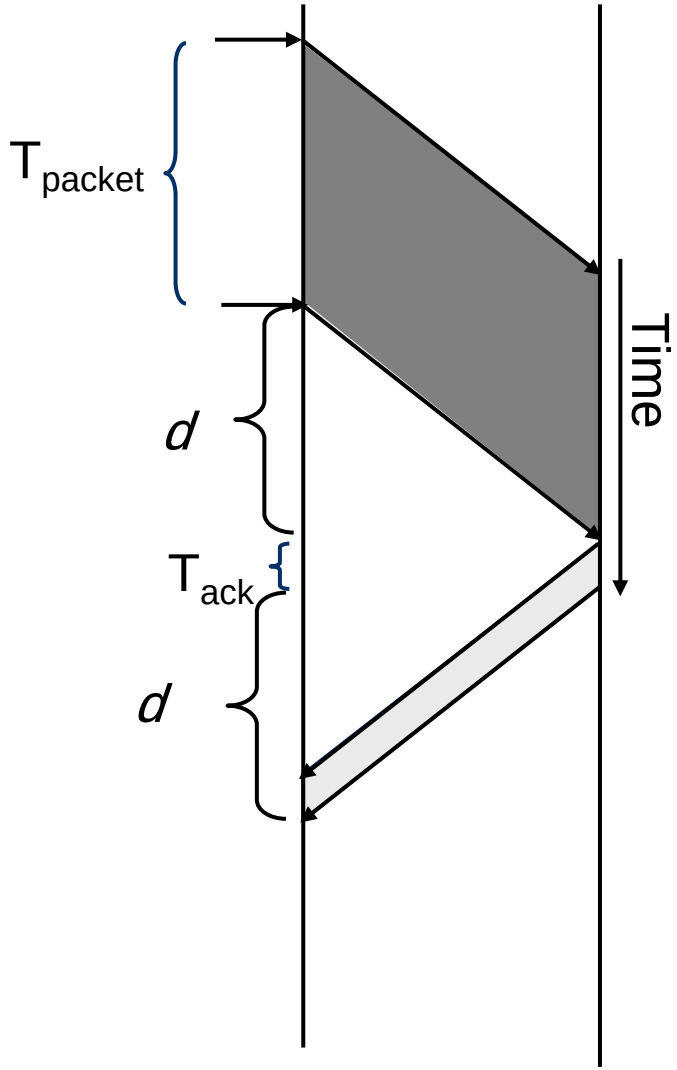
Alternating bit protocol

- Simple, but reliable protocol over noisy channels
- Uses 0 and 1 as sequence numbers, ARQ for retransmission
- Simple form of flow control (here combined with error control, other protocols separate these functions)



