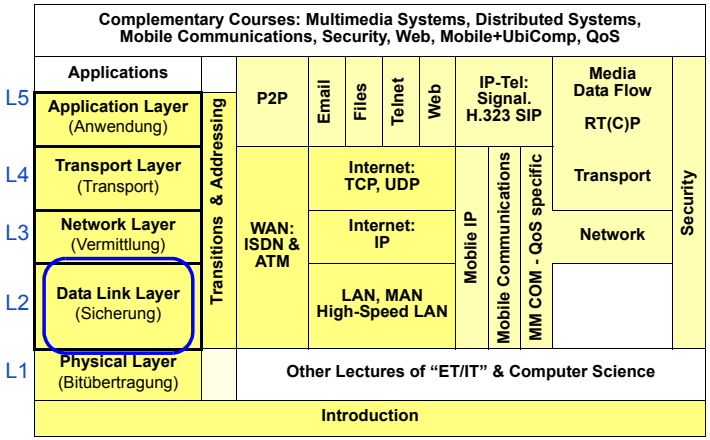
Communication Networks I
Data Link Layer

Prof. Dr.-Ing. **Lars Wolf**

TU Braunschweig
Institut für Betriebssysteme und Rechnerverbund

Mühlenpfordtstraße 23, 38106 Braunschweig, Germany
Email: wolf@ibr.cs.tu-bs.de

Scope



Overview

1. **Function**
2. **Services**
3. **Asynchronous and Synchronous Operation Mode**
4. **Error Detection and Correction**
   4.1 Basics: Code Word, Hamming Distance
   4.2 Error Correction
   4.3 Error Detection
5. **Flow Control and Error Treatment**
   5.1 Utopia
   5.2 Stop-and-Wait
   5.3 Frame Format Example
   5.4 Sliding Window
   5.5 Protocol Comparison
6. **Connection Management**
7. **HDLC: Family**
8. **Internet L2 Protocols**
   8.1 Internet: Serial Line IP (SLIP)
   8.2 Internet: Point-To-Point Protocol (PPP)
9. **Perspective: L2 Communication Design Tasks**

1. Function

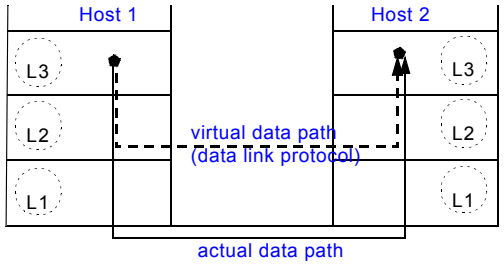
L1 Service:
- **transmission of a bit stream ("unreliable bit pipe")**
  - without sequence errors
- **'malign' features of the L1 service (& the communication channel)**
  - finite propagation speed
    - between sending and receiving on L2
  - limited data rate
  - i.e., loss, insertion and changing of bits possible

L2 Service:
- **(reliable), efficient data transfer between ADJACENT stations (possibly more than 2)**
  - adjacent = connected by a physical channel (coax, telephone line, optical fiber,...)

L2 Functions:
- **data transmission as frames**
- **error control and correction**
- **flow control of the frames**
- **configuration management**

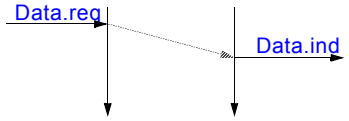
2. Services

Actual data path and virtual data path:



L2 service classes:
- **unconfirmed connection-less service**
- **confirmed connection-less service**
- **connection-oriented service**

Unconfirmed Connection-less Service

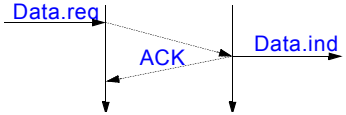


Transmission of isolated, independent units (frames)
- **loss of data units possible**
  - L2 does not try to correct this
  - L2 transmits only correct frames

Features
- **no flow control**
- **no connect or disconnect**

Applications
- **on L1 communication channels with VERY LOW ERROR RATE**
  - corrections will possibly be done at a higher level
- **possibly during real time data transfer like interactive voice communication**
  - timing errors probably more critical than errors in the voice data
- **often used in LANs**

Confirmed Connection-less Service

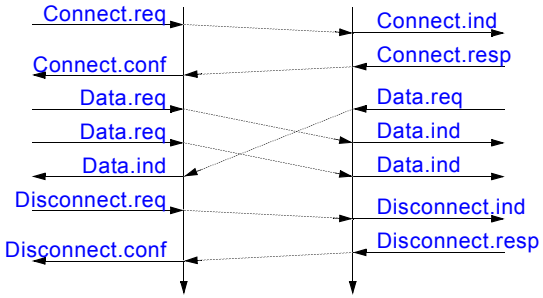


Receipt of data units (implicitly) acknowledged
- **no loss (each single frame is acknowledged)**
- **timeout and retransmit (if sender does not receive an acknowledgement within a certain time frame)**

Features
- **no flow control, no connect or disconnect**
- **duplicates and sequence errors due to "retransmit"**

Application
- **L1 communication channel with high error rate e.g., mobile communication**

L2 Service Class "Connection-Oriented Service"



Connection = error free channel
- **no loss, no duplication, no sequencing error**
- **flow control**

3-phased communication
1. **connect**
   - initializing the counters/variables of the sender and receiver
2. **data transfer**
3. **disconnect**

L2 Services: Comments

Acknowledging on L2:

- **is only for optimization but is not indispensable**
- **because this can also be done at a higher level (L4)**

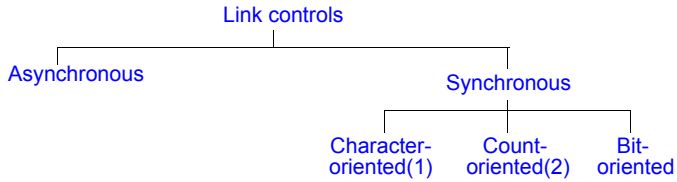
however

- **L4 message usually consists out of n (e.g., 20) L2 frames**
  - if there is an error in a frame => the whole message will be retransmitted
  - that means loss of time and efficiency

