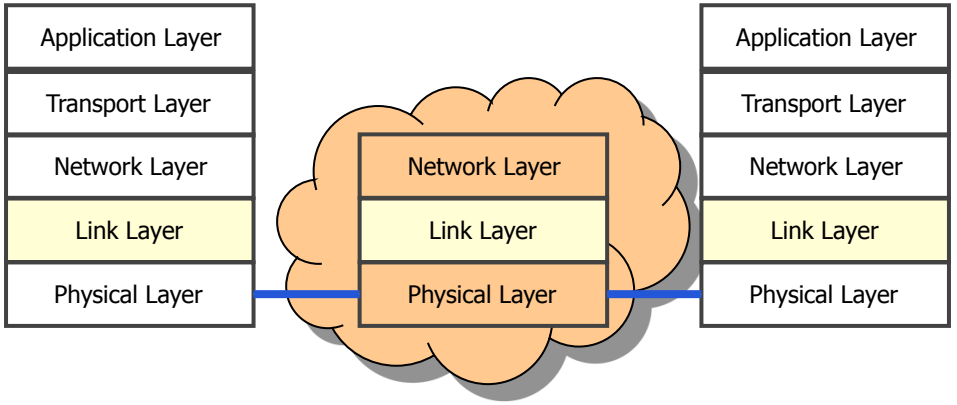
Freie Universität Berlin

Telematics
Chapter 6

The Link Layer

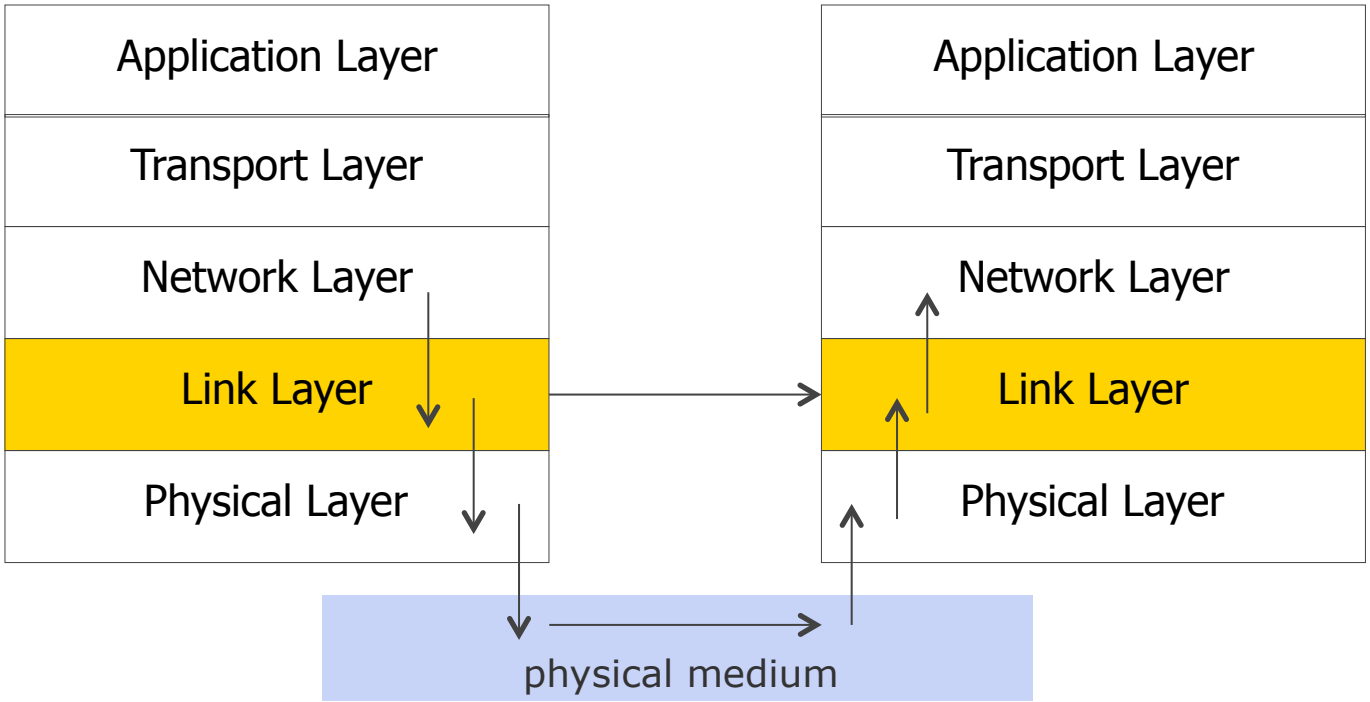


Dr. habil. Emmanuel Baccelli
INRIA / Freie Universität Berlin

Institute of Computer Science
Computer Systems and Telematics (CST)

Context

- The Link Layer provides services to the Network Layer
  - Framing
  - Error control
  - Flow control
  - Addressing and medium access control (MAC)
- The Link Layer uses the bit-by-bit service from the Physical Layer



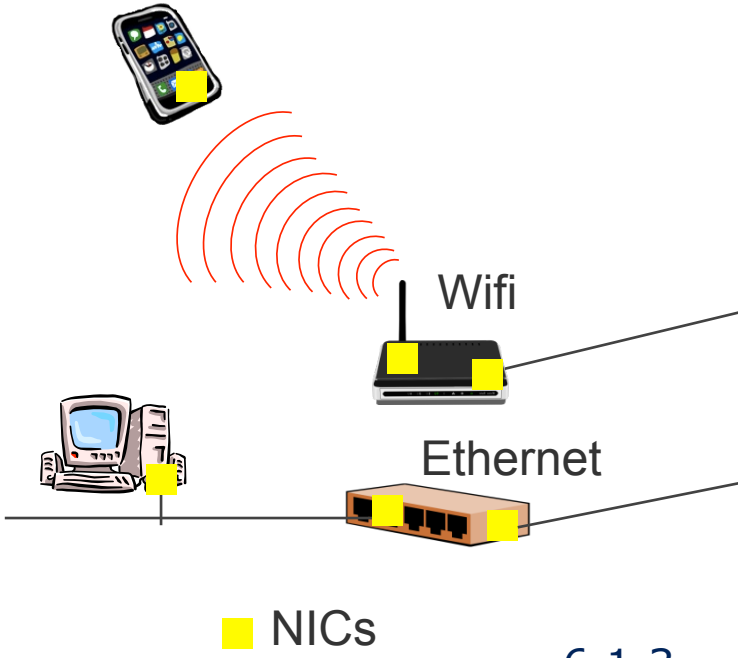
The Hardware Aspect: NIC

- Devices connected to the network have a piece of hardware dedicated to interfacing with the communication medium
  - **Network Interface Card (NIC)**
  - Provides link layer abstraction to device's network stack: send/reveive frames

- **The Link Layer manages communication between NICs**

- One NIC per physical interface
  - e.g. routers and switches have several NICs
  - modern hosts too (laptops, smartphones etc.)

- Link Layer protocols and Physical Layer protocols usually run directly in the NIC drivers.

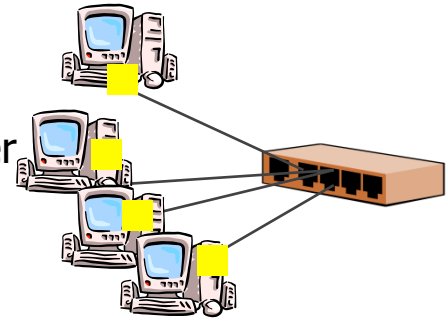


Types of NIC

- Essentially two types of NIC

  - Point-to-point NIC
    - e.g. NIC on fibers of a WAN
    - Typically high bandwidth bidirectional link
    - Dedicated channels per direction => **no collisions**
    - Rather **straightforward** to manage on top of services from the Physical Layer

  - Broadcast NIC
    - e.g. wireless broadcast: NIC connecting to Wifi in a LAN
    - e.g. wired broadcast: NIC connecting to a wired hub in a LAN
    - Medium shared in a way that creates **collisions** if two devices transmit at the same time
    - **More complex** to manage on top of services from the Physical Layer
    - Need for Medium Access Control mechanisms, and addressing schemes



CONTENT of this CHAPTER

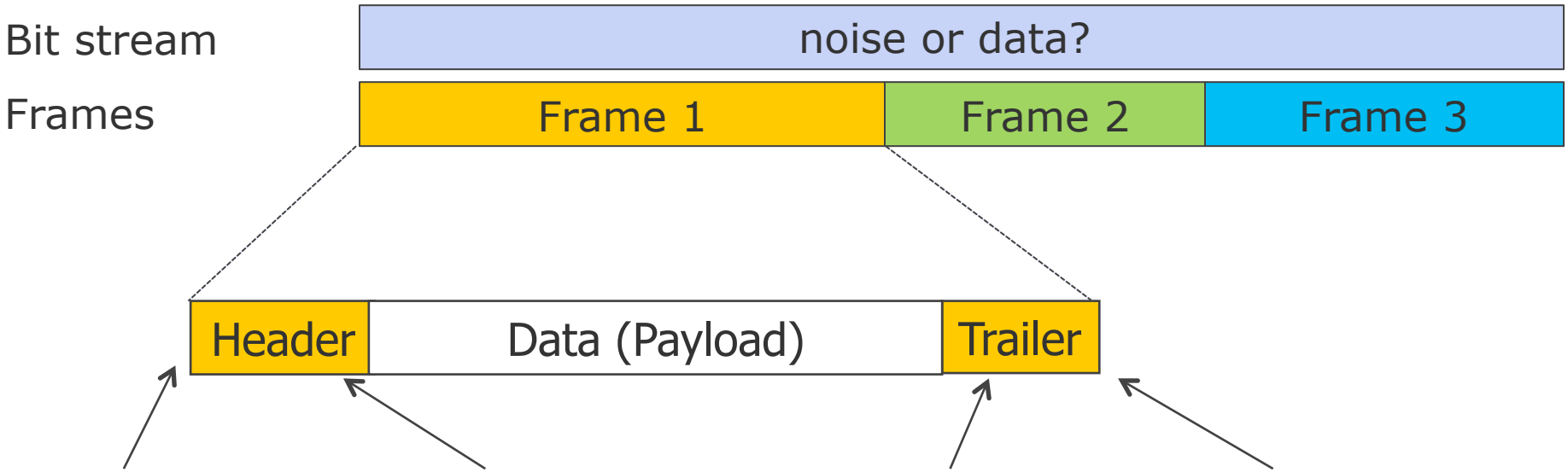
- Framing
- Error Detection & Correction
- Flow control
- Multiple Access Control

- Protocols
  - PPP
  - Ethernet
  - Wifi
  - ATM
  - SDH

- Infrastructure
  - Physical elements
  - Virtual LANs

Framing: Principle & Generic Structure

- Help receiving NIC distinguish noise from data!
- Organization of messages into well defined structures: **frames**



Preamble:

flag byte sequence marking beginning of a frame (and sometimes the frame's length)

Header:

Control information (addresses, frame numbers,…)

Error check:

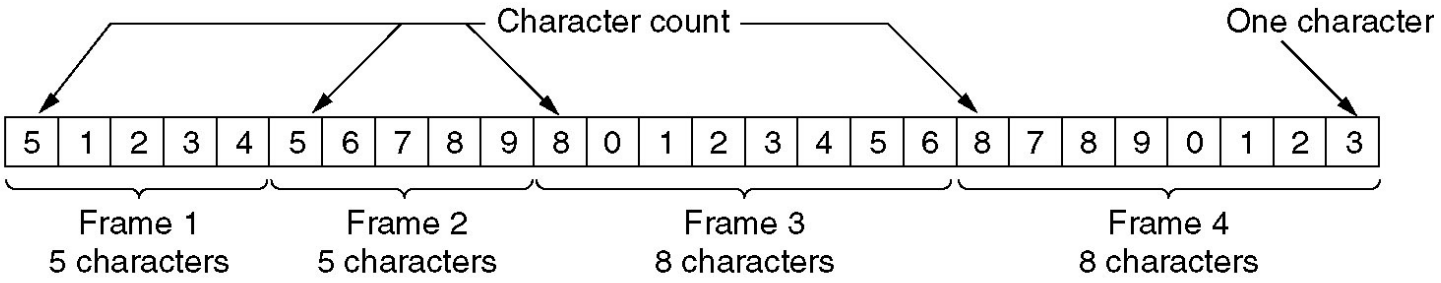
Frame Checking Sequence (FCS)

Postamble:

flag byte sequence marking end of a frame

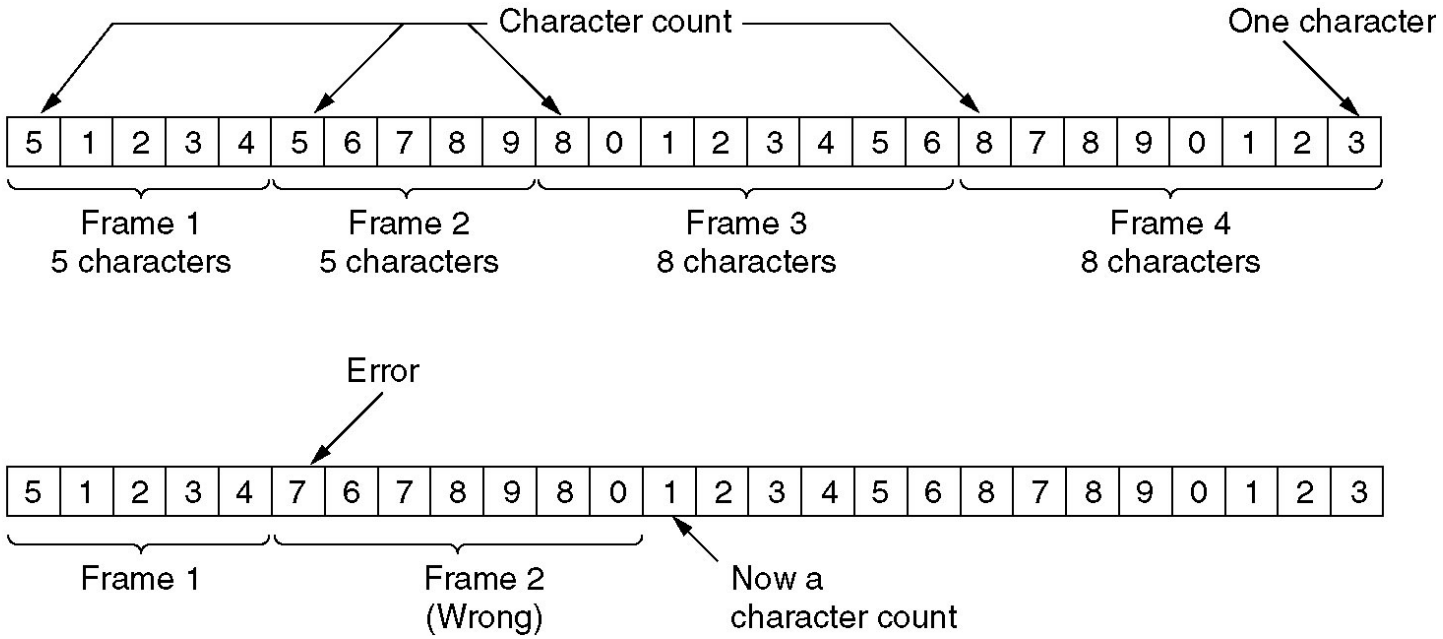
Framing: Character Count

- Specifying the number of characters in the frame
  - Spare the postamble



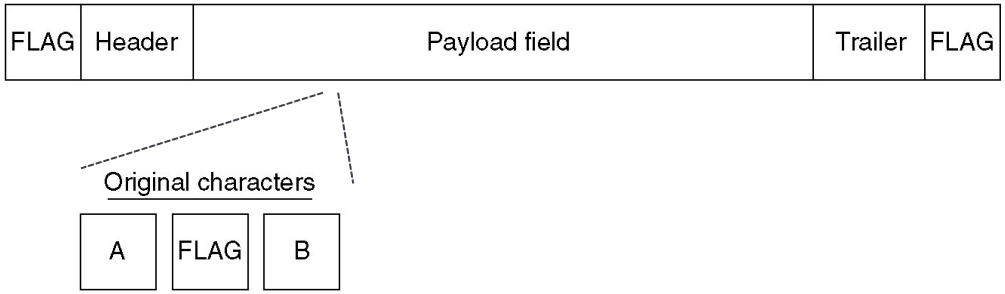
Framing: Character Count

- Specifying the number of characters in the frame
  - Spare the postamble
  - A bit error in the character count is catastrophic! Loss of synchronization.