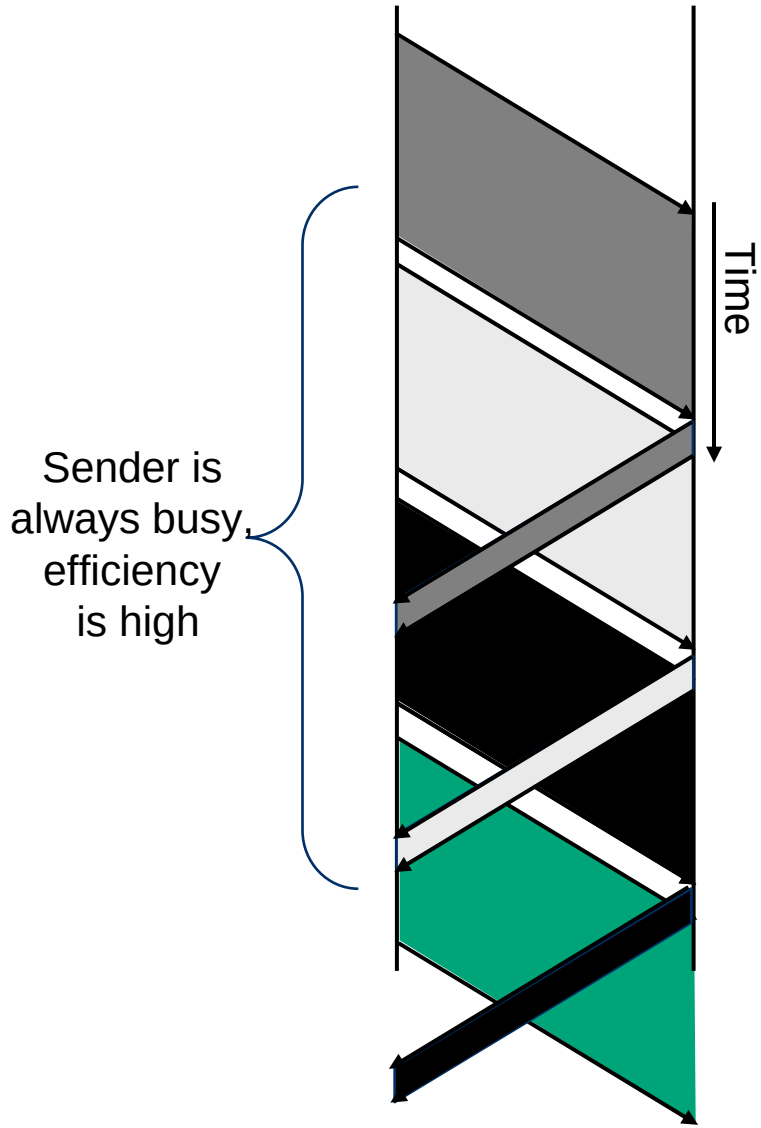
Alternating bit protocol – Efficiency

- Efficiency η: depends on delay and bandwidth
  - Defined as the ratio of time during which the sender sends new information (assuming an error-free channel in the simplest case; error-considerations make efficiency discussions difficult)
  - $\eta = T_{packet} / (T_{packet} + d + T_{ack} + d)$
- Efficiency of simple alternating bit protocol is low when delay is large compared to data rate
  - Bandwidth-delay product



Improving efficiency – have more "outstanding" packets

- Inefficiency of alternating bit in large bandwidth-delay situations is owing to not exploiting "space" between packet and acknowledgement
- Always sending packets results in high efficiency
  - More packets are "outstanding" = sent, but not yet acknowledged
  - "pipelining" of packets
- But not feasible with a single bit as sequence number
- ➔ Need larger sequence number space!
  - It also needs – ideally – some full-duplex support
  - How to live without full-duplex?



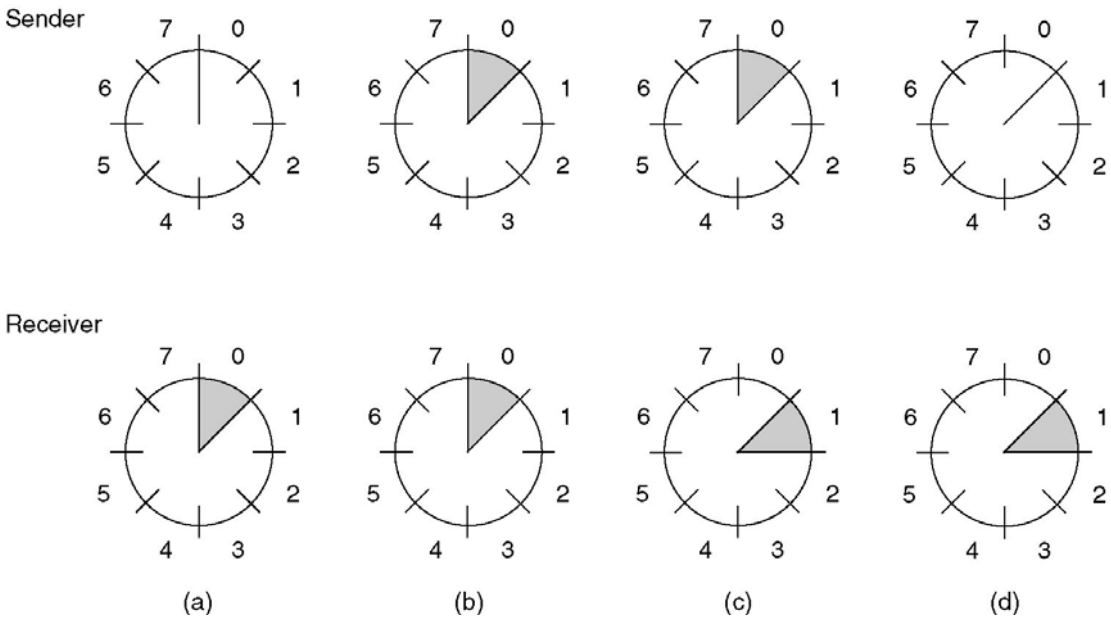
Sliding windows to handle multiple outstanding packets

