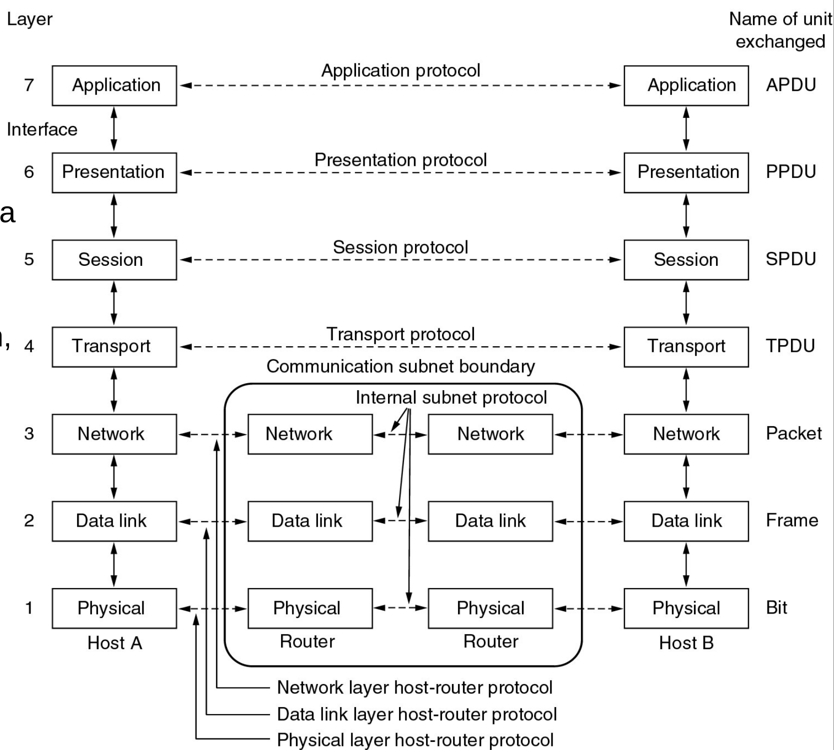
Data Link Layer

ISO/OSI Reference Model

7. Application
   - E-Mail, Terminal, Remote login
6. Presentation
   - System dependent presentation of data (EBCDIC/ASCII)
5. Session
   - Connection establishment, termination, check points for restart
4. Transport
   - Segmentation, congestion avoidance
3. Network
   - Routing
2. Data Link
   - Check sum, flow control
1. Physical
   - Electrical, mechanical, optical devices