End-to-end argument:

J.H. Saltzer, D.P. Reed, D.D. Clark:
"End-to-End Arguments in System Design",
ACM Transactions on Computer Systems,
Vol. 2, No. 4, November 1984, pp. 277-288

3. Asynchronous and Synchronous Operation Mode

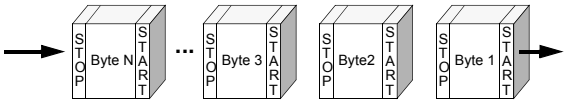


(1) also called Byte-oriented

(2) also called Block-oriented

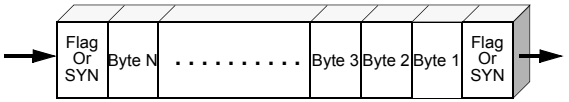
Asynchronous and Synchronous (2)

Asynchronous transmission



- **each character is bound by a start bit and a stop bit**
- **simple + inexpensive, but low transmission rates, often up to 200 bit/sec**

Synchronous transmission
(to be discussed in more detail in the following)



- **several characters pooled to frames**
- **frames defined by SYN or flag**
- **more complex, but higher transmission rates**

Synchronous Data Transmission: Framing

L2 forms a frame from the L1-bits; which

- **(as a unit) undergoes error treatment**

Possibilities for definition and recognition of frame boundaries

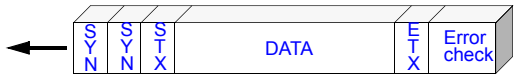
- **bound frames by idle times**
  - problems:
    - networks (L1) usually have no suitable notion of time,
    - possibly loss of efficiency

1. **character-oriented**
2. **count-oriented**
3. **bit-oriented**
4. **using invalid characters of the physical layer**

Comment

- **Combinations may be used in L2:**
  - count-oriented and bit-oriented
  - the transmission is error free only if both match

Character-Oriented Protocols



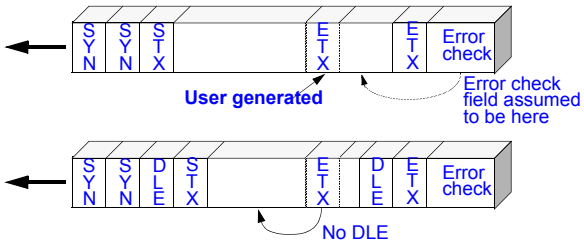
STX: Start of Text
ETX: End of Text

Control Fields
- **flag frame areas**
- **depends on encoding (e.g., ASCII, EBCDIC)**

Character-Oriented Protocols (2)

Problem: user data may contain "control characters"

Solution: CHARACTER STUFFING

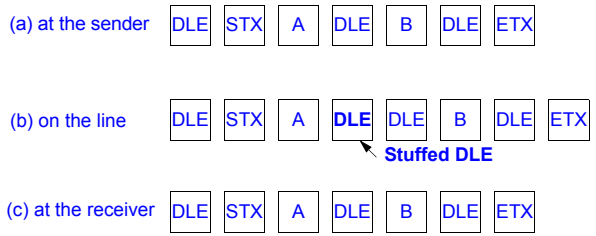


- **SENDER: each control character is preceded by a DLE (Data Link Escape), (but not in user generated data)**
- **RECEIVER: only control characters preceded by DLEs are interpreted as such**

Character-Oriented Protocols (3)

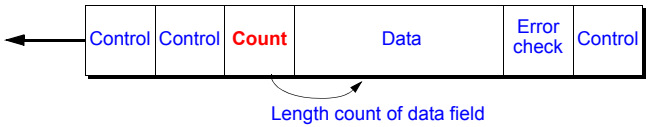
Problem: user generated data contain DLE

Solution:
- **the sender inserts an additional DLE before the DLE in the user's data**
- **the receiver ignores the first of two back-to-back DLEs**



Disadvantages:
- **DLE insertion requires additional efforts/time**
- **usually a derivation of an ASCII 8 bit encoding**
- **conversion if codes are different**

Count-Oriented Protocols



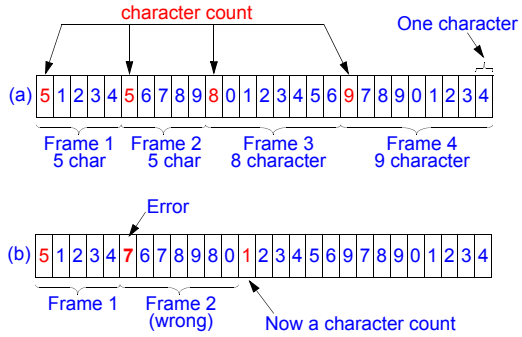
Frame contains LENGTH COUNT FIELD

PROBLEM: Transmission error destroys length count
- **sender and receiver are not synchronized anymore**

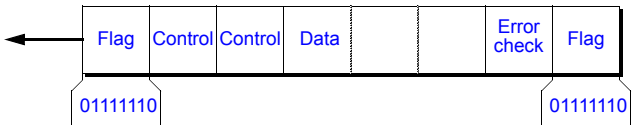
that means
- **where does the next frame start?**

Where do retransmitted frames start?