Freie Universität Berlin

- Introduce a larger sequence number space
  - Say, n bits or 2^n sequence numbers
- Not all of them may be allowed to be used simultaneously
  - Recall alternating bit case: 2 sequence numbers, but only 1 may be "in transit"
- Use sliding windows at both sender and receiver to handle these numbers
  - Sender: sending window – set of sequence numbers it is allowed to send at given time
  - Receiver: receiving window – set of sequence numbers it is allowed to accept at given time
  - May be fixed in size or adapt dynamically over time
  - Window size corresponds to flow control

Sliding window – simple example

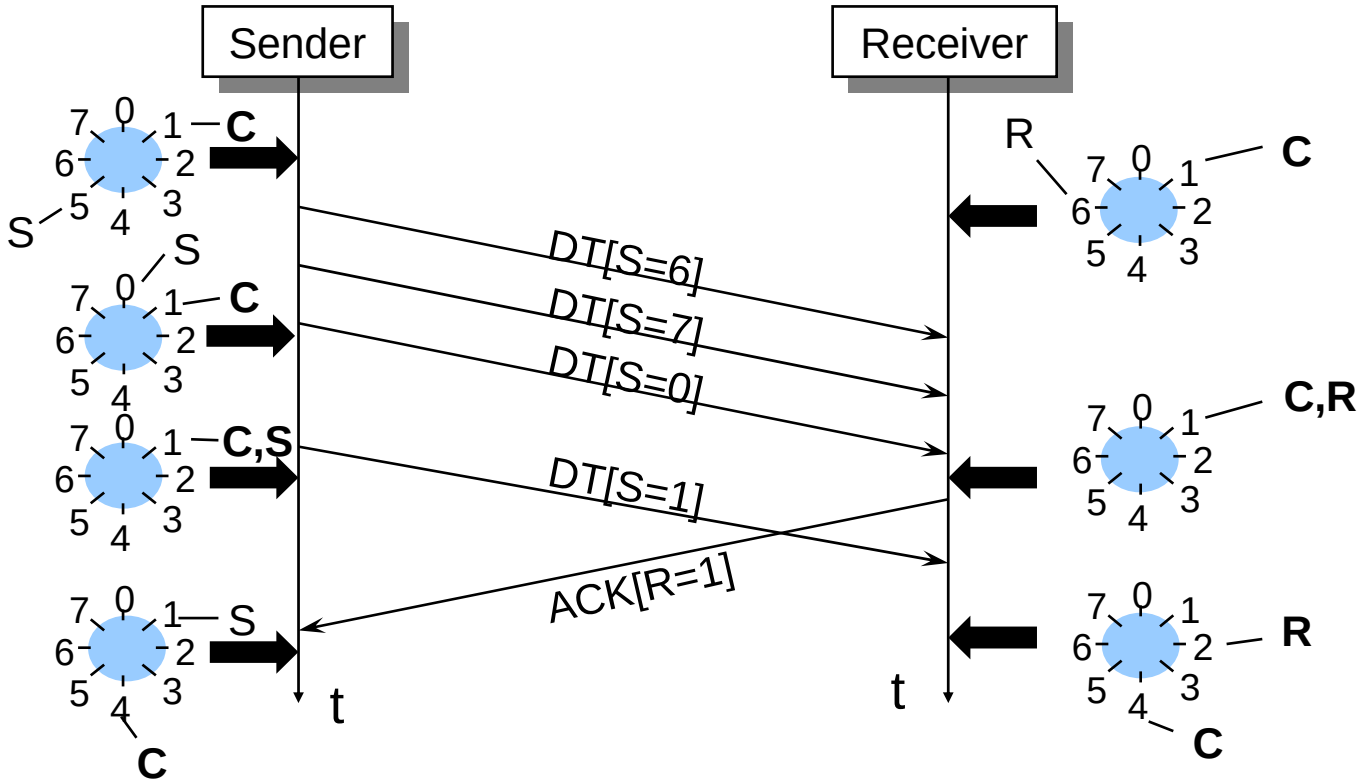
- A simple sliding window example for n=3, window size fixed to 1
- Sender here represents the currently unacknowledged sequence numbers
  - If maximum number of unacknowledged frames is known, this is equivalent to sending window as defined on previous slide