Framing: Character Stuffing

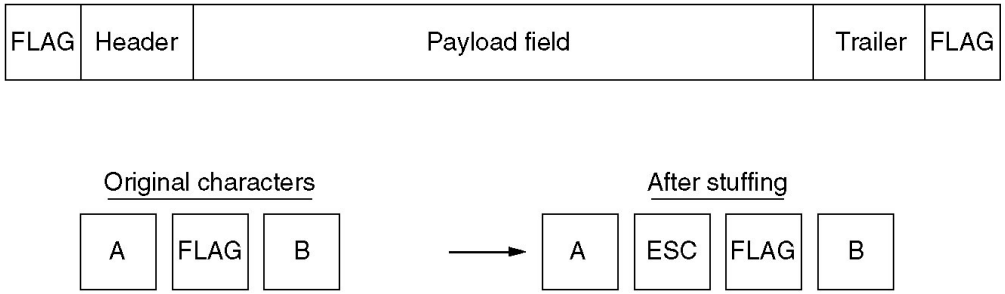
- Start and end of a frame is represented by a special **flag byte** sequence.
- Problem: what happens if the flag byte occurs in the payload?



Framing: Character Stuffing

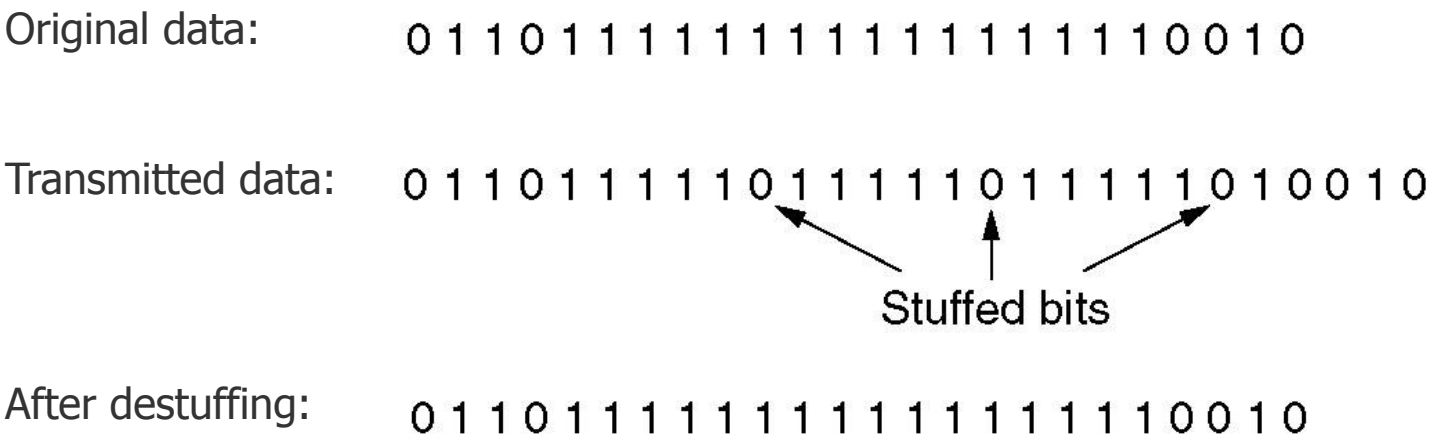
- Start and end of a frame is represented by a special **flag byte** sequence.

- Problem: what happens if the flag byte occurs in the payload?
- Solution: byte stuffing/**character stuffing**
  - Special escape byte (ESC) inserted by the sender and removed by the receiver
  - If the escape byte occurs in the data, then it is also stuffed



Framing: Bit Stuffing

- Character stuffing is bound to the character set (disadvantage)!
- General form: Bit stuffing
  - Frames begin and end with a special pattern: 01111110
  - Sender inserts after five 1s a 0-bit, i.e., 011111x... ➡ 0111110x... and the receiver removes it



CONTENT of this CHAPTER

- Framing
- Error Detection & Correction
- Flow control
- Multiple Access Control

- Protocols
  - PPP
  - Ethernet
  - Wifi
  - ATM
  - SDH

- Infrastructure
  - Physical elements
  - Virtual LANs

Bit Errors: How to deal with them?

- Observation: transmissions over the physical layer are not error-free
  - Errors tend to come in bursts rather than single
- Problem: how can bit errors be recognized and repaired?
  - Enable receiver to detect bit errors in a received frame
  - Enable receiver to correct bit errors in a received frame
- Solution: add error control data to each frame
  - A frame consisting of m data bits and r **check** bits. Total length: $n = m + r$
  - The n-bit unit with data and check bits is called **codeword**
- Types of solutions
  - Error-detecting codes
  - Error-correcting codes

Frame Check Sequence: Overhead Tradeoff

- (very) naive solution: repeat the payload
  - Detect errors via bit-wise mismatch between payload and repeated payload
- Advantages
  - Probability of same bit-error repeating is very low.
- Drawbacks
  - Overhead: half of the throughput is dedicated to control.

- Tradeoff
  - Amount of overhead compared to detection/correction performance gains

Frame Check Sequence: Parity Bit Check

Parity Bit:

Count the number of 1s

Sender: 10111001 PB: 1 sent: 101110011

Receiver: 001011011 PB computed: 0 => error!

Parity Bit Advantages

- Just 1 bit of overhead!
- Odd-Bit errors are detected

Parity Bit Drawbacks

- Even-Bit errors not detected
- Corrections are not possible!

Double Parity:

Group bits together to form a matrix. Compute parity bits for each row and column.

Sender:		
	1011	1
	0010	1
	1100	0
	0110	0
	0011	

Receiver:		
	1011	1
	0110	0
	1100	0
	0110	0
	0111	

Double Parity Advantages

- can identify more errors
- can correct some errors

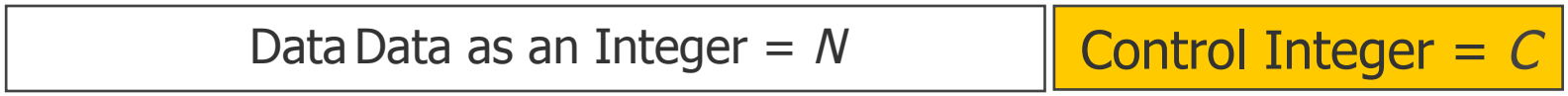
Double Parity Drawbacks

- does not detect all errors
- not practical in networking

Frame Check Sequence: Modulo Check

- **Problem**: parity bits don't detect enough errors don't deal well with error bursts
  - in data communication errors on several bits in a row are frequent

- **Solution**: compute frame check sequence based on a modulo operation
  - Payload can be interpreted as a (large) integer number N
  - Append a control integer C such that: $(N + C) \% 11 = 0$



- Bit errors are detected at the receiver by (re)computing the modulo
  - if different from zero, there was a transmission error in the frame

Frame Check Sequence: Cyclic Code Checksum

- Cyclic code = generalization of the %11 example and parity bit (= %2)
- Example with 32 bit frame check sequence
  - Choose positive integer $i < 2^{32}$
  - Compute checksum = data % i
  - Append checksum to payload of transmitted frame, as frame check sequence
  - Receiver can check payload and checksum consistency
- Tunable parameters
  - number of bits dedicated to the checksum
  - Modulo base i
  - If tuned appropriately, the probability of undetected error is extremely low
    - e.g. less than 10^{-10} with 32 bits, assuming any bit could be corrupted
    - Results based on pure mathematics based on group theory, applied in your NIC!

Frame Check Sequence: CRC

- **Cyclic Redundancy Checksum (CRC)**
  - Polynomial code
    - a m-bit PDU $(a_{m-1}, \ldots, a_0)$ is seen as a polynomial $a_{m-1}x^{m-1} + \ldots + a_0$ with the coefficients a_i "0" and "1".
    - Example: 1100101 is interpreted as $x^6x^5x^4x^3x^2x^1x^0$ ➡ $x^6 + x^5 + x^2 + 1$
  - Sender and receiver agree on a **generator polynomial** $G(x)$

$$G(x) = g_r x^r + g_{r-1}x^{r-1} + \ldots + g_1 x^1 + g_0 x^0 \qquad \text{(with by convention } g_r = g_0 = 1\text{)}$$

Sender interprets a data block of length m as polynomial $M(x)$

$$M(x) = a_{m-1}x^{m-1} + \ldots + a_1 x^1 + a_0 x^0$$

Sender adds 'redundant' bits so that the extended polynomial $M'(x)$ is divisible by $G(x)$

Receiver divides the received extended polynomial $M'(x)$ by $G(x)$.

➡ If the remainder is 0, there was no error, otherwise some error occurred

- Note: the parity bit check can be seen as CRC, with generator polynomial $x + 1$

CRC: Algorithm

- Algorithm for computing the checksum
  - Let r be the degree of generator polynomial $G(x)$.
    - Append r zero bits to the low-order end, so it now contains $m+r$ bits.
    - The corresponding polynomial is $x^r M(x)$.
  - Divide the bit string corresponding to $G(x)$ into the bit string corresponding to $x^r M(x)$.
  - Subtract the remainder from the bit string corresponding to $x^r M(x)$.
  - The result is the checksummed frame to be transmitted, denoted as $T(x)$.

- Note: polynomial arithmetic is used
  - Modulo 2 : addition and subtraction without carriage

➡ Addition and subtraction are identical to exclusive OR (XOR), e.g.

```
  10011011        00110011        11110000        01010101
+ 11001010      + 11001101      - 10100110      - 10101111
  01010001        11111110        01010110        11111010
```

CRC: Example

Data to be transmitted: 10111001

Generator polynomial: $x^4 + x + 1$ ➡ 10011

Note: here, the "extra" positions are preset with zeros – but many systems like, e.g., Ethernet use the inverted bits as presets.

Sender:

```
101110010000 : 10011 = 10100111
10011
   10000
   10011
       11100
       10011
         11110
         10011
           11010
           10011
             1001   =  x^3 + 1 = R(x)
```

CRC = 1001, sending 10111001**1001**

Receiver:

```
101110011001 : 10011 = 10100111
10011
   10000
   10011
       11110
       10011
         11010
         10011
           10011
           10011
                 0
```

Data received correctly

CRC: Undetected Errors

Receiver:

```
00101001 0001 : 10011 = 00101110
  10011
   11110
   10011
    11010
    10011
     10010
     10011
        11
```

Error detected

Receiver:

```
00111011 0001 : 10011 = 00111111
  10011
   11101
   10011
    11100
    10011
     11110
     10011
      11010
      10011
       10011
       10011
          0
```

Error not detected

CRC: Characteristics

- Fundamental characteristic:
  - A polynomial code with r check bits will detect all burst errors of length $\leq r$
- What kind of errors will not be detected?
  - Instead of $T(x)$, erroneous bit string $T(x)+E(x)$ is received
    - Each 1 bit in $E(x)$ corresponds to a bit that has been inverted
    - If there are k 1 bits in $E(x)$, k single-bit errors have occurred
  - Receiver computes: $[\ T(x) + E(x)\ ] / G(x) = E(x)/G(x)$
    - *since* $T(x)/G(x) = 0$
  - Thus errors $E(x)$ that contain $G(x)$ as a factor will be not detected

CRC: 16 bits and 32 bits CRC

Common 16-bit generator polynomials:

- CRC-16: $G(x) = x^{16} + x^{15} + x^2 + 1$
- CRC CCITT: $G(x) = x^{16} + x^{12} + x^5 + 1$
- Ethernet: $G(x) = x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$

Error detection performance for CRC based on 16-bit generator polynomials:

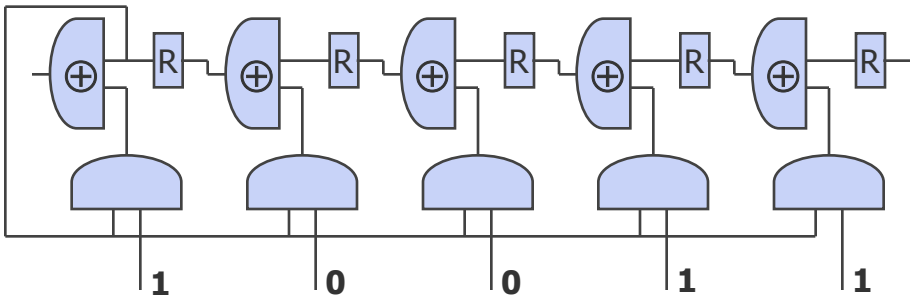
- all single bit errors
- all double bit errors
- all three-bit errors
- all error samples with odd number of bit errors
- all error bursts with 16 or fewer bits
- 99.997% of all 17-bit error bursts
- 99.998% of all error bursts with length ≥ 18 bits
- Remaining error rate < 0.5×10^{-5} block error rate (original)

CRC: Hardware Implementation

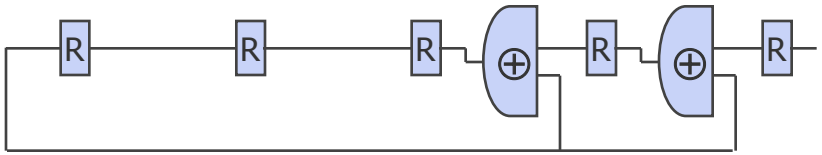
Implementation using shift registers:

- XOR for subtraction
- AND for applying subtraction:
  - first register = 0: no subtraction
  - first register = 1: subtraction

Generator polynomial $x^4 + x + 1$:



Simplified schematic:



When no more input is given in the leftmost register, the other registers contain value of the CRC.