⇒ **Therefore not spread widely!**



Bit-Oriented Protocols



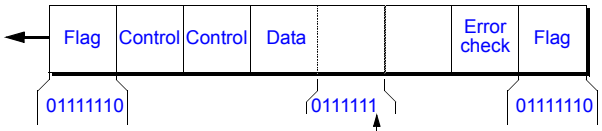
Most of today's protocols
- **independent from encoding**
- **block definition**

  `flag (01111110)`

Start / end
- **may be different flags**
- **typically identical**

Bit-Oriented Protocols (2)

PROBLEM:
- **"flag" in user data**



SOLUTION:
- **bit stuffing**

0 Bit stuffed

(a) at the sender 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

(b) transmitted 0 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 1 1 1 1 0 1 0 0 1 0

Stuffed Bits

(c) at the receiver 0 1 1 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 1 0

- **SENDER**
  - inserts a "0" bit after 5 successive "1" (only in the user data stream)
- **RECEIVER**
  - suppresses "0" after 5 successive "1"

Protocol with Invalid Characters

Invalid
- **with regard to the layer in consideration: in this case the physical layer**

Method
- **L1 defines digital encoding**

Example
- **Return to Zero (RZ)**
  - 1:clock pulse (double frequency) during the interval
  - 0: low level
- there is always a combination of "high-low" or a sequence of "low"
- there is never a "high-high" combination (invalid symbol)
  - i.e., define an invalid symbol in L2 as the bit boundary

Comment
- **actually inconsistent with the layer model**

4. Error Detection and Correction

BIT ERROR: **Modification of single bits**

BURST ERROR: **Modification of a sequence of bits**

Causes for errors:
- **thermic noise:**
  - electron movement generates background noise
- **impulse disruptions (often last for 10 msec):**
  - cause: glitches in electric lines, thunderstorms, switching arcs in relais, etc.
  - most common cause for errors
- **crosstalk in adjacent wires**
- **echo**
- **signal distortion (attenuation is dependent on frequency)**

⇒ **errors usually occur in bundles: BURST ERROR**

Error detection:
- **inserting redundancies so that receiver is able to DETECT an error**
- **no error correction: usage of separate method, if needed**
  - e.g., retransmission

Error correction:
- **inserting redundancies so that receiver is able to DETECT & CORRECT an error**

4.1 Basics: Code Word, Hamming Distance

Frame (= code word) contains
- **data**
- **checking information**

Code = set of all valid code words

Hamming distance of two words w1 and w2:
- **number of bit positions by which w1 and w2 differ**
- **example:**

```
      w1  10001001
XOR   w2  10110001
      =   00111000   i.e.   d = 3
```

Hamming distance of a code:
- **minimum Hamming distance between two words of a code**
- **example:**

```
      w1  10001001
      w2  10110001
      w3  10110011
```

because the Hamming distance w2.w3=1 is the smallest, the Hamming distance of the code is d = 1

Basics: Detection (according to Hamming)

Hamming Distance determines
- **a code's error detection and correction properties**

1. **DETECTION of *f* 1-bit errors:**
   - if the Hamming distance of the code
     $d \geq f + 1$
   - i.e., ***f*** and less errors generate an invalid code word
   - example:

```
     parity bit
         P
0    0   0
0    1   1
1    0   1
1    1   0
```

d = 2:
i.e. maximum value for ***f***: ***f***=1
detection of a 1-bit error

Basics: Correction (according to Hamming)

2. **CORRECTION of *f* 1-bit errors:**
   - if the Hamming distance of the code
     $d \geq 2f + 1$
   - ***f*** and less errors transcribe word ***w*** into an invalid word, which is "closer" to ***w*** than to any other word
   - example: d=5 ⇒ ***f***= 0,1 or 2

```
0  0  0  0  0  0  0  0  0  0
0  0  0  0  0  1  1  1  1  1
1  1  1  1  1  0  0  0  0  0
1  1  1  1  1  1  1  1  1  1
```

   - correction of two 1-bit errors (***f*=2**) in the following word:

```
0  0  0  0  0  0  0  1  1  1  ⇒
```

   - result

```
0  0  0  0  0  1  1  1  1  1
```


4.2 Error Correction

CORRECTION of *f* 1-bit errors:
- if Hamming distance of the code

$$d \geq 2f + 1$$

Lower bound for the number of check-bits for correcting 1-bit errors:

$(m + r + 1) \leq 2^r$

m is # data bits; **r** is # **check-bits**

e. g.

m = 8 ⇒ **r** = 4
m = 1000 ⇒ **r** = 10

Procedure:
- **according to Hamming, 1950**

Correction of burst errors (up to length *k*)
- **treat *k* consecutive codewords as matrix**
- **transmission by columns**

Properties:
- **only possibility for simplex operation**
- **but high redundancy in each block**

⇒ **usually less efficient than error detection**

4.3 Error Detection

CYCLIC REDUNDANCY CODE (CRC) is one of the error detection procedures

Basic idea:

- **bit strings are treated as polynomials**

n-bit string: $k_{n-1} \cdot x^{n-1} + k_{n-2} \cdot x^{n-2} + \ldots + k_1 \cdot x + k_0$ with $k_i = [0,1]$

Example:

1 1 0 0 0 1 ⇒ $x^5 + x^4 + 1$

Polynomial arithmetics: modulo 2 ("algebraic field theory")

- **addition and subtraction: XOR**

```
Sender                                  Receiver
     /* transmission of block B */
     B(x)/G(x) = Q(x)+ R(x);
                 (B, R)
send ------------------------> receive;
                        (B(x)-R(x))/G(x) =Q(x)+R'(x)
                        if R'(x) = 0
                            then Accept B
                            else Reject B
```


Error Detection (2)

Algorithm

B(x) ... Block polynomial

G(x) ... Generator polynomial of degree *r*

- ***r* < degree of B(x)**
- **highest and lowest order bit = 1**

1. **Add *r* 0-bits at the lower order end of B.**

Let result be B^E and corresponds to: $x^r \cdot B(x)$

2. **Divide $B^E(x)$ by G(x)**
   - modulo 2: subtraction and addition correlate to XOR
   - result: Q(x) + R(x)

3. **Subtract R(x) from B^E(modulo 2)**

Transmit the result.