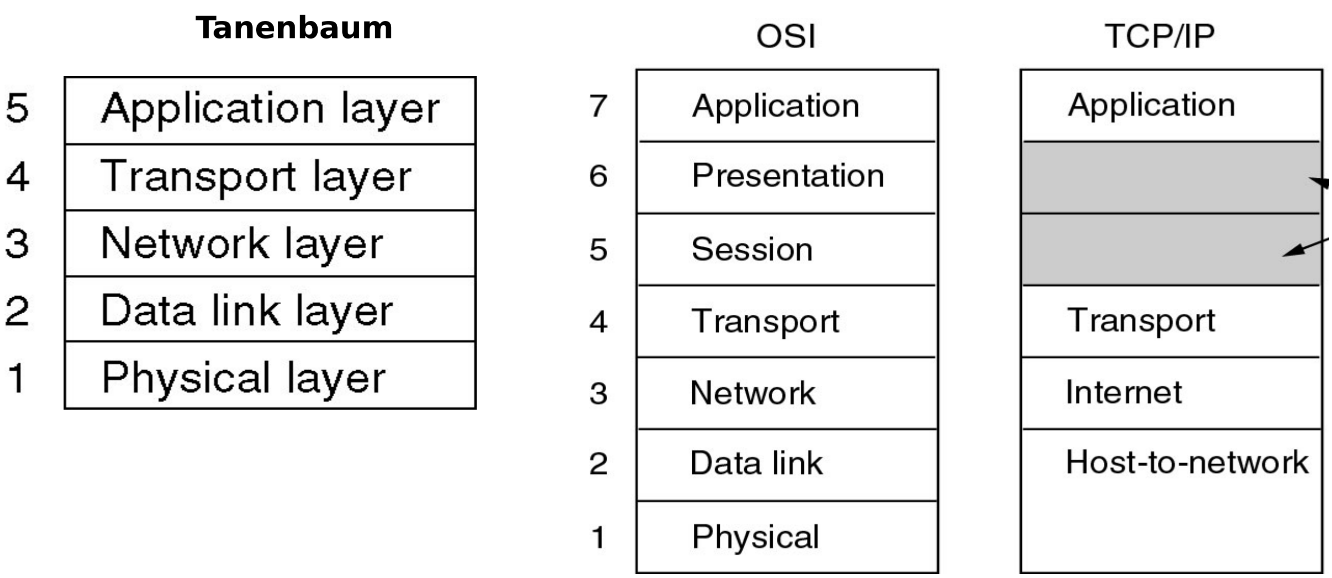


Layers of the Internet - TCP/IP Layers

Application	**Telnet, FTP, HTTP, SMTP (E-Mail), DNS, ...**
Transport	**TCP (Transmission Control Protocol)** **UDP (User Datagram Protocol)**
Network	**IP** (Internet Protocol) **+ ICMP** (Internet Control Message Protocol) **+ IGMP** (Internet Group Management Protoccol)
Host-to-network	**LAN** (z.B. Ethernet, Token Ring etc.)

Hybride Model

We use Tanenbaum’s hybride model



(Tanenbaum)

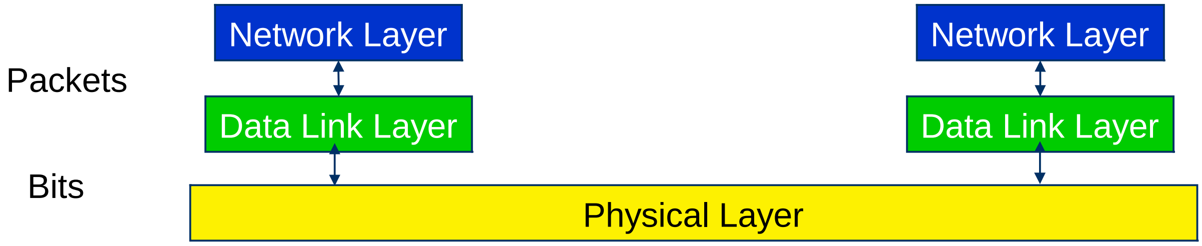
Data Link Layer

- Tasks of the data link layer:
  - Providing services for the network layer
  - Frames
  - Error control
  - Flow control
- Error detection and correction
  - Error correcting codes

Medium Access Controll

Services of the Data Link Layer

- The situation of he data link layer
  - the physical layer transmits bits
  - without structure, possibly with errors
- The network layer requires following from the data link layer:
  - transmission without errors
  - transmission of structured data
    - data packets or data stream
  - reliable data flow



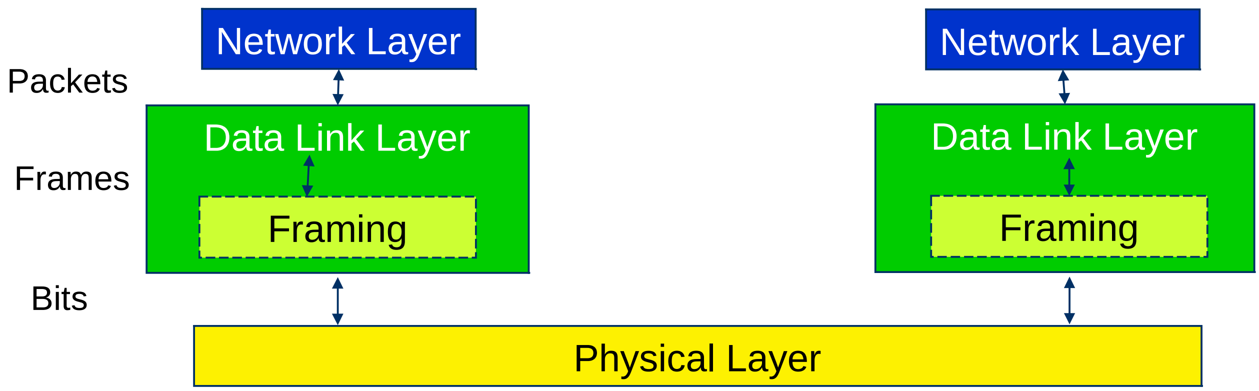
Distinction: Service and Implementation

- Example
  - The network layer requires a connectionless reliable service
  - The data link layer **internally** implements a connection oriented service with error control