a. Initially, before any frame is sent
b. After first frame is sent with seq. num 0
c. After first frame has been received
d. After first acknowledgement has arrived

Sliding window – another example

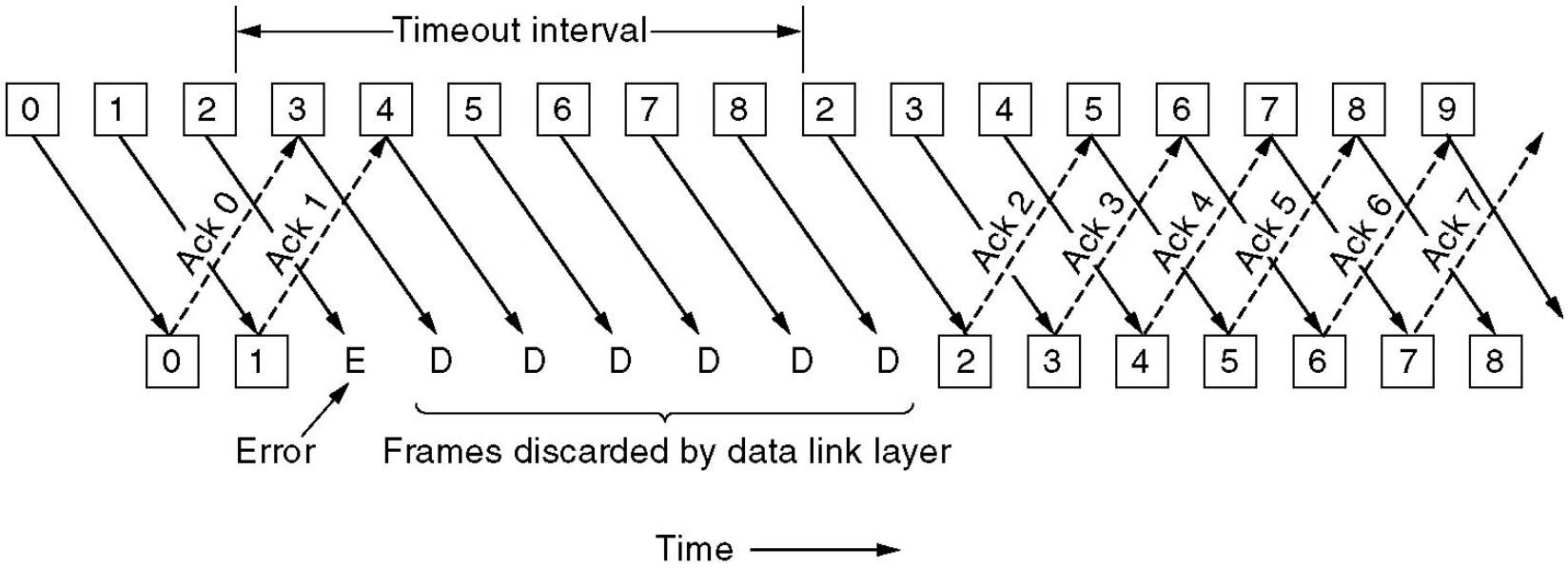
- Sender credit = 4



S: Sequence number (last sent packet)
R: Next expected sequence number = Acknowledges packets up to R-1
C: Upper window limit (current max. sequence number)
Disadvantage: Coupling of flow control and error control

Transmission errors and receiver window size

- Assumption:
  - Link layer should deliver all frames correctly and in sequence
  - Sender is pipelining packets to increase efficiency
- What happens if packets are lost (discarded by CRC)?
- With receiver window size 1, all following packets are discarded as well!