Error Detection (3)

Example: frame: **1101011011**
Generator G(x), degree 4: 10011
Frame with 4 attached 0-bits: **1101011011**0000

```
          1100001010
10011 | 11010110110000
        10011
         10011
         10011
          00001
          00000
           00010
           00000
            00101
            00000
             01011
             00000
              10110
              10011
               01010
               00000
                10100
                10011
                 01110
                 00000
```

Rest → **1110**

Transfered frame: 1101011011**1110**
Transfered frame: 11010110111110

Error Detection (4)

Standardized polynomials:

CRC - 12 = x12 + x11 + x3 + x2 + x1 + 1
CRC - 16 = x16 + x15 + x2 + 1
CRC - CCITT= x16 + x12 + x5 + 1

CRC - CCITT recognizes

- **all simplex and duplicate errors**
- **all errors with odd bit numbers**
- **all burst errors up to a length of 16**
- **99,99 % of all burst errors of a length of 17 and more**

Implementation

- **looks complicated but can be done using simple shift register in hardware**
- **virtually all LANs use it**

Assumptions & analyses

- **have been made for long time assuming frames contain random bits**
- **recent work inspecting real data showed this to be wrong**
  - undetected errors are more common than previously assumed

5. Flow Control and Error Treatment

Basics, problems statement:
- **sender can send faster than receiver can receive**

WITHOUT FLOW CONTROL:
- **sender can send faster than receiver can receive**
- **that means that the receiver loses frames despite error-free transmission**

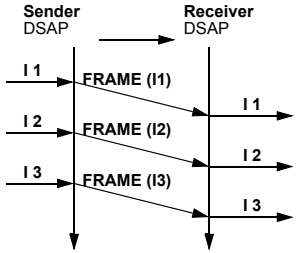
WITH FLOW CONTROL:
- **sender can adapt to receiver's abilities by feedback**

Comment:
- **error control and flow control are usually interlinked**
- **rate control (in contrast to flow control)**
  - reference to frame sequence (not single frames)
  - used with continuous-media data (audio, video)

5.1 Utopia

Protocol 1

Assumptions:
- **error-free communication channel**
- **receiving buffer infinitely large**
- **receiving process infinitely fast**



DSAP: Data link (layer) Service Access Point

but: finite buffer, finite processor performance
...

sender floods receiver with data faster than latter is able to process

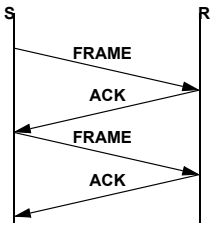
5.2 Stop-and-Wait

Protocol 2

Assumptions:
- **error-free communication channel**
- **NOT [infinitely large receive buffer]**
- **NOT [receiving process infinitely fast]**
  - but always fast enough to process a frame

Further
- **simplex mode for actual data traffic**
  - acknowledgement requires at least semi-duplex mode

Flow control necessary: STOP-AND-WAIT



- **receive buffer for a frame**
- **communication in both directions (frames, ACKs)**

but: additionally, noisy communication channel (loss of frames)...

Stop-and-Wait / PAR

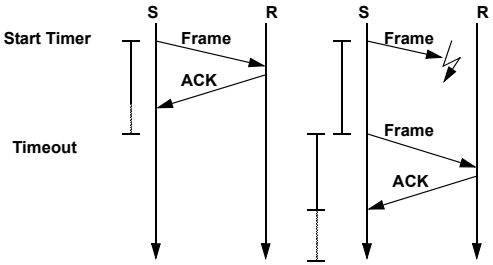
Protocol 3a

Assumptions:
- **NOT [error-free communication channel]**
- **NOT [infinitely large receiving buffer]**
- **NOT [receiving process infinitely fast]**

Problem: protocol 2 locks down during loss of both frames and ACKs

Solution:
- **PAR (Positive-Acknowledgement with Retransmit)**
- **also called ARQ (Automatic Repeat reQuest)**

Timeout interval:
- **TOO SHORT: unnecessary sending of frames**
- **TOO LONG: unnecessary long wait in case of error**

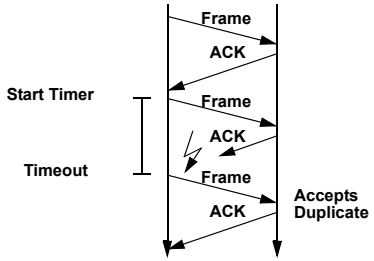


but: in addition if ACK is lost
...

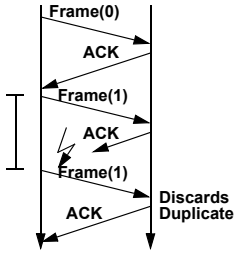
Stop-and-Wait / PAR / SeqNo

Protocol 3b

Problem:
- **loss of ACKs leads to block duplication**



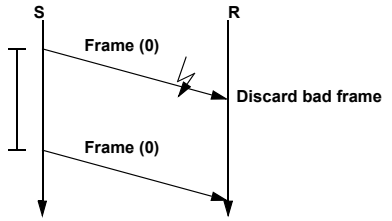
Solution: sequence numbers
- **each block contains a sequence number**
- **sequence number is kept during retransmissions**
- **range**
  - Stop-and-Wait: 0,1
  - in general: [0, …, k], $k = 2^n-1$
    n: window size (here: n=1)



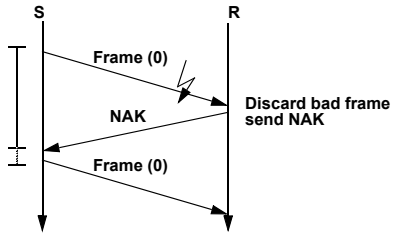
Stop-and-Wait / NAK / SeqNo

Protocol 3c

Until now: Passive error control
- **no differentiation between**
  - missing frames and
  - faulty frames
- **if few errors occur**
  - a lot of traffic as ACK
  - i.e. overhead