- Other combinations are also possible

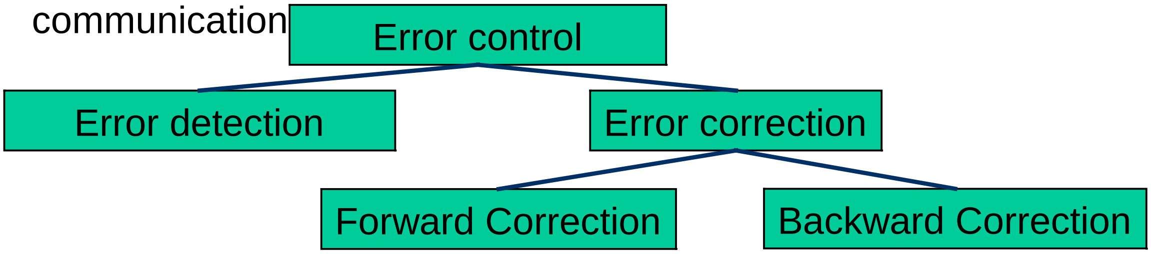
Frames

- Divide the bit stream of the physical layer into peaces: into **frames**
  - Necessary for error control
  - The frames are the packets of the data link layer
- Fragmentation (and defragmentation at the receiver) is necessary, if the packets of the network layer are larger that the frames



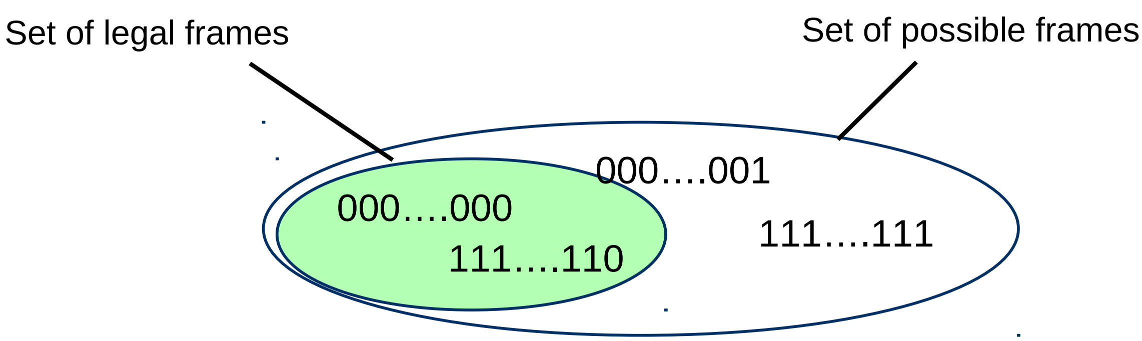
Error Control

- The minimum service required from the data link layer
  - by using frames
- Error detection: detects if there are incorrectly transmitted bits
- Error correction: Cleaning bit errors
  - Forward Error Correction
    - Using redundant code, which allows to correct the error without retransmission
  - Backward Error Correction
    - After detecting an error, the error will be corrected using additional communication



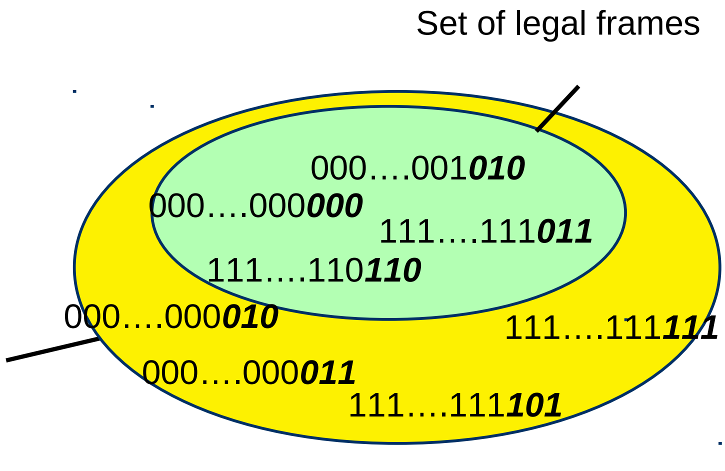
Redundancy

- Redundancy is a precondition of error control
- Without redundancy
  - a frame of length m can represent 2^m possible data,
  - each of them is legal
- A bit error results in new data content