CRC: Software Implementation

- Shift register implementation of CRC-CCITT
  - 16-bit CRC, $r = 16$
  - Generator polynomial $G(x) = x^{16} + x^{12} + x^5 + 1$
  - R represents the content of the r-bit shift register
  - G represents the r least-significant coefficients of the generator polynomial
    - G = 0x1021
  - i represents the input bit for each cycle

CRC: Software Implementation

```c
#include <stdio.h>
static unsigned int R;              // Content of r-bit shift register
static unsigned int Gr1 = 0x0810;   // G = least-significant coeff. of generator polynomial
                                    // Gr1 = G right-shifted one bit position

void ResetCRC()
{
    R = 0;
}

void UpdateCRC(char x)
{
    int i;                                  // Input bit for cycle i
    int k;                                  // Counter
    printf("\nUpdateCRC(%02x)\n", x);
    for (k=0; k<8; k++)
    {
        i = (x >> 7) & 1;                   // msb
        printf(" %04x < %1x -> ", R, i);
        if (R & 0x8000)                     // is msb of R == 1?
            R = ((R ^ Gr1) << 1) + (i ^ 1); // ^ is xor
        else
            R = (R << 1) + i;
        R &= 0x0ffff;
        printf("%04x\n", R);
        x <<= 1;
    }
}
```

CRC: Software Implementation

```
long CheckCRC(FILE *fp, int length)
{
    int i;
    ResetCRC();
    for (i = 0; i < length; i++)
    {
        UpdateCRC(getc(fp));
    }
    return R;
}
```

CRC: Software Implementation

```
main() {
    FILE *fp;
    int c, length = 0;
    long crcSent;
    if ((fp = fopen("tmp", "w")) == NULL) {
        printf("Can't open tmp file. Exiting\n");
        exit(-1);
    }
    ResetCRC();
    while ((c = getchar()) != EOF) {
        UpdateCRC(c);
        putc(c, fp);
        length++;
    }
    printf("CRC is %04x\n", R);
    crcSent = R;
    putc(((R >> 8) & 0xff), fp);
    putc((R & 0xff), fp);
    fclose(fp);
    if ((fp = fopen("tmp", "r")) == NULL) {
        printf("Can't read tmp file. Exiting\n");
        exit(-2);
    }
    if (CheckCRC(fp, length) == 0)
        printf("CRC checks.\n");
    else
        printf("Computed CRC doesn't match\n");
}
```

Error Detection & Correction: Hamming Distance

- **Hamming distance**
  - Number of places, in which two binary sequences differ
  - Example: two codewords w_1=10001001 and w_2=10110001

```
    10001001
XOR 10110001
    --------
    00111000   ➡   d(w1, w2) = 3
```

- Hamming distance of a code (= set of codewords)
  - m bit data ➡ 2^m possible data words, typically all used
  - r bit check bits
  - $m+r = n$ bit codeword ➡ 2^n possible codewords, typically not all used!
  - Principle: construct a **list of all valid codewords**
  - Find the two codewords with minimum Hamming distance
    ➡ this distance is the **Hamming Distance** of this code

Error Detection & Correction: Hamming Distance

The error detecting and correcting properties of a code depends on its Hamming distance

- to detect d errors, a distance of $d+1$ is required
- to correct d errors, a distance of $2d+1$ is required

- Example:
  - Code with only four valid codewords
    w_1=0000000000
    w_2=0000011111
    w_3=1111100000
    w_4=1111111111
  - Distance 5
    - it can detect 4 bit errors
    - it can correct 2 bit error
  - If 0000000111 is received, the original must be 0000011111 ➡ correction OK
  - If 0000000000 is distorted to 0000000111, cannot recover from error

Error Detection & Correction: Hamming Code

- Goal:
  - a code with m data bits and r check bits, which can correct all single bit errors

- Analysis:
  - Each of the 2^m data bits has n invalid codewords with distance 1
    - ➡ Systematically invert each of the n possible bits
    - ➡ Each of the 2^m data bits requires $n+1$ codewords 'closest' to itself
  - But the total number of bit sequences is 2^n
    Therefore we must have $(n+1)2^m \leq 2^n$
    With: $n = m+r$ we have then $m + 1 \leq 2^r - r$

- Result:

  Given m: lower bound on number of check bits needed to correct all single bit errors

  - The **Hamming Code** fulfills this lower limit

Hamming Code: Expansion in Powers of Two

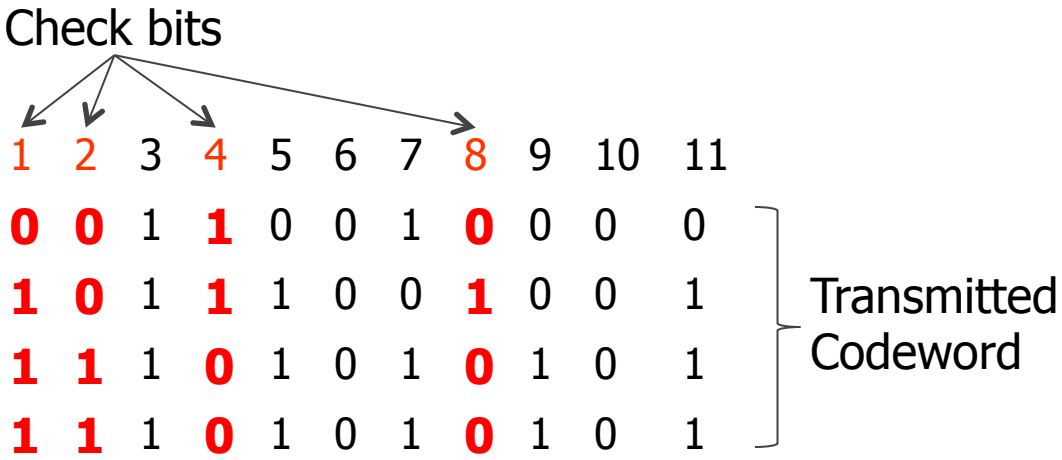
- Idea: Use of several parity bits, each of them considering several bits (overlapping). Errors can be identified/corrected by combining parity bits.
  - The Hamming code is the "minimal" code of this category.
- Principle: Consider each positive integer is the sum of powers of two.
- Algorithm: in a codeword with $n = m + r$ bits:
  - Insert r parity bits in the data, at 2^x positions
  - The data ($m = n - r$ bits) is left with unchanged around the inserted r parity bits
  - An r bit at position 2^x is a parity bit for bits at a position using x in its expansion in powers of two
    - Example: 11 = 1+2+8
    - ➡ Bit 11 is checked by parity bits 1, 2, and 8
    - Example: 17 = 1+16
    - ➡ Bit 11 is checked by parity bits 1 and 16

Bit Position	1	2	4	8	16
1	X				
2		X			
3	X	X			
4			X		
5	X		X		
6		X	X		
7	X	X	X		
8				X	
9	X			X	
10		X		X	
11	X	X		X	
12			X	X	
13	X		X	X	
14		X	X	X	
15	X	X	X	X	
16					X
17	X				X

Hamming Code: Example

Example Data (ASCII-Code)

H	1001000
A	1100001
M	1101101
M	1101101
I	1101001
N	1101110
G	1100111



Parity bit 1: Data bit 3, 5, 7, 9, 11

Parity bit 2: Data bit 3, 6, 7, 10, 11

Parity bit 4: Data bit 5, 6, 7

Parity bit 8: Data bit 9, 10, 11

Receiver:

- Recompute & compare parity bits
- If mismatch, sum up indices of the incorrect parity bits

  ➡ this sum is the index of the incorrect bit

Hamming Code: Strengths & Weaknesses

Hamming Code's Avantage:

- 1-bit errors can be correctly identified and recovered

Hamming Code's Weaknesses:

- Hamming Code is expensive in terms of required check bits
- 2-bit errors are not corrected (or wrongly corrected!)
- 3-bit errors are not recognized

Receiver recomputes parity bits: **11**1**0**0000**0**000

Sum up indices of mismatching parity bits 1, 2 and 4 ➡ bit 7 is detected as false

e.g. Bit 4 and bit 11 inverted:
- ➡ parity bits 1, 2, 4, 8 are wrong
- ➡ bit 15 is to be corrected, but does not exist

e.g. Bit 2 and bit 4 inverted
- ➡ parity bits 2, 4 wrong
- ➡ bit 6 is falsely recognized as incorrect

e.g. Bits 1, 8, 9 inverted
- ➡ all parity bits are correct
- ➡ no error is recognized

Error Correction Mechanisms

Forward Error Correction (FEC)

- Use of error-correcting codes (see also RS- or BCH-codes)
- Errors can be corrected in most cases. Uncorrectable data are simply dropped
- Low latency: feedback from the receiver to the sender is not necessary
- **Suitable for delay sensitive transmissions (e.g. video, audio)**

Automatic Repeat reQuest (ARQ)

- Use of error-detecting codes (CRC)
- Errors are detected, but cannot be corrected.
- Received data containing errors must be requested again from sender
- **Suitable for transmissions which do not tolerate errors (e.g. files)**

- A scheme such as ARQ introduces the need for **flow control**, managing:
  - numbering of data blocks to be sent
  - acknowledgement of blocks by the receiver
  - Retransmission of incorrectly transmitted data blocks

CONTENT of this CHAPTER

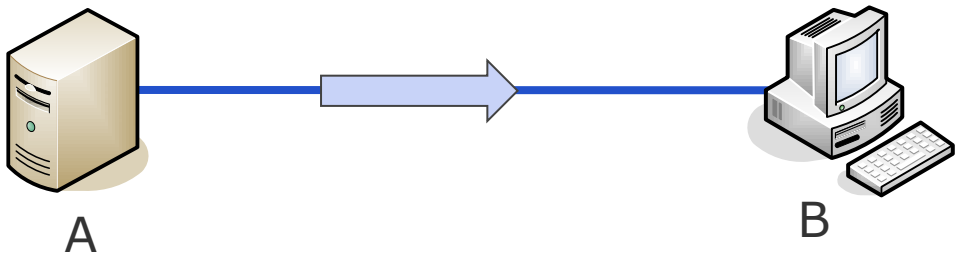
- Framing
- Error Detection & Correction
- Flow control
- Multiple Access Control

- Protocols
  - PPP
  - Ethernet
  - Wifi
  - ATM
  - SDH

- Infrastructure
  - Physical elements
  - Virtual LANs

Scenario of Interaction with the Network Layer

- Scenario:
  - The network layer asks to send a long stream of data packets from A to B
  - How does the link layer deal with this stream of data packets?



Link Layer: Sketch of Interface with Layers 1 and 3

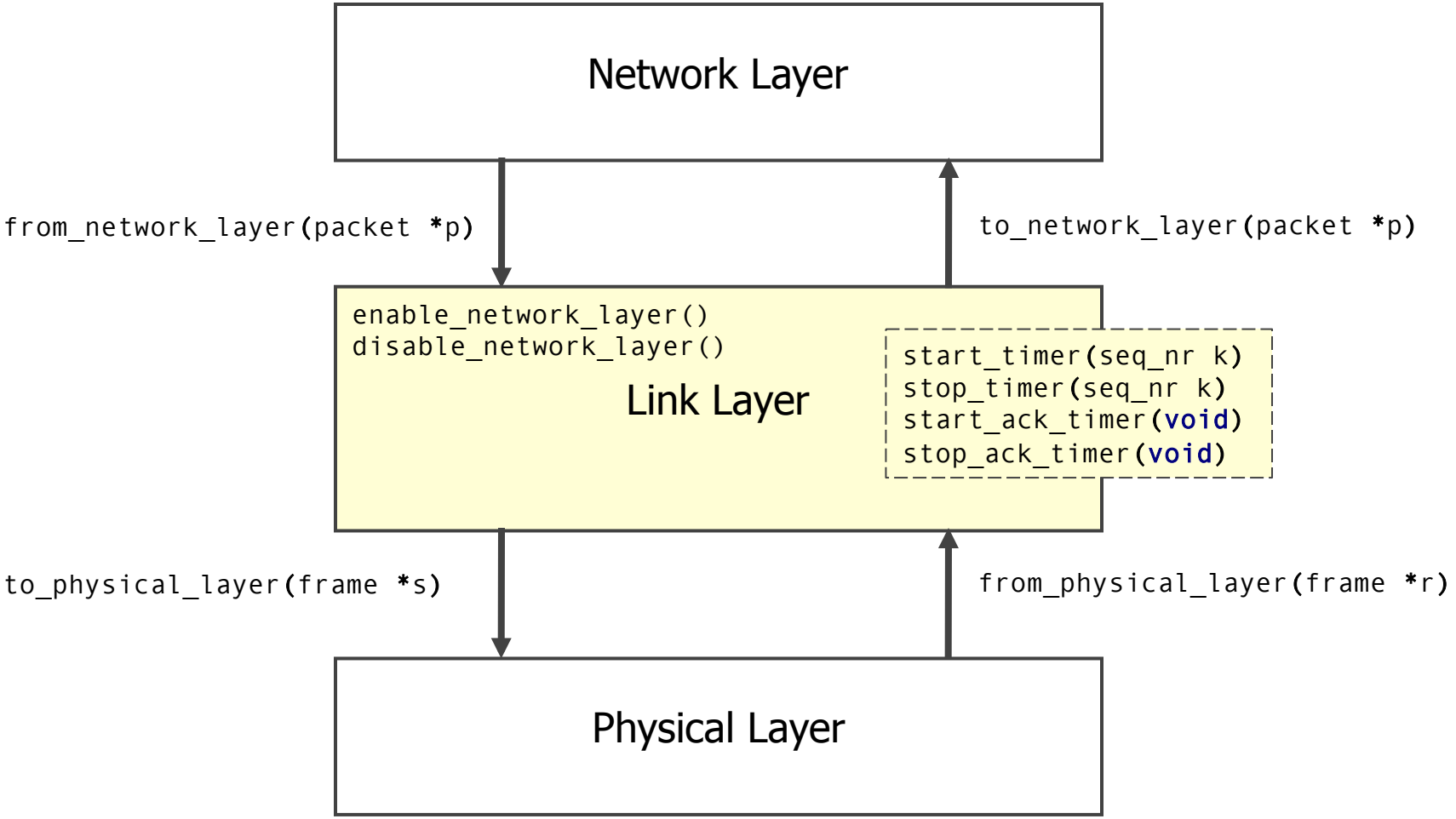
- Definition of some data types (protocol.h):

```
#define MAX_PKT 1024                          /* determines packet size in bytes      */
typedef enum {false, true} boolean;           /* boolean type                         */
typedef unsigned int seq_nr;                  /* sequence or ack numbers              */


typedef struct {
  unsigned char data[MAX_PKT];
} packet;                                     /* packet definition                    */


typedef enum {data, ack, nak} frame_kind;     /* kinds of frames                      */

typedef struct {                              /* frames are transported in this layer */
  frame_type type;                            /* what kind of a frame is it?          */
  seq_nr seq;                                 /* sequence number                      */
  seq_nr ack;                                 /* acknowledgement number               */
  packet info;                                /* the network layer packet             */
} frame;
```