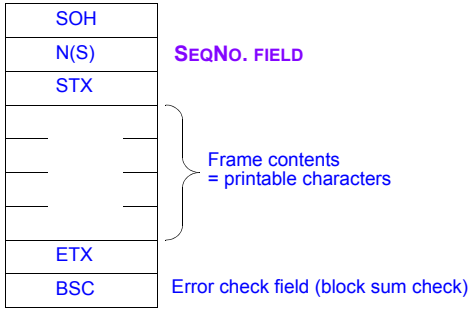


Active error control
- **only negative ACK**

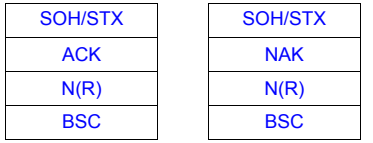


5.3 Frame Format Example

E.g.: using a character-oriented protocol



I-frame format (information, data frame)



ACK-frame format

NAK-frame format

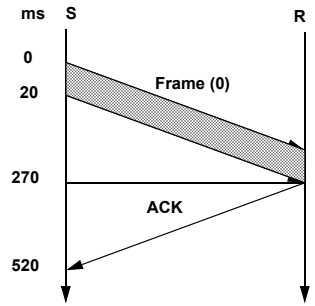
5.4 Sliding Window

Stop-and-Wait, problems:
- **undefined state (parallelism) with simultaneous**
  - lost frames, damaged frames and
  - premature time-out
- **poor utilization of the channel**

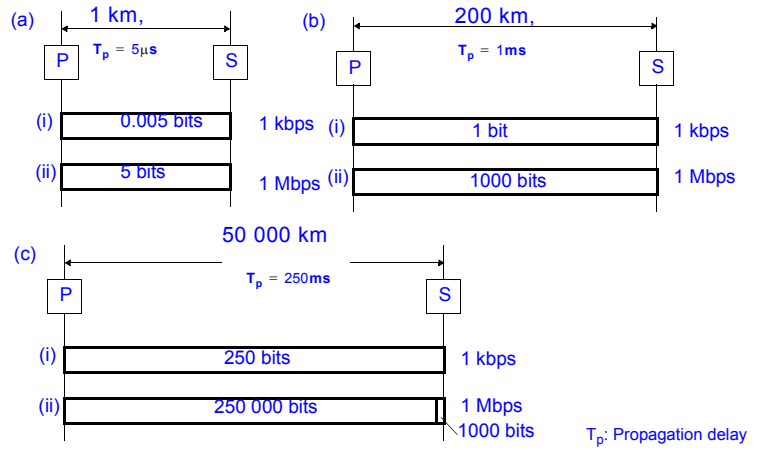
Example: satellite channel
- transmission rate: 50 kbps
- roundtrip delay 500 ms (2*250 ms)
- frame size: 1000 bit
- in comparison; ACK is short and negligible

this means
- sending takes
  1000 bit / 50.000 bps = 20 ms
- sender is blocked for 500 ms of 520 ms

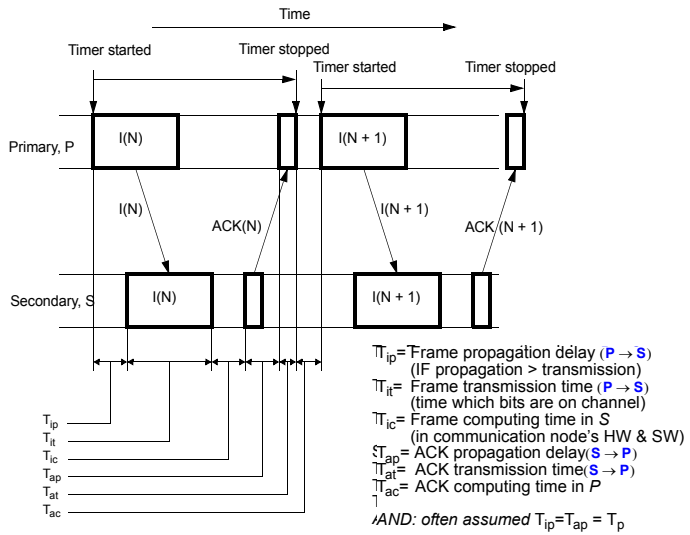
⇒ **Channel utilization < 4%**



Channel Utilization and Propagation Delay



Channel Utilization and Propagation Delay (2)



Channel Utilization and Propagation Delay (3)

exact formula (note: some values based on assumptions):

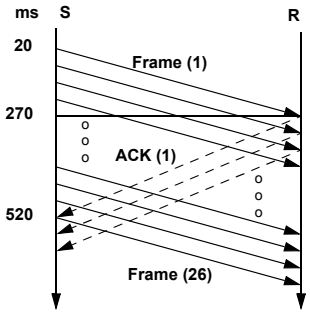
$$U = \frac{T_{it}}{\sum T_{information + acknowledgement}} = \frac{T_{it}}{T_{ip} + T_{it} + T_{ic} + T_{ap} + T_{at} + T_{ac}}$$

approximated formula:

$$U = \frac{T_{it}}{T_{it} + 2T_{ip}} = \frac{1}{1 + 2\frac{T_{ip}}{T_{it}}}$$

Hence: Pipelining ... Sliding Window

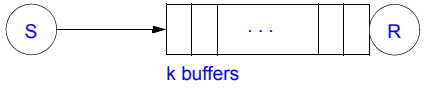
Solution: pipelining



Flow control: sliding window mechanism

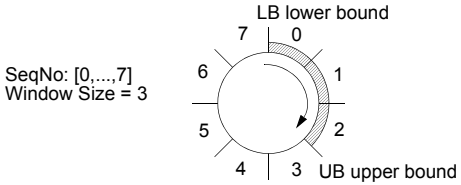
Sliding Window: Concept

Flow control: receiver buffer must not be flooded



Sender and receiver window per connection (communication) relationship

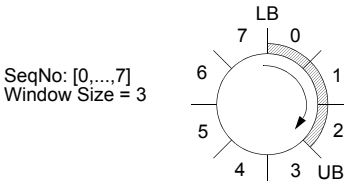
- **R-WINDOW:**
  - sequence numbers, which can be accepted
- **S-WINDOW:**
  - sequence numbers, which were sent but not yet acknowledged