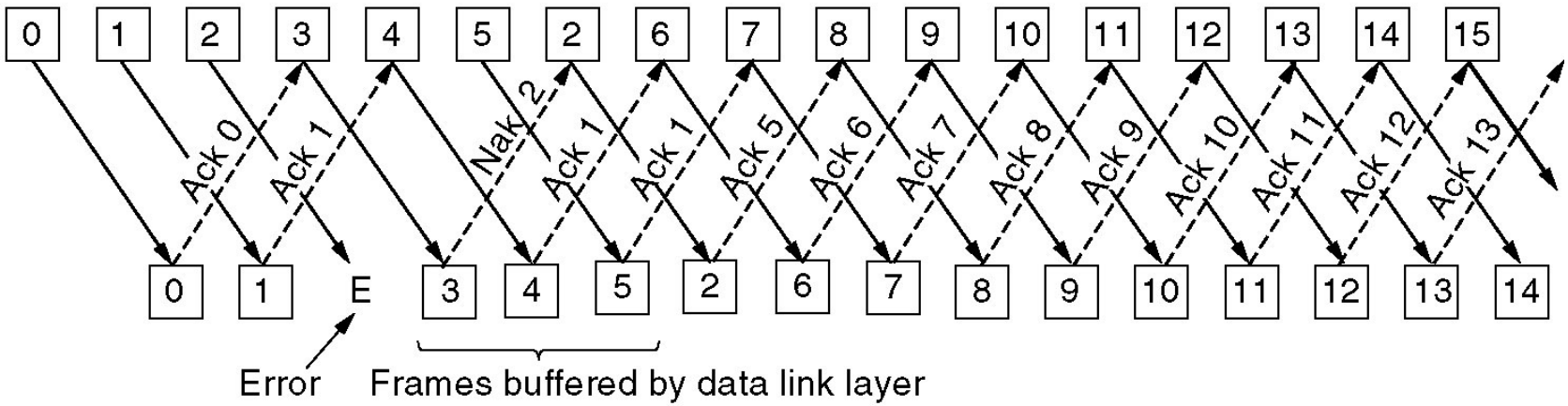


Go-back-N

- With receiver window size 1, all frames following a lost frame cannot be handled by receiver
  - They are out of sequence
  - They cannot be acknowledged, only ACKs for the last correctly received packet can be sent
- Sender will timeout eventually
  - Since all frames sent in the meantime, they have to be repeated
    ➔ Go-back N (frames)!
- Assessment
  - Quite wasteful of transmission resources
  - But saves overhead at the receiver

Selective repeat

- Suppose we invest into a receiver that can buffer packets intermittently if some packets are missing
  - Corresponds to receiver window larger than 1
- Resulting behavior:



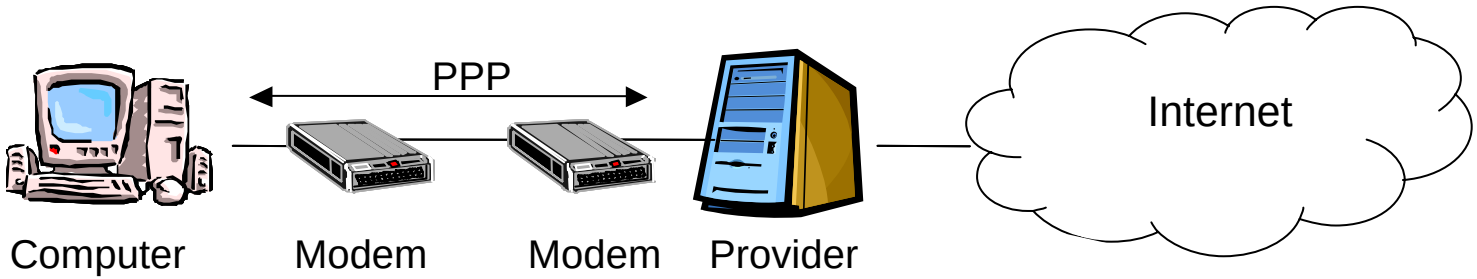
  - Receiver explicitly informs sender about missing packets using ***Negative Acknowledgements***
  - Sender selectively repeats the missing frames
  - Once missing frames arrive, they are all passed to the network layer