Link Layer: Sketch of Interface with Layers 1 and 3

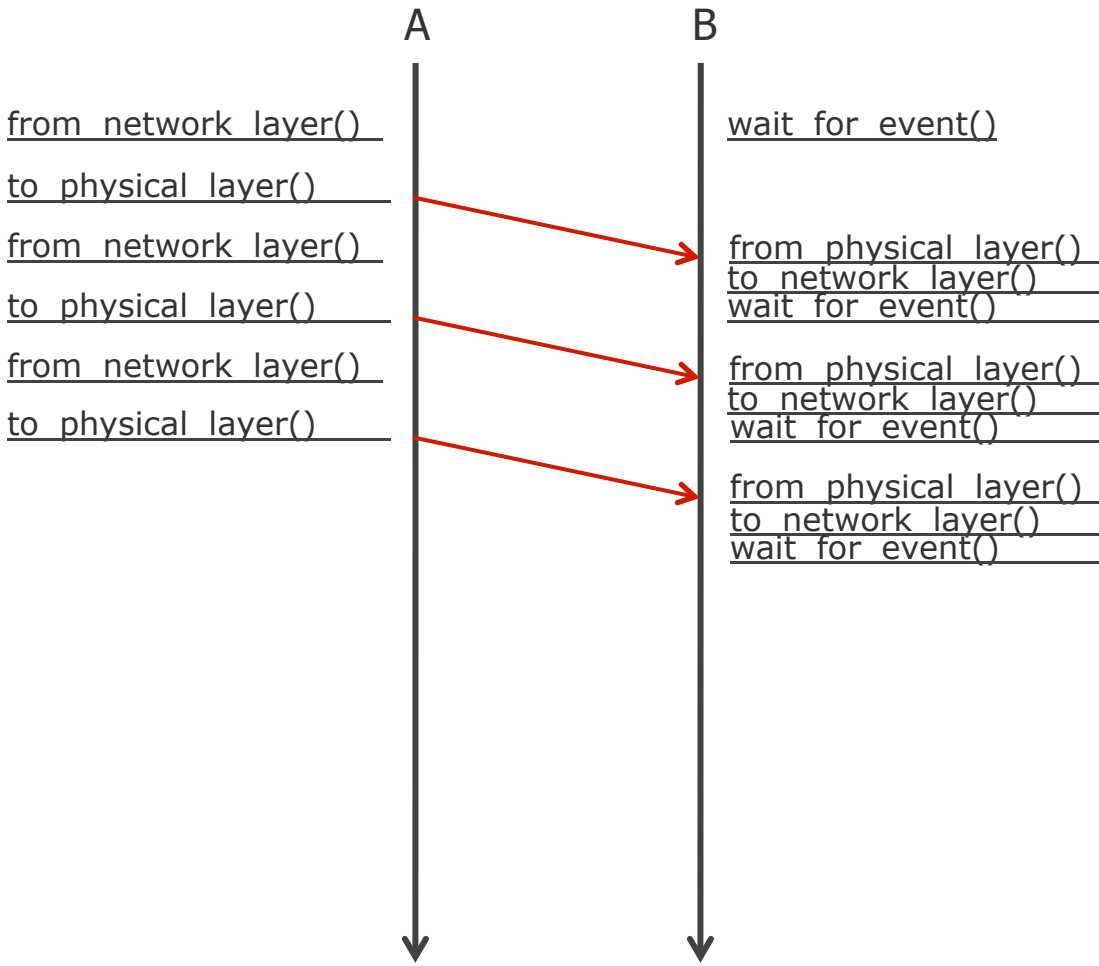
```
void wait_for_event(event_type *event); // Wait for an event; return its type in event


void from_network_layer(packet *p);     // Fetch a packet from the network layer
void to_network_layer(packet *p);       // Deliver packet to the network layer


void from_physical_layer(frame *r);     // Get frame from the physical layer
void to_physical_layer(frame *s);       // Pass the frame to the physical layer


void start_timer(seq_nr k);             // Start the clock running; enable timeout event
void stop_timer(seq_nr k);              // Stop the clock; disable the timeout event


void start_ack_timer(void);             // Start an auxiliary timer; enable ack_timeout
void stop_ack_timer(void);              // Stop auxiliary timer; disable ack_timeout


void enable_network_layer(void);        // Allow the network layer to cause a
                                        // network_layer_ready event.
void disable_network_layer(void);       // Forbid the network layer from causing a
                                        // network_layer_ready event.


// Macro inc is expanded in-line: Increment k circularly.
#define inc(k) if (k < MAX_SEQ) k = k + 1; else k = 0
```

Link Layer: Sketch of Interface with Layers 1 and 3



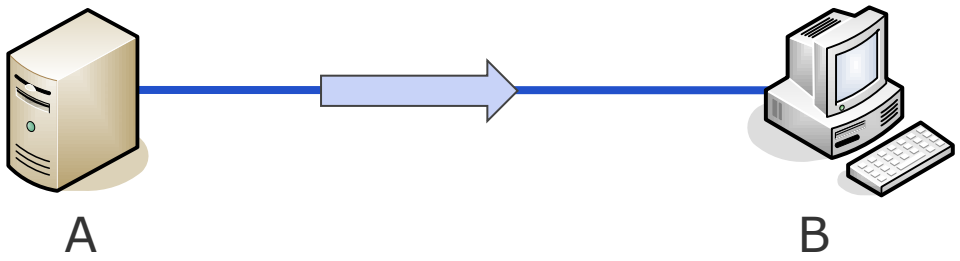
Basic Flow Control Mechanisms: Simplex

- Naïve approach: **Simplex**
  - Transmission in one direction
  - No sequence numbers and no acknowledgements
  - Processing time is ignored
- Implementation of Simplex
  - Two procedures
    - `sender1()` and `receiver1()`
  - Sender in an infinite loop
    - Fetch data, send data
  - Receiver in an infinite loop
    - Get data, pass to network layer
- Advantage: extremely simple!
- Drawback: many assumptions
  - Channel never damages frames
  - Channel never loses frames
  - Network layer is always 'ready'



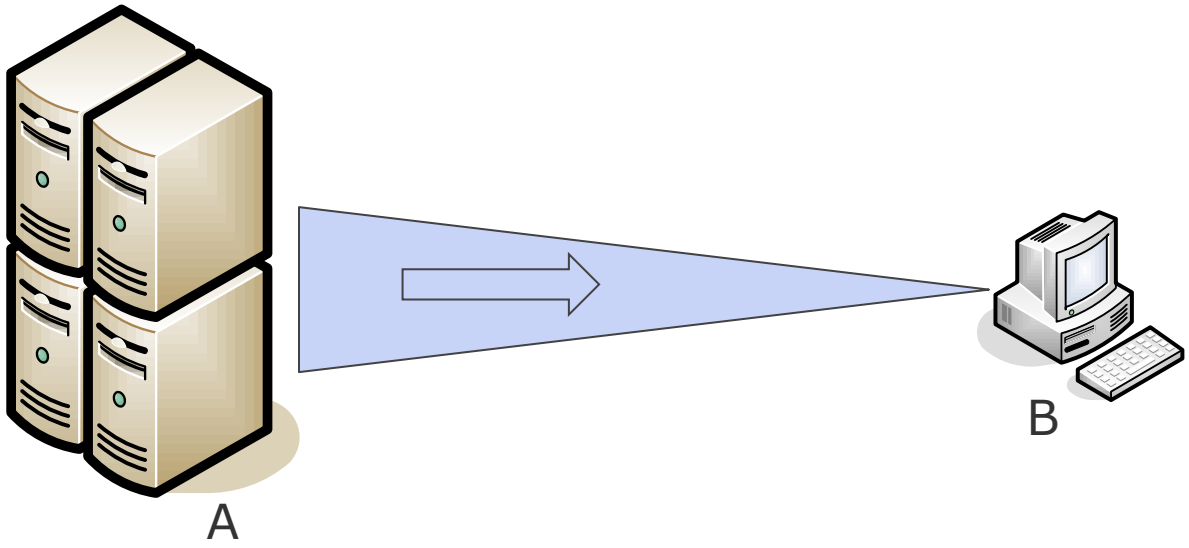
Scenario of Interaction with the Network Layer:

- Scenario:
  - The network layer asks to send a long stream of data packets from A to B
  - How does the link layer deal with this stream of data packets?
    - This time, no assumption that the receiving network layer is 'always ready'



Scenario of Interaction with the Network Layer

- Scenario:
  - The network layer asks to send a long stream of data packets from A to B
  - How does the link layer deal with this stream of data packets?
    - This time, no assumption that the receiving network layer is 'always ready'
    - e.g. fast sender and slow receiver: how to prevent receiver overflow?
      - ➡ Need for **flow control**

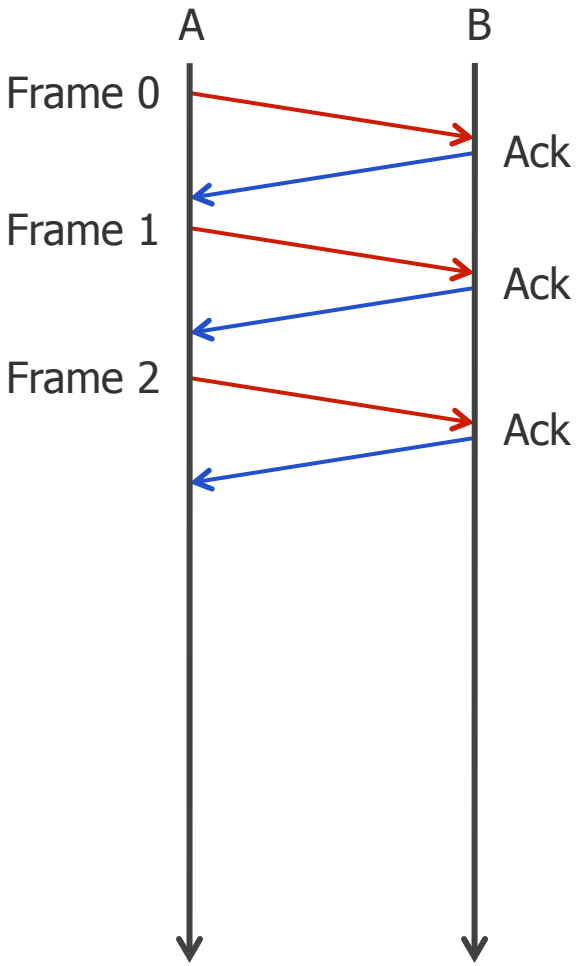


Flow Control

- Two types of approaches for flow control
  - Feedback-based flow control
    - Receiver sends information back to the sender giving permission to send more data
    - Note: needs bidirectional channel (for the feedback to get to the sender)
  - Rate-based flow control
    - Protocol limits the rate of data a sender may transmit without feedback from receiver
  - In this course we focus on **feedback-based flow control**

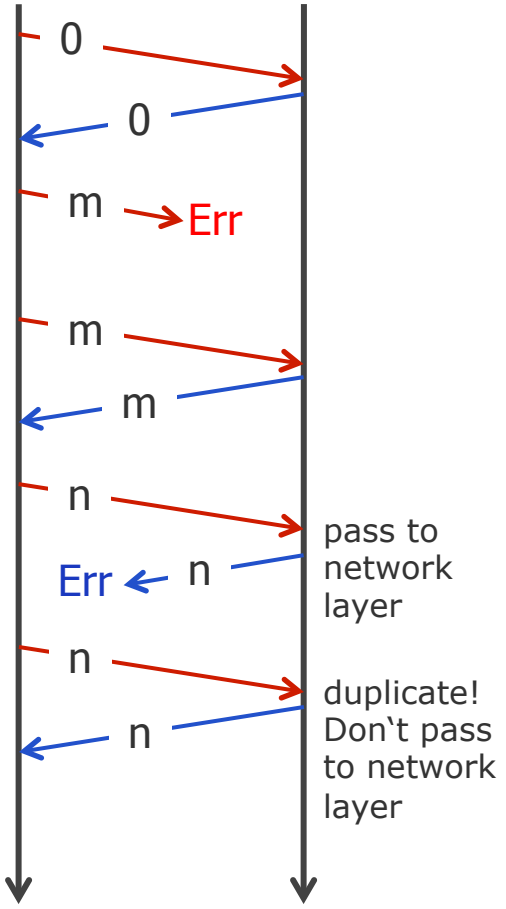
Basic Flow Control Mechanisms: Stop-and-Wait

- Approach: **Stop-and-Wait**
  - bidirectional channel
  - sender sends a data block and **waits**, until an **acknowledgement** from the receiver arrives or a **timeout** is reached.
  - An unacknowledged data block is resent, otherwise the next block is sent.

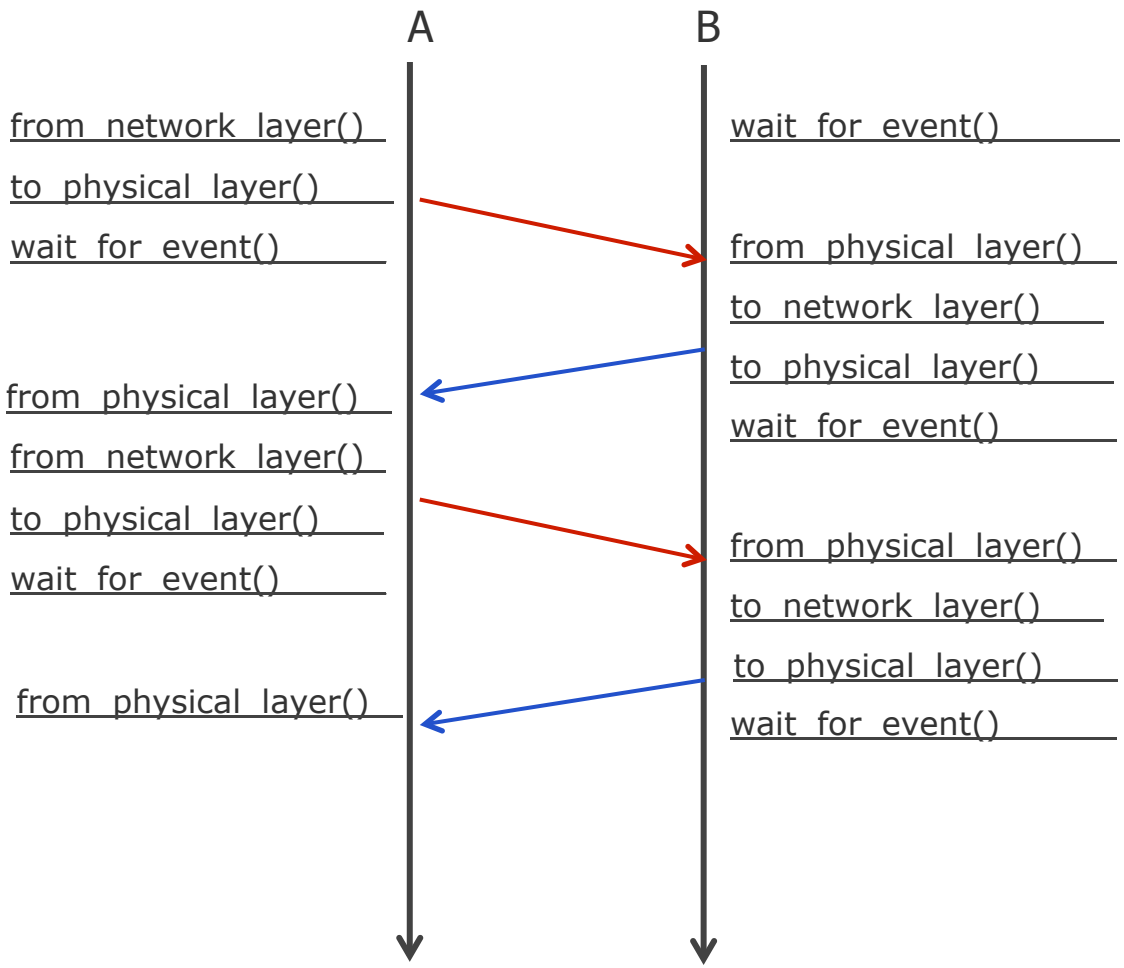


Basic Flow Control Mechanisms: Stop-and-Wait

- Approach: **Stop-and-Wait**
  - bidirectional channel
  - sender sends a data block and **waits**, until an **acknowledgement** from the receiver arrives or a **timeout** is reached.
  - An unacknowledged data block is resent, otherwise the next block is sent.
  - Also works if the channel is **not error-free**
    - Frames and Acks could be damaged or lost
    - Need 1-bit flag in header to detect duplicates
- Advantages: very simple mechanism, avoids receiver overflow
- Drawbacks: **large waiting periods** between the transmission of blocks. Transmission capacity is wasted.
- Basic principle used by:
  - Automatic Repeat reQuest (**ARQ**)
  - Positive Acknowledgement with Retransmission (**PAR**)