Initial window size:

- **R-Window: number of buffers reserved**
- **S-Window: maximum number of blocks, which may still be open (waiting) for acknowledgement**

Sliding Window: Concept (2)



Lower Bound & Upper Bound:

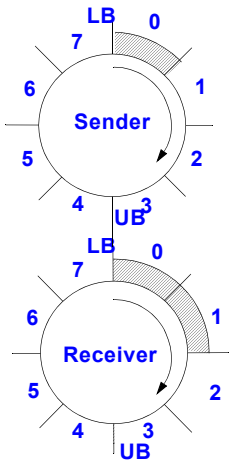
	Sender	Receiver
LB	oldest not yet confirmed SeqNo	next expected SeqNo
UB	next SeqNo to be sent	highest SeqNo to be accepted

Manipulation

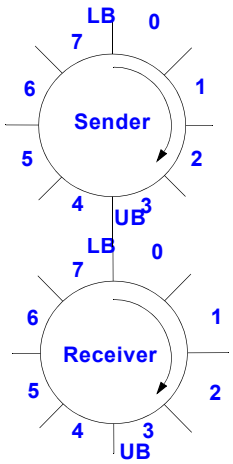
- **increment(LB), increment(UB), if**

	Sender	Receiver
LB	on reception of an ACK	on reception of a frame
UB	when sending a frame	when sending an ACK

Sliding Window: Concept (3)



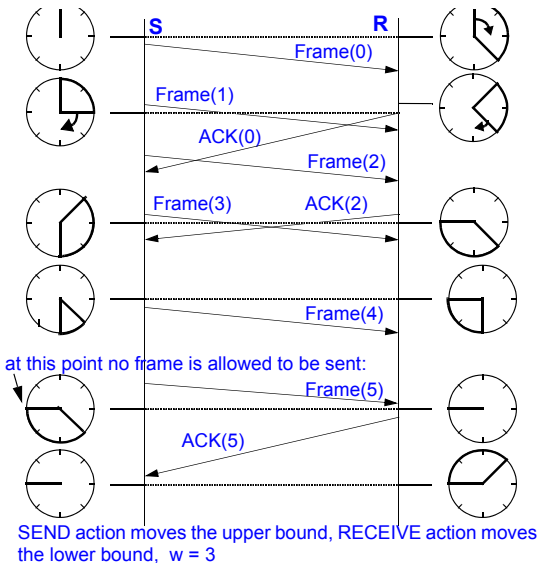
Sliding Window: Concept (4)



Sliding Window: Including Acknowledgement

Including Acknowledgement
- **ACKs contain SeqNo**
- **that means ACK(SeqNo) confirms all frames(SeqNo') with**
  - SeqNo' = SeqNo
  - and
  - SeqNo' before SeqNo

Example:



SEND action moves the upper bound, RECEIVE action moves the lower bound, w = 3

Sliding Window: Remarks

Stored frames at the sender
- **maximum number defined by sender's window size (here: 3)**
- **the frames which are not yet acknowledged by the receiver**

Stored frames at the receiver
- **maximum number determined by receiver's window size (here 3)**
- **the frames which are not yet acknowledged to the sender**

ACK sent by receiver if frame
- **has been identified as being correct**
- **has been transmitted correctly to the network layer (or a corresponding buffer)**

Applet

`http://www.kom.e-technik.tu-darmstadt.de/projects/iteach/itbeankit/Applets/Sliding_Window/sliding-window/index.html`
- **initialization (additionally!) different**
  - send window LB=UB=0 (as described here)
  - receiving window LB=UB=0 (different)

Sliding Window: Examples

Sender: Sliding Window	Stored Frames	Situation
	0	in this case sender may send up to 3 frames
	2	in this case sender may send 1 frame
	3	sender is not permitted to send anything, sender's L3 must not transmit further data to L2

Sliding Window: Influence of the Window Size

Expected order
- **if window size 1**
  - sequence always correct
- **if window size n (n>1)**
  - no requirement to comply with the sequence
  - but is limited by the window size

Among other things the efficiency depends on
- **type and amount of errors on L1**
- **amount and rate of data**
- **end-to-end delay on L1 (e.g., satellite)**
- **window size**

Resources and quality of service
- **if the window size is small**
  - generally shorter end-to-end delays at the L2 service interface
  - less memory neecessary per L2 communication relation

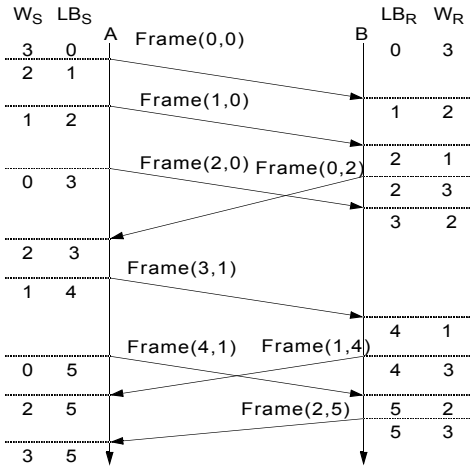
Sliding Window: Piggybacking

Frames may contain implicit ACKs

- **duplex operation**

```
Frame(SeqNo,
      ACK-SeqNo,
      ...Data...)
```

- **comment: in the example always the next expected SeqNo. is given**



with W (Window Size Sender/Receiver)
LB (Lower Bound Sender/Receiver)