Duplex operation and piggybacking

- So far, simplex operation at the (upper) service interface was assumed
  - The receiver only sent back acknowledgements, possibly using duplex operation of the lower layer service
- What happens when the upper service interface should support full-duplex operation?
  - One option: Use two separate channels for each direction – wasteful
  - Better: Interleave acknowledgement and data frames in a given direction
  - Best (and usual): Put the acknowledgement information for direction A$\rightarrow$ B into the data frames for B $\rightarrow$ A
    - As part of B's header – ***piggyback*** it

Example: PPP (Point to Point Protocol)

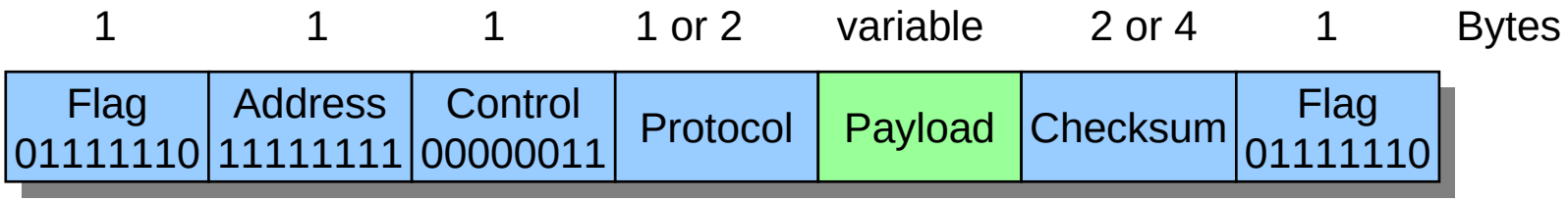
- The majority of the Internet consists of simple point-to-point connections
  - Connections in the backbone between routers, connections from the home via a modem and phone lines…



- PPP (RFC 1661/1662):
  - Layer 2 frames with error detection, frame delimiters
  - Control protocol (LCP, Link Control Protocol) for connection setup, connection test, connection negotiation, connection release
  - Negotiations of layer 3 options independent of layer 3 protocol (separate Network Control Protocol [NCP] for all supported layer 3 protocols)
  - Versions: PPPoE (Ethernet), PPTP (Tunneling), PPoA (ATM)

PPP – Packet format

- Format derived from a classical protocol: HDLC



  - Byte oriented (not bit oriented)
  - Code transparency by character stuffing
  - Typically, no reliability through acknowledgements. However, at higher error rates (e.g., wireless communication) switching to a reliable mode with sequence numbers and acknowledgements
  - Example protocols for payload: IP, AppleTalk, IPX…
  - If not negotiated otherwise, the maximum payload length is limited to 1500 byte
  - Optional header compression possible

PPP – Connection management

- Typical scenario while accessing the Internet from a PC via a modem
  - Call to a service provider via the modem, thus setting up a layer 1 (physical) connection
  - Initiator sends several LCP packet in a PPP frame for negotiation of appropriate PPP parameter, check of link quality
  - Exchange of NCP packets to set up the network layer
    - E.g. dynamical assignment of an IP address using DHCP (if IP if chosen as protocol)
  - The initiator can now use Internet services like any permanently connected computer
  - For connection release the IP address is released via NCP and the layer 3 connection terminated
  - LCP terminates the layer 2 connection
  - Finally, the modem disconnects the physical connection

Conclusion

- Most problems in the link layer are due to errors
  - Errors in synchronization require non-trivial framing functions
  - Errors in transmission require mechanisms to correct them so as to hide from higher layers
  - Or to detect them and repair them afterwards
- Flow control is often tightly integrated with error control in practical protocols
  - But it IS a separate function and can be realized separately as well
- Connection setup/teardown still has to be treated
  - Necessary to initialize a joint context for sender and receiver