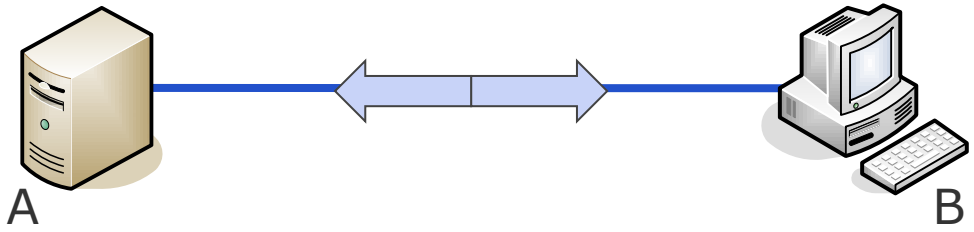


Stop-and-Wait Implementation



Scenario of Interaction with the Network Layer

- Scenario:
  - The network layer asks to send a long stream of data packets from A to B
  - How does the link layer deal with this stream of data packets?
    - no assumption that the receiving network layer is 'always ready'
    - no assumption that the channel is error-free
    - the network layer also sends a long stream of data packets from B to A



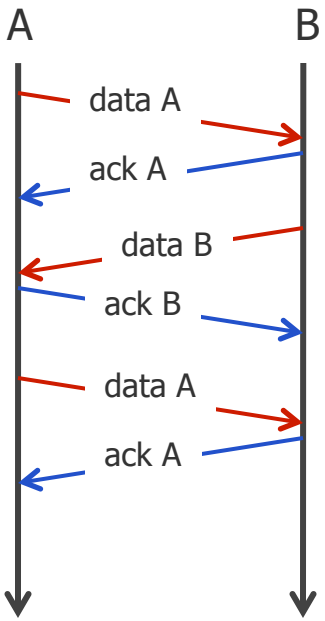
Basic Flow Control Mechanisms: Stop and Wait with Piggybacking

- Approaches for **full-duplex** communication
  - Naïve approach: use two simplex communication channels with stop-and-wait
    - Waste of resources, because the acknowledgements are 'rarer and smaller' than data
  - More efficient approach: stop-and-wait on a single channel for both directions
    - Data frames and acks are intermixed
    - **Type field** in header **distinguishes** data- and ack-frames
  - Even more efficient approach: single channel, stop-and-wait, with **piggybacking**
    - Instead of ack-packets, use a field in the header of a data frame to inform the receiver
    - When a data packet arrives, the receiver does not send immediately an ACK, but instead waits a particular time interval for a data packet to the other direction
    - Question: How long to wait before sending the ACK?
      - Estimate/Guess
      - Fix time
      - RTT

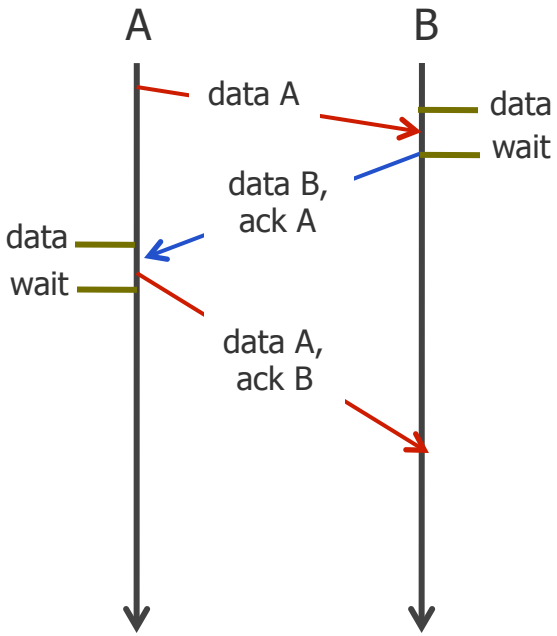
Basic Flow Control Mechanisms: Stop and Wait with Piggybacking

Freie Universität Berlin

Data- and ack-frames as individual messages

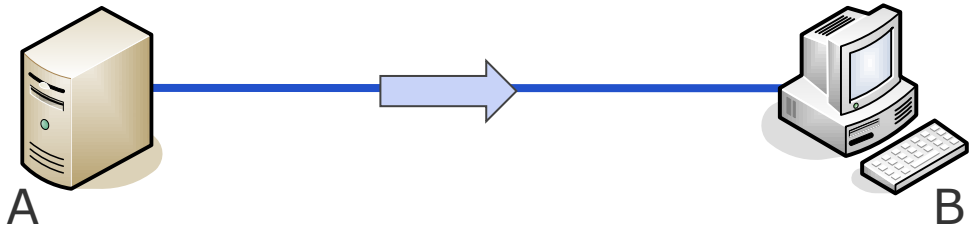


Ack-frames piggybacked



Scenario of Interaction with the Network Layer

- Scenario:
  - The network layer asks to send a long stream of data packets from A to B
  - How does the link layer deal with this stream of data packets?
    - no assumption that the receiving network layer is 'always ready'
    - no assumption that the channel is error-free
    - better throughput compared to stop-and-wait



Basic Flow Control Mechanisms: Sliding Window

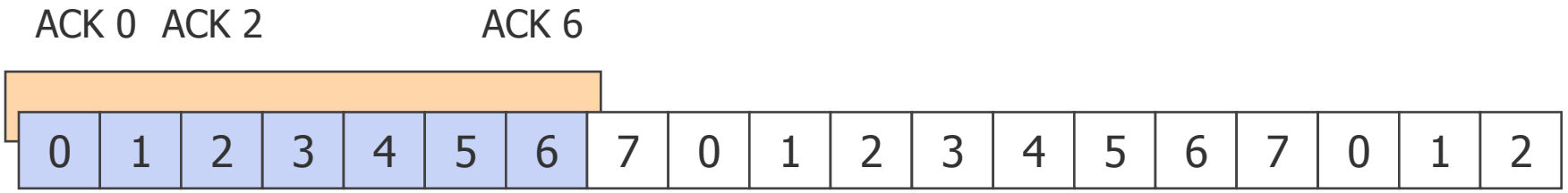
- Goal: avoiding long waiting periods of the sender
  - sender and receiver agree upon a **transmission window**
    - If W is the window size: the sender may send up to W messages without an acknowledgement of the receiver
    - Sender and receiver transmission window do not need to have the same size
  - Need to introduce **sequence numbers** to identify messages
    - The messages are sequentially numbered in the frame header (seqnum)
    - If coded on n bits, seqnum $\in \{0, 1, 2, 3, \ldots, 2^n\text{-}1\}$, **wrap around** after 2^n
  - Receiver confirms reception of a frame by an acknowledgement (ACK)
    - Optimization: acknowledge only last contiguous message seqnum
    - Frames outside the window are discarded
  - The sender 'slides' the window forward as soon as an ACK arrives
  - A
  - Remark: all frames in the window **must be buffered** (may need retransmission)
    - Window size n ➡ **Buffer for n frames required**

Sliding Window: Example

Example with 3 bits for sequence/acknowledgement number, $m = 2^3 = 8$

- Stations agree upon a window size W with $1 \leq W < m$, e.g., $W = 7$
- The window limits the number of unacknowledged frames allowed at one time
  - In this example max. 7, because of $W = 7$
- With receipt of an acknowledgement, the window is shifted accordingly
- Frames are numbered sequentially modulo m (for $m=8$, numbers from 0 to 7)

- Concrete scenario with reception of Acks 0, 2 and 6, sequentially:



Send packet 7

Send packet 10, 11, 12 and 13 (2 to 5 % 8)

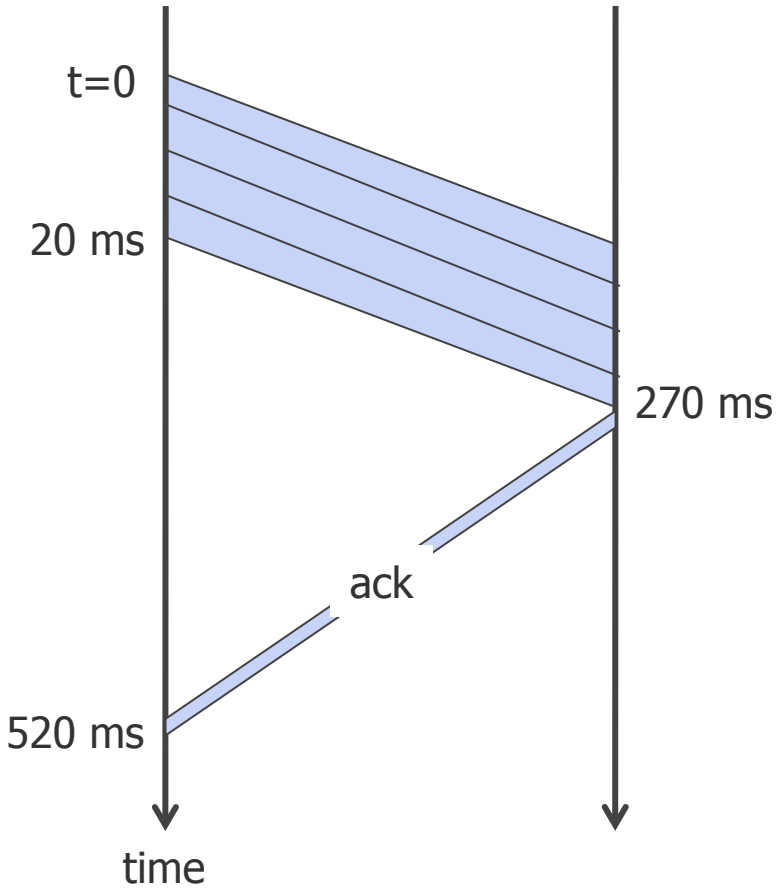
Send packet 8 and 9 (0 and 1 % 8)

Sliding Window: Maximum Window Size

- There is a reason why window size W has to be smaller than 2^n with n being the number of bits with which sequence numbers are represented
  - In the example, sequence numbers have 3 bits
    ➡ sequence numbers {0, …, 7}
- Assume the window size to be $W=8$ and A sends 3 frames to B
  - B acknowledges frame 2 (ACK 2 was sent to A)
    - B has acknowledged 0 1 2
- Now assume A sends 8 frames, without acknowledgements from B
  - A sends 0 1 2 3 4 5 6 7 0 1 2
  - A receives an acknowledgement ACK 2. There are two possibilities
    - Case 1: B only has received 0 1 2
    - Case 2: B has received 0 1 2 3 4 5 6 7 0 1 2
- Ambiguity: A does not know whether case 1 or 2 holds for B!
  ➡ This is the reason for the rule $W < 2^n$

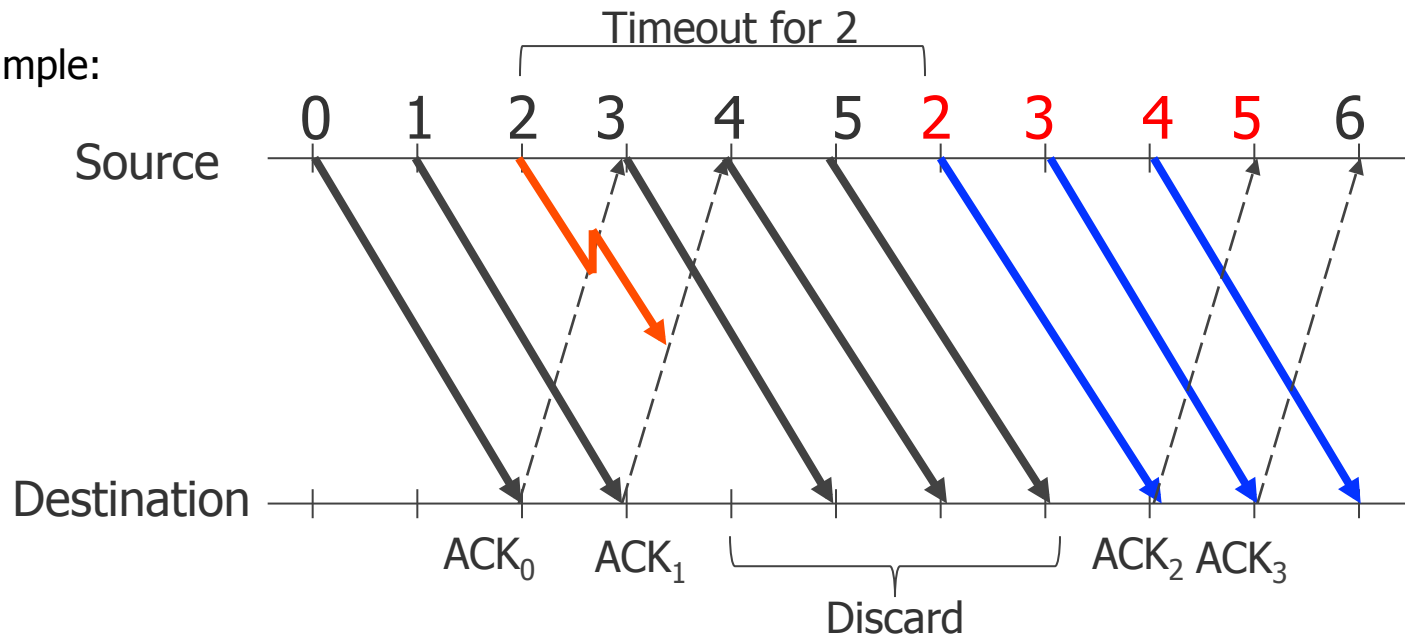
Sliding Window: Effect of RTT

- Long round-trip time is an issue in terms of efficiency
- Example:
  - 50 kbps satellite/GPRS channel with 500 msec round-trip propagation time
  - Sequence numbers over 3 bits
  - Window of $4 < 2^3 = 8$
  - Transmission of 4 frames of 256 bits before sender has to wait for acknowledgements
  - Sender is blocked 500/520=96% of the time
  - Utilization of the channel is only 4%
- If bandwidth × round-trip-delay is large a big window is needed to fill the pipe's capacity
  ➡**Pipelining**
- In the example W=520/5=104
  ➡ Window size should be more than 104
- Need to care about bit errors and packet loss, even more with a big window size!