Redundancy

- Main idea:
  - Some possible m-bit sequences are illegal
  - For the representation of 2^m different data,
    - > 2^m possible frames are necessary
    - > m bits are necessary within a frame
  - Length of the frames is $n > m$
  - $r=m-n$ redundant bits
    - Pl. Header and/or Trailer
- Error control is just possible, if we distinguish between legal and illegal frames



Illegal Frames

- The sender only sends legal frames
- In the physical layer bit errors can occur
- Hope:
  - Error in a legal frame always leads to an illegal frame
  - Never to another legal frame
- Necessary condition:
  - The number of bits altered in the physical layer is bounded by a certain number
  - e.g. $\leq$ k bits per frame
- The legal frames are sufficiently different, in order to recognize this frame error rate

Simplest Redundancy: Parity Bit

- A simple rule for adding a redundant bit (i.e., n=m+1): is the parity
- **Odd parity**
  - Insert 0, if the number of 1s is odd, otherwise, insert 1
- **Even parity**
  - Insert 0, if the number of 1s is even, otherwise, insert 1

- Example:
  - Original message without redundancy: 01101011001
  - Odd parity: 01101011001***1***
  - Even parity: 01101011001***0***

Error Detection: CRC

Efficient error detection: Cyclic Redundancy Check (CRC)

Often used in practice

High error detection rate

Hardware implementation is easy

Based on polinomial arithmetic over the field Z_2

Bit string is interpreted as representing a polynomial

Coefficients 0 and 1 are possible, interpreted modulo 2

Computing in Z_2

Arithmetic modulo 2:

Rules:

addition mod 2

A	B	A + B
0	0	0
0	1	1
1	0	1
1	1	0

subtraction mod 2

A	B	A -B
0	0	0
0	1	1
1	0	1
1	1	0

multiplication mod 2

A	B	A · B
0	0	0
0	1	0
1	0	0
1	1	1

Example: 0110111011
+ 1101010110
= 1011101101

Polinomial Arithmetic Modulo 2

- Consider polynomials over the field Z_2
  - $p(x) = a_n \cdot x^n + \dots + a_1 x^1 + a_0$
  - Coefficients a_i and variables x are $a_i, x \in \{0,1\}$
  - Computation modulo 2
- Addition, subtraction, multiplication and division (with remainder) of polynomials, as known

Bit Strings and Polynomials of Z_2

- Idea:
  - Consider the bit strings of n bits as coefficients of a polynomial

- Bit string: $b_n b_{n-1} \ldots b_1 b_0$
  Polynomial: $b_n \cdot x^n + \ldots + b_1 \cdot x^1 + b_0$
- A bit string of (n+1) bits corresponds to a polynomial of degree n

- Isomorphism
  - A xor B = A(x) + B(x)
  - A left shift of A with k position corresponds to:
  - $C(x) = A(x)\, x^k$
  - By this isomorphism, we can define division with bit strings

Division by Bit Strings

Example:

1101010101 : 1001 = 1100110 remainder 11
1001
1000
1001
001101
1001
1000
1001
0011