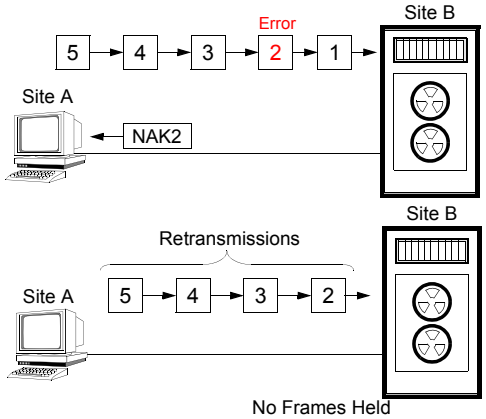
Sliding Window: Error Treatment: Go-Back-N

Procedure

- **after a FAULTY FRAME has been received**
  - the receiver **DROPS** all **FURTHER FRAMES** until the correct frame has been received

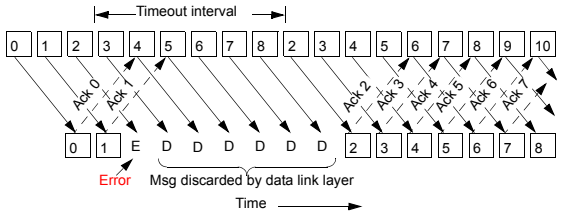
Evaluation

- simple
- no buffering of "out-of-sequence" frames necessary (only for optimization purposes)
- poor throughput

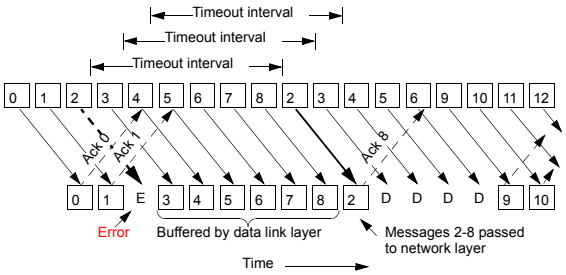


Sliding Window: Error Treatment: Go-Back-N (2)

Example: sender: error detection by timeout



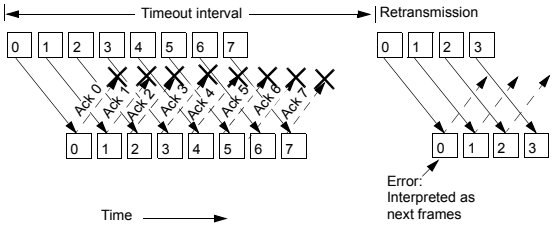
Optimization



Sliding Window: Maximum Window Size

Example:

- **number of sequence numbers** **8**
- **window size** **8**
- **all ACKs lost**



Correlation between

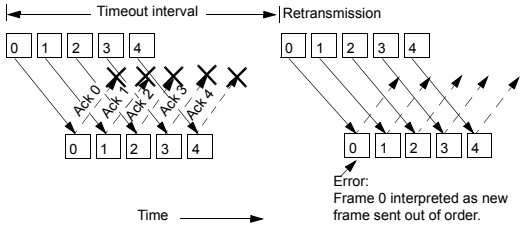
- **window size and**
- **number of possible sequence numbers:**

⇒ **max. window size < range of sequence numbers**

Sliding Window: Maximum Window Size & Frame Sequence

If the sequence is arbitrary, the following situation may occur, for example:

- number of sequence numbers 8
- window size 5
- all ACKs are lost, and the frame that has been lost last is the first one to arrive at the receiver again



Correlation between window size and number of possible sequence numbers:

⇒ **max. window size <= 1/2 range of sequence numbers**

- during Go-back-N (otherwise possibly different)

Sliding Window: Error Treatment: Selective Repeat

Procedure

- **receiver stores all correct frames following a faulty one**
- **if sender is notified about an error**
  - it retransmits only the faulty frame
  - (i.e. not all the following ones, too)
- **if received properly**
  - receiver has a lot of frames in its buffer
  - and transfers frames in correct sequence from L2 to L3

Comments

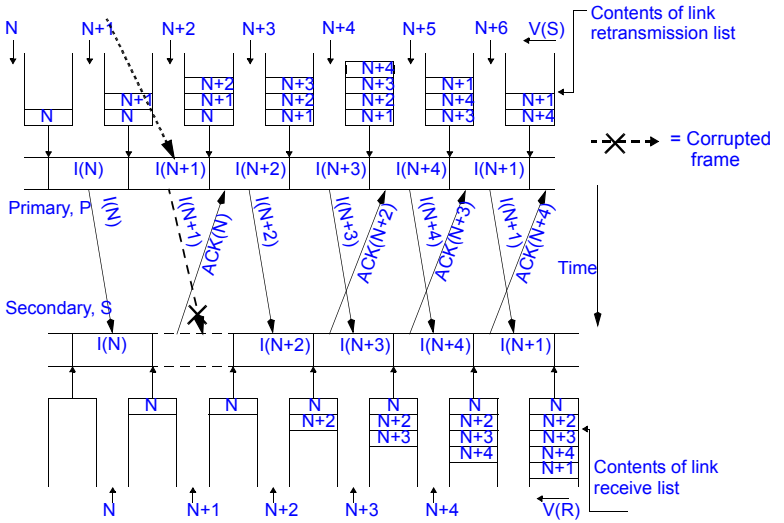
- **corresponds to window size > 1**
- **occurance of bursts at data link layer service interface**

Features

- **more complex**
- **buffering of "out of sequence" frames**
- **increased throughput**
- **no cumulative acknowledgements**