Sliding Window: Pipelining and Go-back-N

- Go-back-N
  - Sender:
    - Transmits and buffers frames according the window size
    - Each frame has an associated timer (expected time to get an ack)
    - When an ack is received, a new packet is sent and the buffer is updated (window slide)
    - When the timer expires for a frame, the **sender retransmits all buffered frames**
  - Receiver:
    - Correct frames are acked
    - Incorrect frames are discarded and no ack is send
  - Example:



Sliding Window: Pipelining and Selective Repeat

- Selective Repeat (SREPEAT)
  - Receiver
    - When a frame is received correct, send ack
    - When a frame is missing **buffer** following **correct** frames
    - When the missing frame arrives, send an ack for the subsequently received frames
  - Sender
    - Send frames according sliding window
    - If an ack times out, the sender starts repeat all frames (like in Go-back-N)
    - When it receives the ack, the sender stops repeating frames, resumes with new frames

Advantage: capacity is used more efficiently

Drawback: the **receiver needs more buffer**

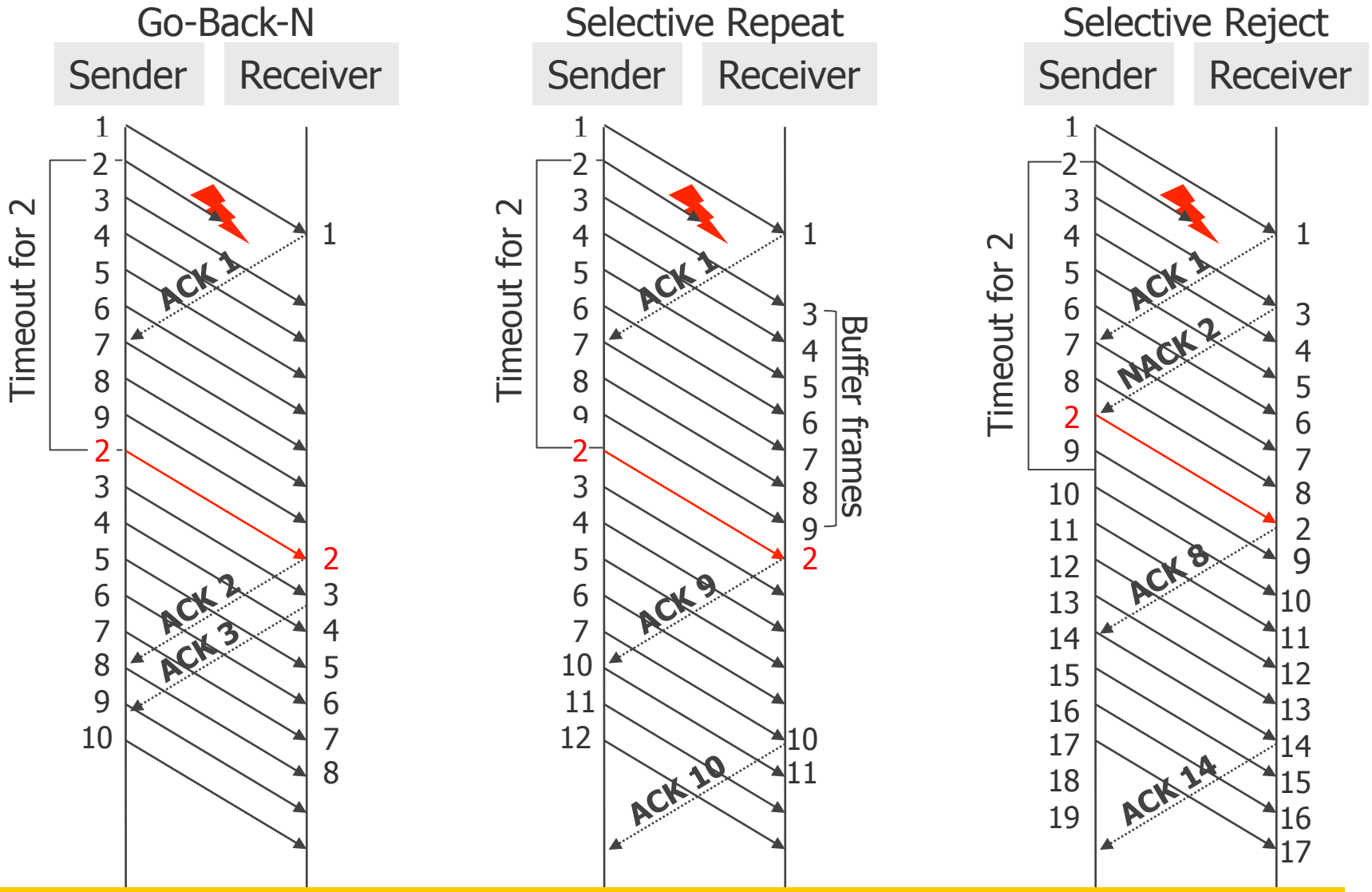
Sliding Window: Pipelining and Selective Reject

- Selective Reject SREJj
  - Receiver:
    - Received frames after a missing frame *j* are buffered (same as sRepeat)
    - Replies with a negative acknowledgement (NACK) for the missing frame *j*
  - Sender:
    - Repeats only frame *j*
- Variant of SREJj with Ack piggybacking
  - The receiver can transmit at once a list of missing frames to the sender, instead of single negative acknowledgements

Advantage: no unnecessary duplicates are transmitted ➡ enhanced efficiency

Drawback: the receiver needs even more buffer

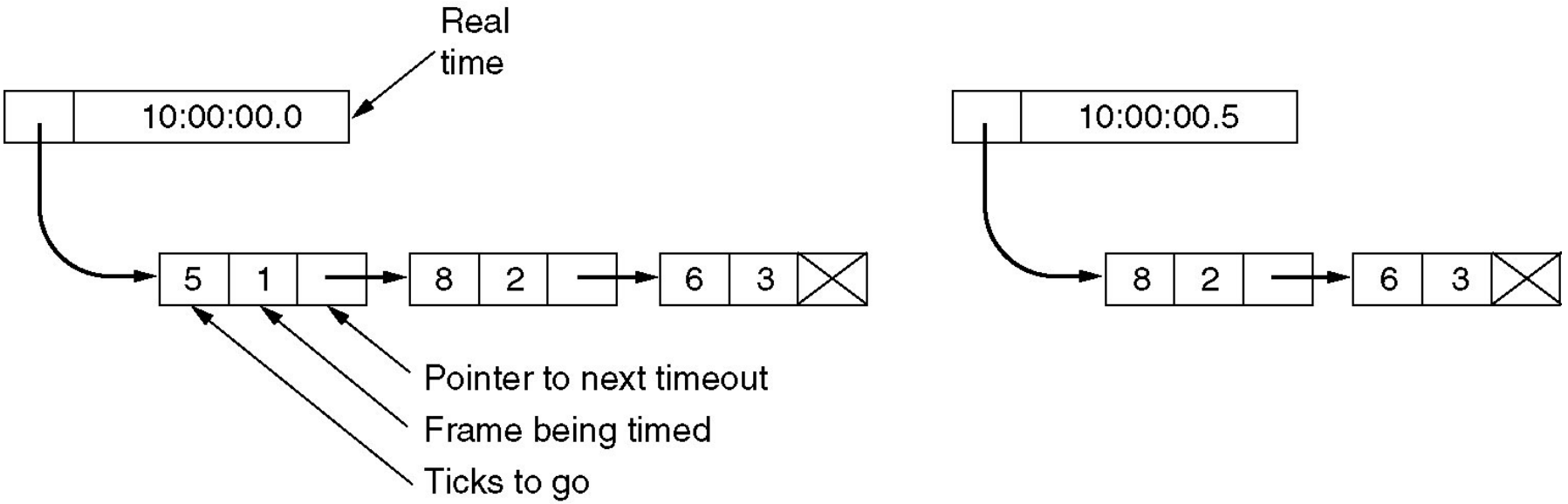
Sliding Window: Summary



BE WARNED - two concepts mixed: error correction via ARQ and flow control!

Remark on Timers

- Some protocols require many timers, but few hardware timers exist
  - Implement timers in software by using one hardware timer
  - Idea: store expiration times in a linked list and update it during protocol runtime



CONTENT of this CHAPTER

- Framing
- Error Detection & Correction
- Flow control
- Multiple Access Control

- Protocols
  - PPP
  - Ethernet
  - Wifi
  - ATM
  - SDH

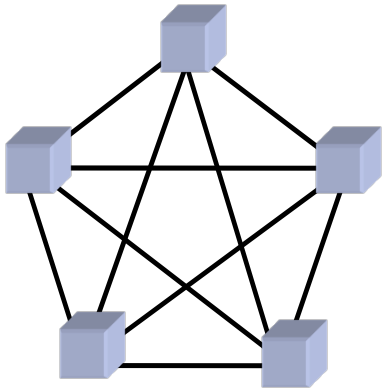
- Infrastructure
  - Physical elements
  - Virtual LANs

Network Interface Cards

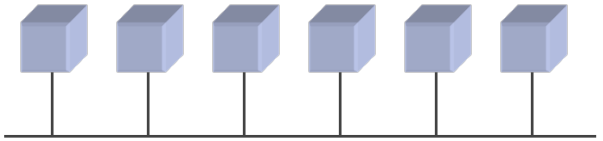
Point-to-Point

NIC for medium shared by $n = 2$ nodes

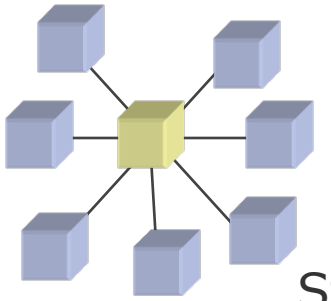
NIC for medium shared by $n > 2$ nodes



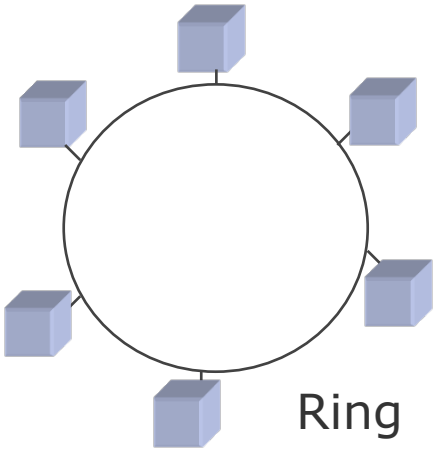
Mesh



Bus



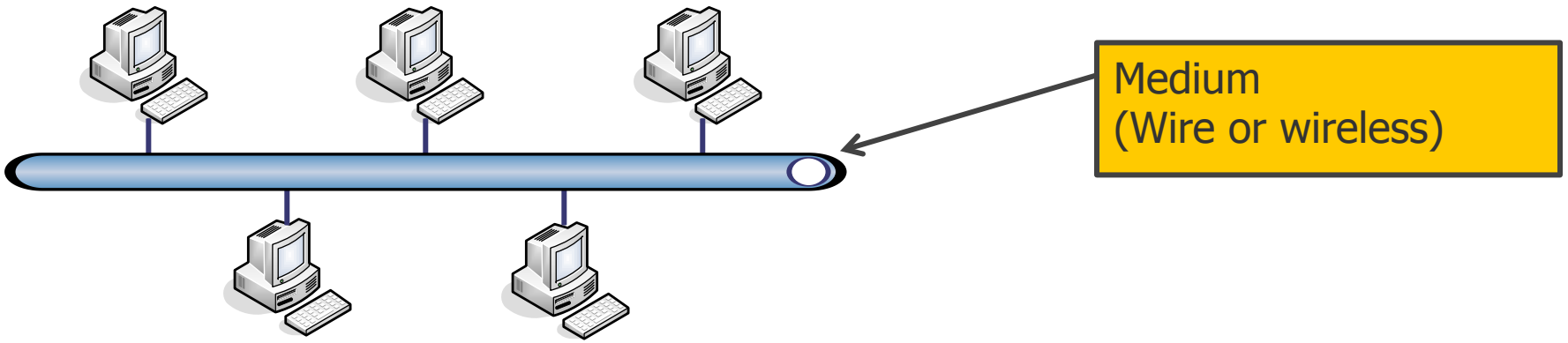
Star



Ring

Network Interface Cards

- a NIC for a medium shared by $n > 2$ nodes must manage specific issues:
  - **Channel allocation**: medium access control (**MAC**)
    - the $n > 2$ nodes want to send through the shared channel
    - Medium access control organizes the order of transmitting nodes on the channel
    - Two types of approaches for medium access control
      - **Distributed MAC**
      - **Centralized MAC**



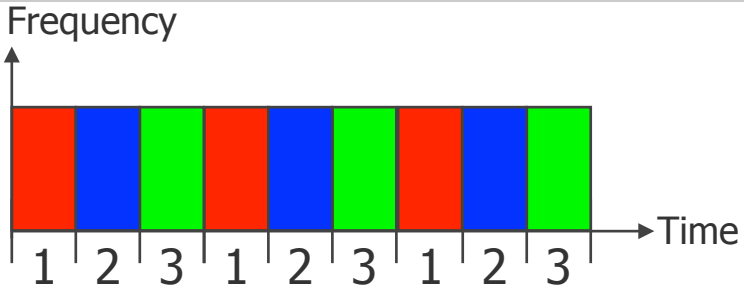
  - **Node identification**: addresses of nodes on the medium (**MAC Addresses**)
    - Typically 6 bytes long, represented in hexadecimal notation
    - Example of MAC address: 01:23:45:67:89:ab

Centralized Medium Access Control (MAC)

Principle: a **master device** manages channel allocation

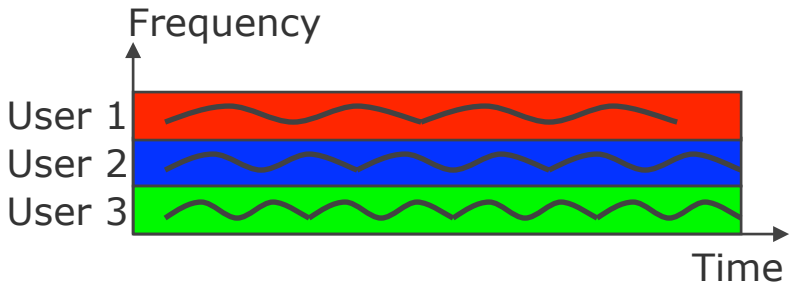
Round-Robin Approach:

- The master device polls each node periodically (TDMA)
  - Each node gets the **entire** transmission capacity for a fixed time interval



FDMA Approach:

- The master allocates a different frequency to each node
  - Each node gets a **portion** of the transmission capacity for the whole time



Drawbacks:

- Users are typically bursty
  ➡ most (sub)channels will be idle most of the time = wasteful!
- Works well for a fixed number of users
  ➡ but complex to adapt to a dynamic number of users

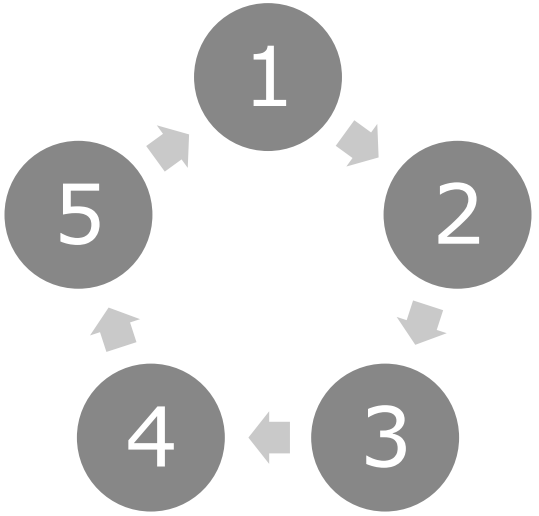
Distributed Medium Access Control (MAC)

- Context of distributed medium access control
  - **Node Model**
    - There are n independent computers (nodes) that generate frames for transmission.
  - **Single channel**
    - A single channel is available for communication. All nodes transmit and receive on it.
  - **Collisions**
    - If two frames are transmitted simultaneously, the signals are garbled, the frames are lost.
  - Time Management
    - **Continuous time**: No master clock, transmission of frames can begin at anytime.
    - **Slotted time**: Time is divided into discrete intervals called slots. Frame transmissions begin always at the start of a slot.