Redundancy by Polynomials: CRC

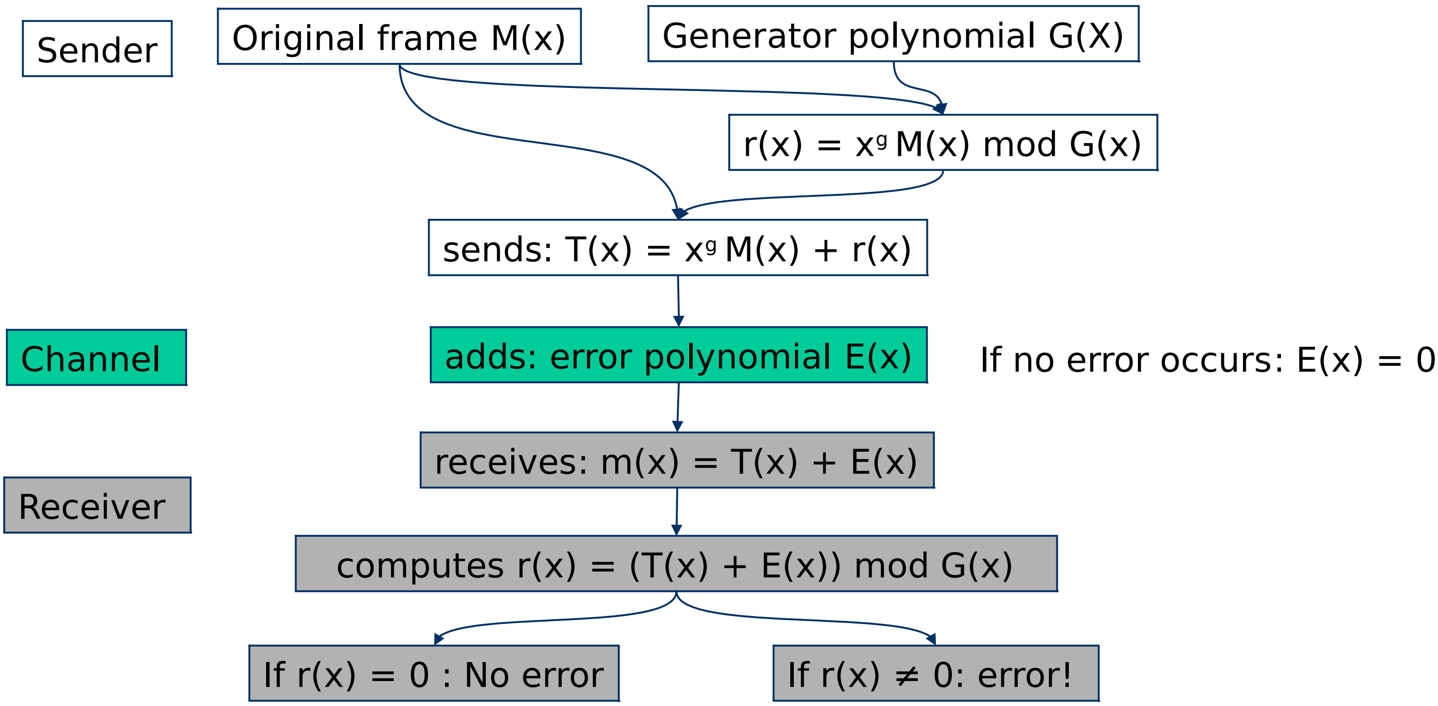
- Define a generator polynomial G(x) of degree g
  - G(x) is known by sender and receiver
  - we generate g redundant bits
- Given:
  - A frame (message) M, as a polynomial M(x)
- Sender
  - Computes the value $r(x) = x^g M(x) \bmod G(x)$
  - Sends: $T(x) = x^g M(x) + r(x)$
  - Note: $x^g M(x) + r(x)$ is a multiple of G(x)
- Receiver
  - Receives m(x)
  - Computes the remainder: m(x) mod G(x)

CRC

- If no error occured:
  - T(x) is received correctly
- Bit error: T(x) contains altered bits
  - This is equivalent to the addition of an error polynomial E(x)
  - The receiver receives m(x) = T(x) + E(x)
- Receiver
  - Computes the remainder m(x) mod G(x)
  - If no error: m(x) = T(x),
    - The remainder is 0
  - Bit errors: m(x) mod G(x) = (T(x)+ E(x)) mod G(x)

$$= \underbrace{T(x) \bmod G(x)}_{0} + \underbrace{E(x) \bmod G(x)}_{\text{error indicator}}$$

CRC – Overwiev



The Generator Determines the Properties of the CRC

- Bit errors are not recognized if and only if E(x) is a multiple of G(x)
- Choosing G(x):
  - 1-bit-error: $E(x) = x^i$ error at position i
    - If G(x) contains at least 2 non-0 coefficients, then E(x) is not a multiple
  - 2-bit-errors: $E(x) = x^i + x^j = x^j (x^{i-j} + 1)$, where i>j
    - G(x) must not divide $(x^h + 1)$, for all h, $0 \leq h \leq k$, where k is the maximum frame size
  - Odd number of errors:
    - Then E(x) is not a multiple of (x+1)
    - Idea: Let (x+1) be a factor of G(x)
      – then E(x) is not a multiple of G(x)
- By clever choice of G(x), all burst errors of length $\leq r$ can be detected

CRC in Practice

- The generator polynomial of the IEEE 802.3 (Ethernet) standard (CRC-32):
  - $x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^{8} + x^{7} + x^{5} + x^{4} + x^{2} + x + 1$

- In practice, residual errors after CRC check are ignored
  - But they may still happen! If E(x) is a multiple of G(x).

- Implementation in hardware:
  - Simple XOR operation
  - HW implementation: shift register circuit
  - Negligible overhead in hardware

Literature

- Andrew S. Tanenbaum and David J. Wetherall: Computer Networks. Pearson, 5 edition, 2010.