It is necessary to have mechanisms dealing with **multiple access** of channel

Multiple Access using a Token

- Introduction of a token (determined bit sequence)
  - Only the owner of the token is allowed to send
  - Token is cyclically passed on between all nodes
    - particularly suitable for ring topologies



Passing on of the token

- Advantages:
  - Guaranteed accesses, no collisions
  - Efficient utilization of the network capacity
  - Fair, guaranteed response times
- Drawbacks:
  - Complex (dealing with case of lost token)
  - Expensive

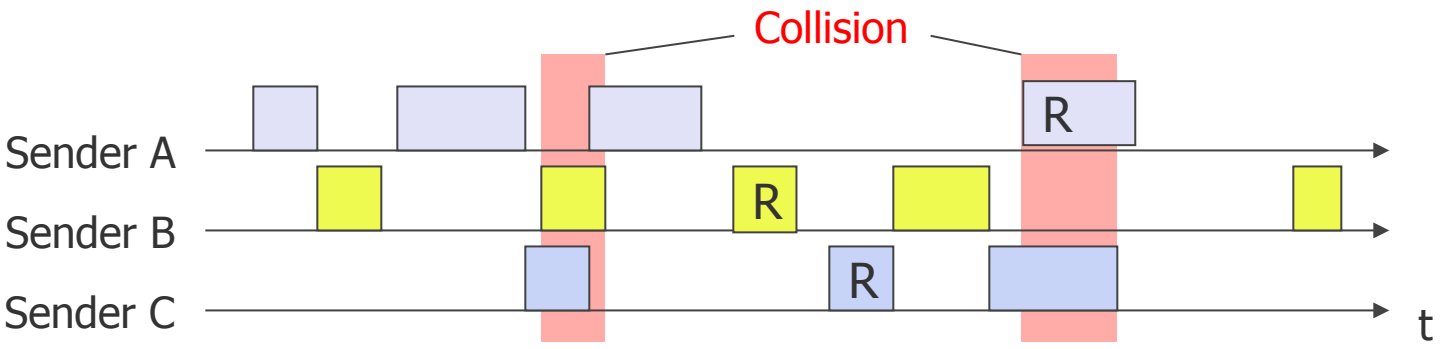
- This technique was used by IBM with Token Ring (4/16/100 Mbps)
  - killed by Ethernet.

Multiple Access using Random: History

- ALOHANET
  - developed by Norman Abramson on the Hawaiian islands in 1970s
  - Network connecting computers on islands over radio
  - Two channels
    - Uplink shared by the clients (collision may occur)
    - Downlink exclusively used by main computer
  - Packets are acked by main computer
  - Good performance under low traffic, but bad under heavy load

Multiple Access using Random: ALOHA

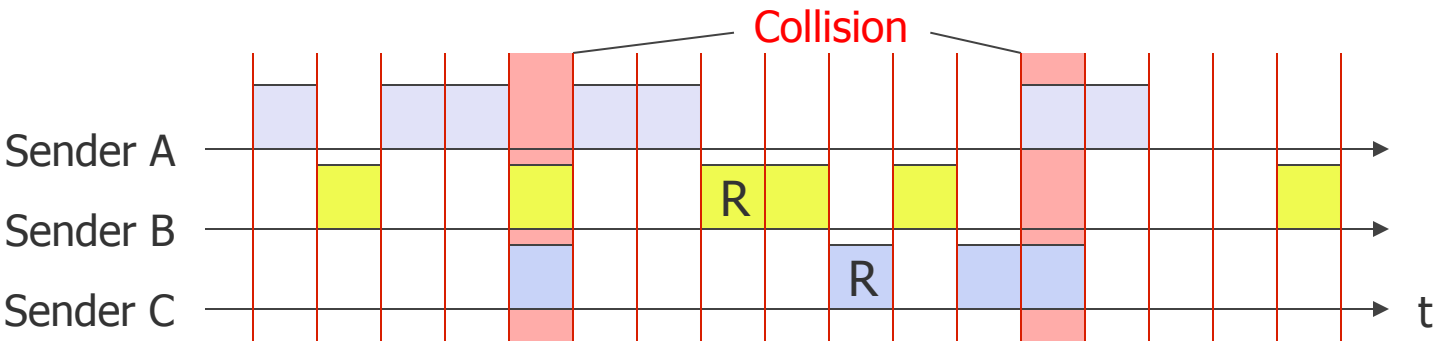
- **ALOHA:** derived from principles used on ALOHANET
  - Principle: similar to a discussion between people. Each person talks when he/she feels like it, stops if there is a collision and then tries again later.
    - Nodes are uncoordinated, send at any time, all using the same frequency
    - When several nodes are sending at the same time, a collision occurs
      - Collisions occur even with very **small overlaps**!
      - **Vulnerability period:** 2 times the length of a frame (assuming equal frame sizes)
    - Collision ➡ frames are lost. Senders each waits a random time, then retransmit



Remark: satellite ➡ long RTT before knowing if transmission went through OK

Multiple Access using Random: Slotted ALOHA

- Problem: even small overlaps result in collisions. With high load:
  - No guaranteed response times
  - Low throughput
- Improvement: Slotted ALOHA
  - Same as Aloha, but the time axis is divided into **time slots**
  - Each sender can send anytime, but must start at the beginning of a time slot



- Advantage: Fewer collisions, **vulnerability period of one frame length**
- Drawback**:** the computers must be synchronized!

Performance of ALOHA & slotted ALOHA

- Evaluation of the performance of ALOHA
  - Model
    - Infinite number of interactive users generating data
    - Transmissions are generated according to a Poisson distribution with intensity λ
      - probablity of k transmission attempts in time interval $[0,t)$ is:

$$P(X=k)=\frac{(\lambda t)^k}{k!}e^{-\lambda t}$$

    - Remark: this counts both transmissions and retransmissions

- Throughput (S) is given by the load (G) and the probability of a successful transmission (P_0)

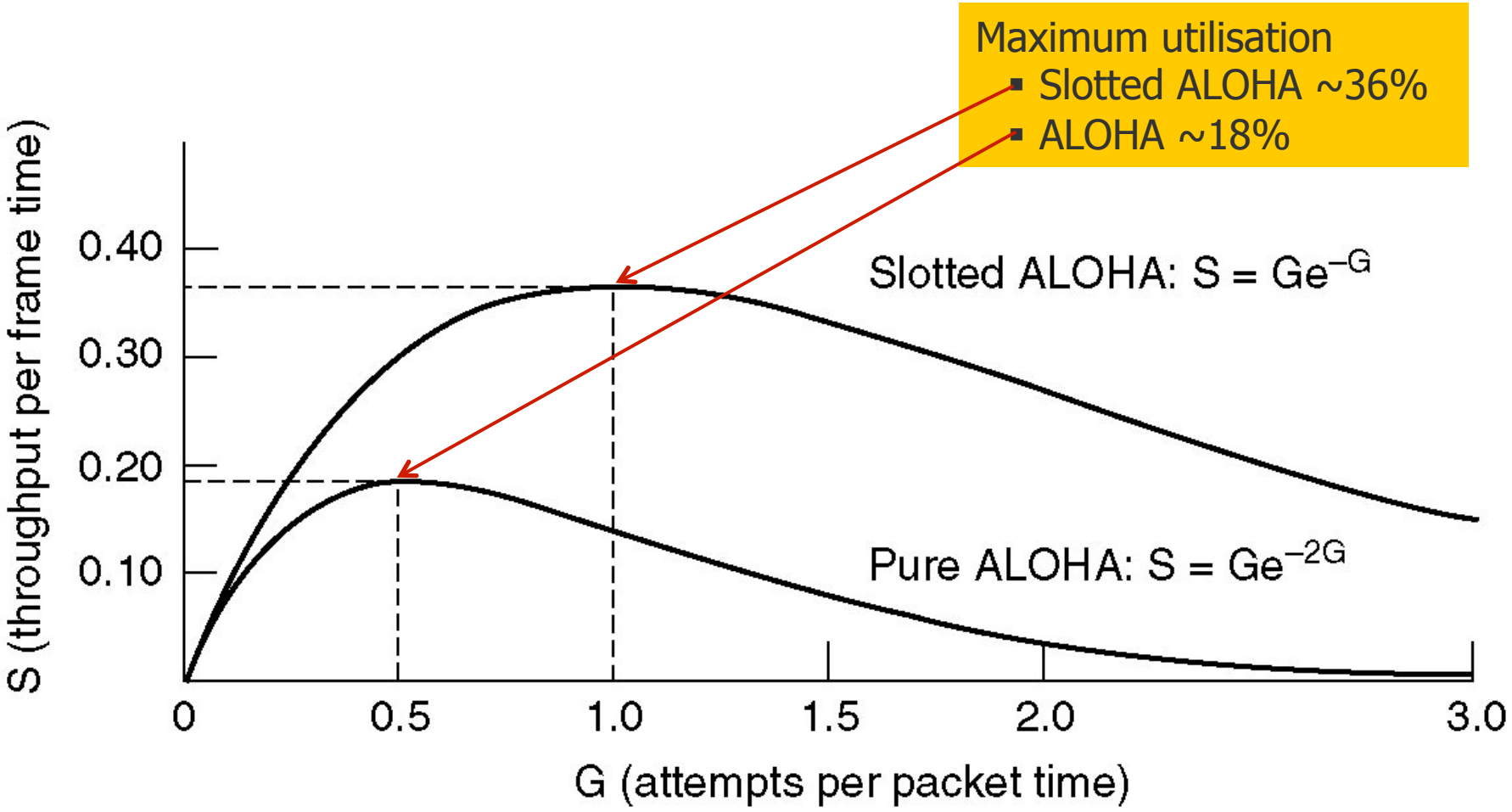
$$S = G \times P_0$$

- What is a successful transmission?
  - A frame is transmitted successful if no other frames are sent within vulnerability period t

$$P_0 = P(X=0) = \frac{(Gt)^0}{0!}e^{-Gt} = e^{-Gt}$$

  - Let's assume time unit is the time to send a single frame, then:
    - With ALOHA vulnerability period = 2 ➡ $S = G\,P_0 = G\,e^{-2G}$
    - With slotted ALOHA vulnerability period = 1 ➡ $S = G\,P_0 = G\,e^{-G}$

Performance of ALOHA & slotted ALOHA



Multiple Access using Random: CSMA

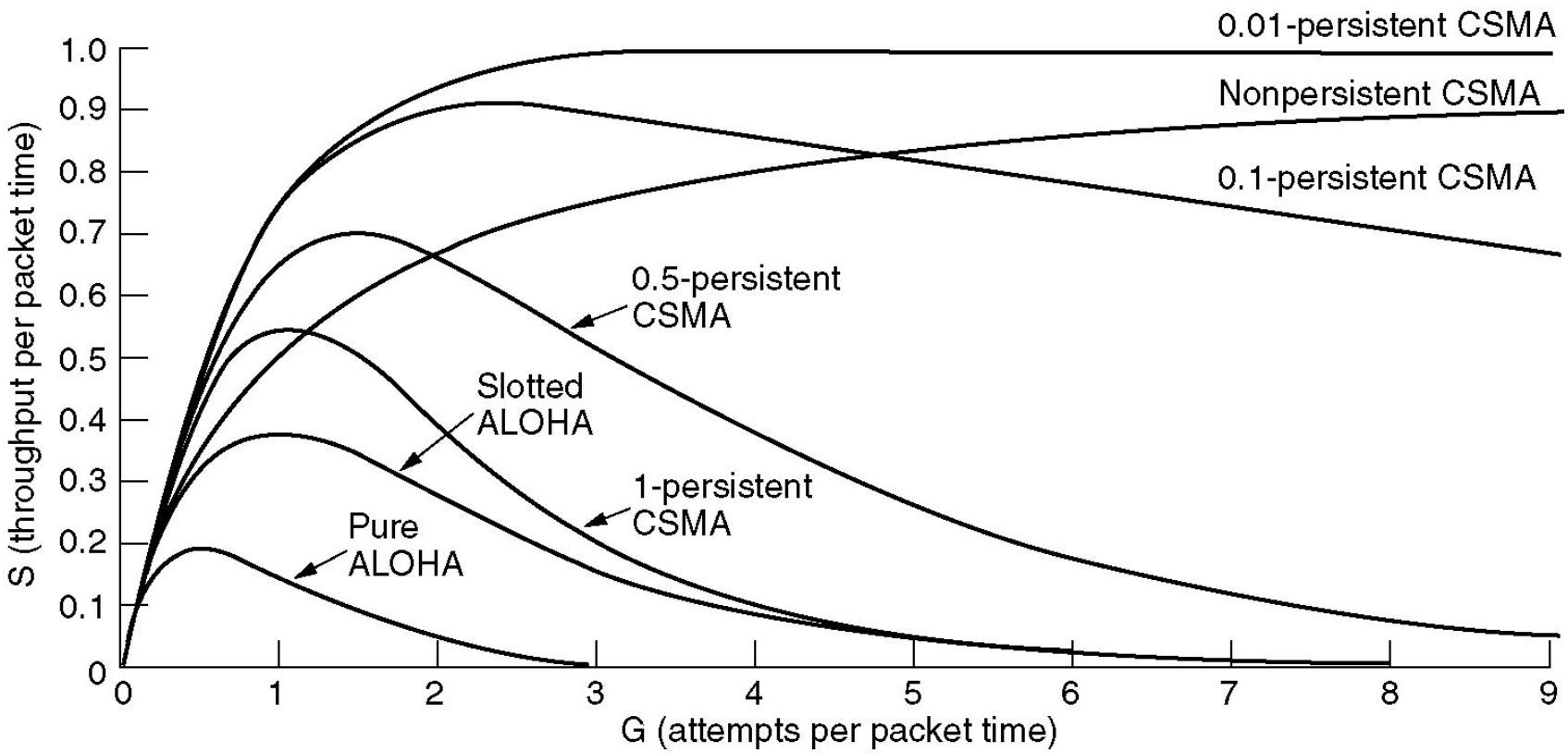
- Problem: ALOHA does not perform well under high load
- Solution: prevent more collisions
  - A sender first examines whether another computer is already transmitting
  - If nobody is currently sending, the station begins to send
  - This mechanism is called **Carrier Sense Multiple Access (CSMA)**
- Advantages: simple (no need for master or tokens) with good utilization of the network capacity
- Drawbacks: no guaranteed medium access, potentially large delays before beginning a transmission is possible
  - Remark: this only works on networks with short transmission delay
    - e.g. it is not possible on satellite networks: no chance to know whether a conflict occurred before the end of the transmission

Persistent and Non-Persistent CSMA

- 1-persistent CSMA
  - Step 1: when a computer has data to send, it first listens to the channel.
  - If channel busy, the computer listens and waits until it becomes idle.
  - When channel is idle, the computer transmits a frame.
  - If a collision occurs, the station waits a random amount of time and goes back to Step 1
- Non-persistent CSMA
  - Step 1: when a computer has data to send, it first listens to the channel.
  - If the channel is busy, the computer waits a random time, and goes back to Step 1
    - No continuous sensing
  - Else, if the channel is idle, the computer transmits.
- p-persistent CSMA
  - Step 1: if the channel is idle, a computer that has data to send transmits with proba. p in current slot and defers until next slot with proba. (1-p) going then back to Step 1.
  - If the channel is busy, the computer waits from the next slot and goes back to Step 1
  - If a collision occurs, the station waits a random amount of time and goes back to Step 1
  - Applicable in slotted time environments (slotted ALOHA)

Performance of CSMA

- CSMA allows more efficient utilization of the available resource compared to ALOHA, especially under high load.



CSMA with Collision Detection (CSMA/CD)

- Problem: collisions between long frames waste the channel during their whole transmission
- Solution: CSMA with Collision Detection: CSMA/CD
  - Computers listen during their transmission
  - If what they hear is different from what they sent (collision) stop transmitting
  - Afterwards wait a random timeout and try again

Pseudo code for CSMA/CD:

```
Backoff_bound = intial_value
Backoff = random(1, Backoff_bound)
TRY: if channel idle, transmit
if collision
                Backoff = random(1, Backoff_bound)
                Backoff_bound = 2 x Backoff_bound
                wait(Backoff)
                goto TRY
```

Exponential backoff

Multiple Access using Reservation Mechanisms

- Non-random approach: reserve the channel to avoid collisions
  - Communication follows in a two-phase scheme (alternating phases)
    - Phase 1: Reservation. The sender makes a reservation by indicating the wish to send data (and in some cases also the length of the data to be sent)
    - Phase 2: Transmission. The data communication takes place (if reservation is OK)

- Advantage: very efficient use of the capacity
- Drawback: delay by two-phase procedure

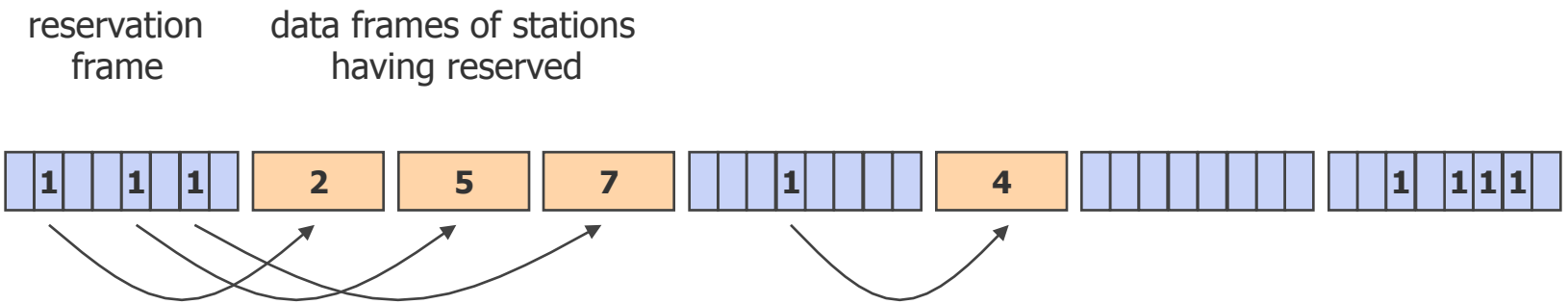
- Centralized reservation
  - A master cyclically queries all other computers whether they have to send data. The master assigns sending rights.

- Distributed reservation
  - Explicit reservation
  - Implicit reservation

Multiple Access using Bit-Map

- The Bit-Map approach: explicit reservation using two types of frames
  - reservation frame (very small) in the first phase
  - data frame (constant length) in the second phase

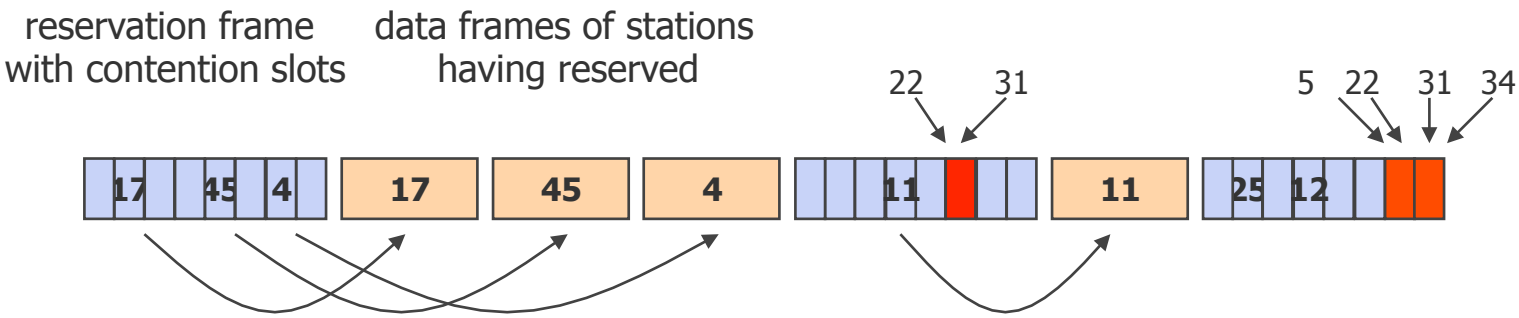
- **Bit-Map without contention** (variant 1)
  - Each user *i* is assigned the *i*-th slot in the reservation frame. If it wants to send data, it sets the *i*-th bit in the reservation frame to 1.
  - After the reservation phase, all stations having set their reservation bit can send their data in the order of their bits in the reservation frame.



  - Remark: this is applicable for small number of users only

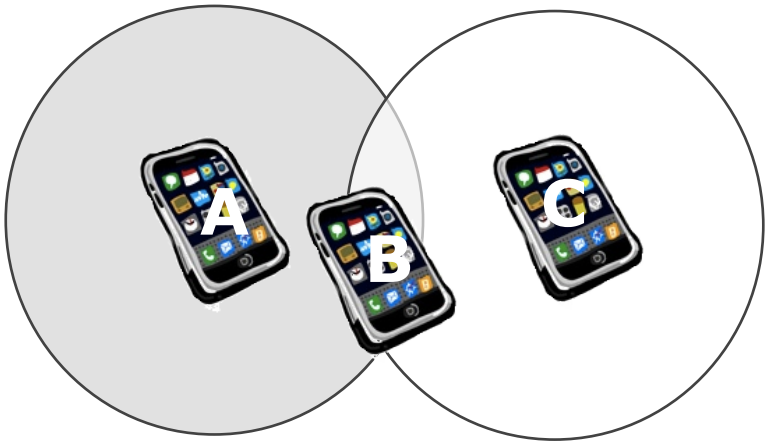
Multiple Access using Bit-Map

- **Bit-Map with contention** (variant 2)
  - The reservation frame consists of a limited number of contention slots (smaller than the number of participating computers)
  - Users try to get a random contention slot (and by that make a reservation for a data slot), writing their computer ID into a slot
  - If there is no collision in the reservation phase, a station may send.
  - Remark: scales to bigger number of users than variant 1



Multiple Access using CSMA/CA & Channel Reservation

- Problem: in some cases, the sender cannot hear what others transmit
  - Hidden terminal problem (with wireless)
    - Example: A does not hear that C is already transmitting to B over the same channel

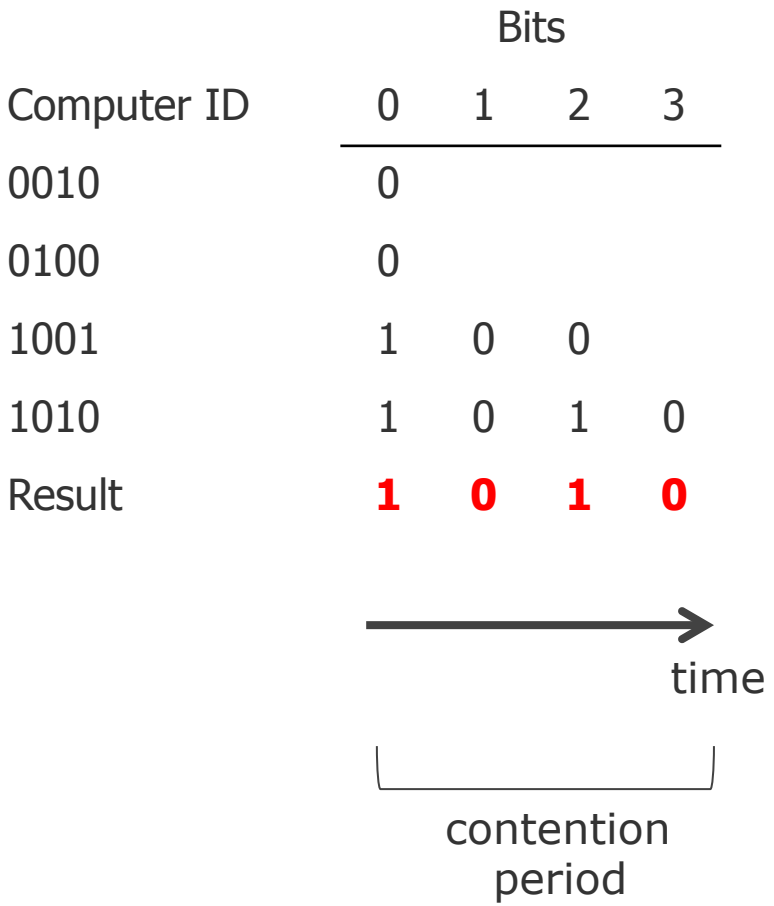


- Solution: the sender gets prior permission from the receiver to send now
  - sender asks « Can I talk to you now ?"
  - if the receiver says "yes", send now
  - if the receiver doesn't confirm ask again later

- CSMA with Collision Avoidance (CSMA/CA) used in WiFi is based on this mechanism

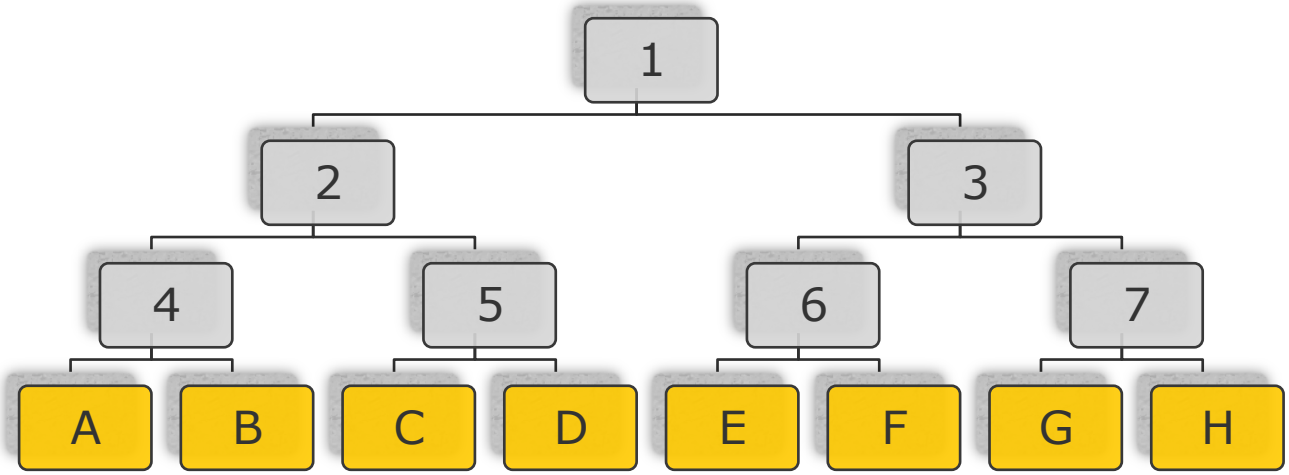
Multiple Access using Binary Countdown

- Implicit reservation with a Binary Countdown
  - Principle: priorities based on ID of the computer
  - Computers with data to send enter a contention phase during which:
    - they broadcast their ID bit by bit
    - high-order bit first
    - 'smaller' addresses give up
  - Example: four stations with addresses 0010, 0100, 1001, 1010
    - 0010 and 0100 give up contending after comparing their first bit
    - 1010 finally gets access



Multiple Access using Adaptive Tree Walk

- Implicit reservation using a limited contention mechanism
- The Adaptive Tree Walk approach
  - Stations are the leaves of a binary tree



  - In the first contention slot following a successful frame, slot 0, all stations (A-H) are permitted to try to acquire the channel
  - If collision, during slot 1 only stations under node 2 (A-D) may compete
    - If one gets the channel, next slot is reserved for stations under node 3 (E-H)
    - If collision, during slot 2, only stations under node 4 (A, B)

The Link Layer: Summary of the Concepts

- The Link Layer manages NIC-to-NIC communication through a link
  - using the bit-by-bit communication service of the Physical Layer
  - provides services to the Network Layer
- Network Layer payload is encapsulated in frames
- Error control is necessary to enable receivers to detect incorrectly transmitted bits within received frames
  - Parity-bit, CRC…
- Flow control is necessary to adjust the rate at which frames should be transmitted
  - Go-back-N, Sliding window…
- Access control needed for NICs on mediums shares by $n>2$ nodes
  - ALOHA, CSMA, Token Ring…
  - Node identification (addressing)