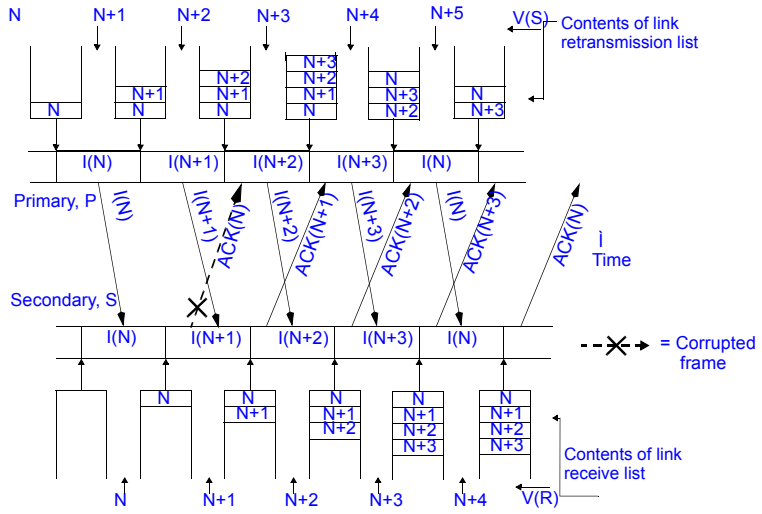
Sliding Window: Selective Repeat: Example

Faulty Data Frame

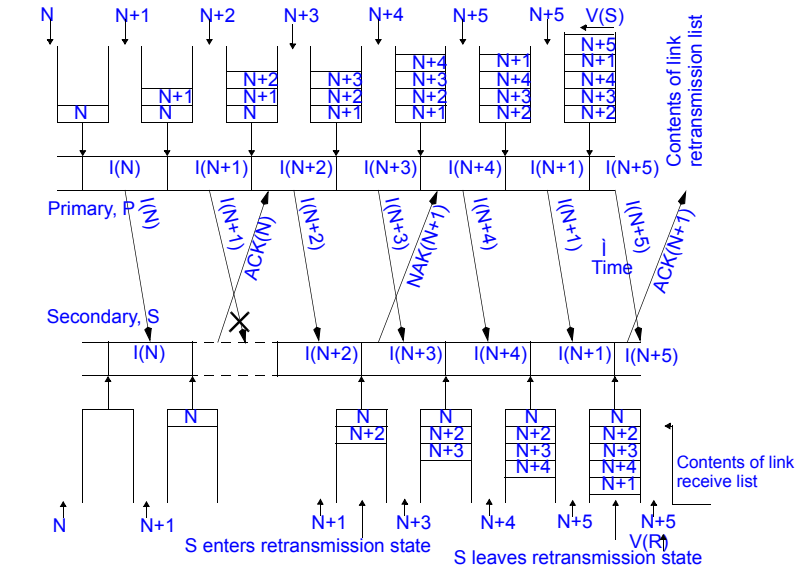


Sliding Window: Selective Repeat: Example (2)

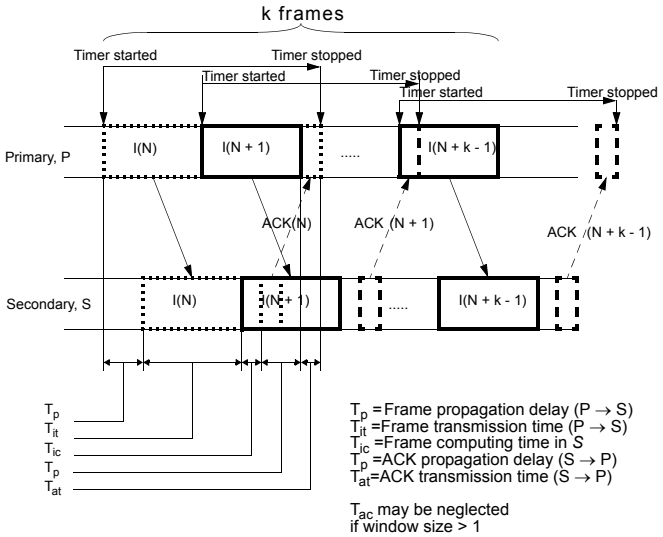
Faulty Acknowledge Frame



Active Error Control



Channel Utilization



Channel Utilization (2)

$$U = \begin{cases} \dfrac{kT_{it}}{T_{it} + 2T_p} = \dfrac{k}{1 + 2\dfrac{T_p}{T_{it}}} & \text{if } \left(k < 1 + 2\dfrac{T_p}{T_{it}}\right) \\ 1 & \text{otherwise} \end{cases}$$

Comment

- **k specifies**
  - how many frames are transmitted simultaneously (sequentially) on L1 channel
  - i.e. k=window size

5.5 Protocol Comparison

Stop-and-Wait

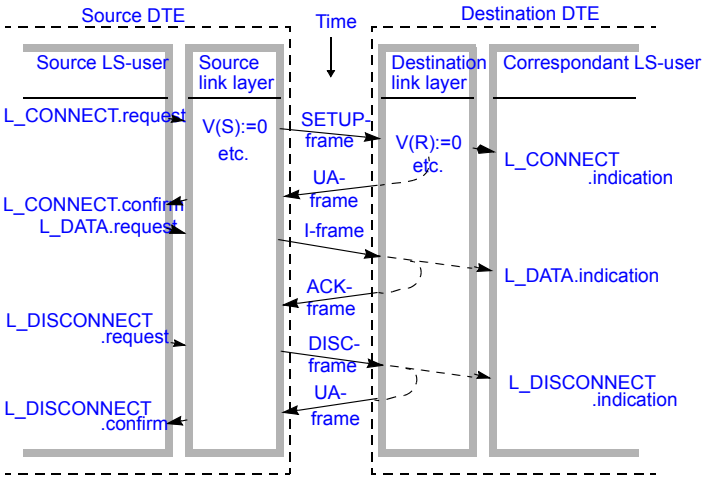
- **+ little demand for buffering**
- **+ well-suited for less complex end devices**
- **- poor channel utilization**
- **- low throughput**

Sliding Window

- **+ good channel utilization**
- **+ good throughput**
- **+ consideration of the current system state by adjusting the window size**
- **- increased buffer demand**
- **- more complex protocol**

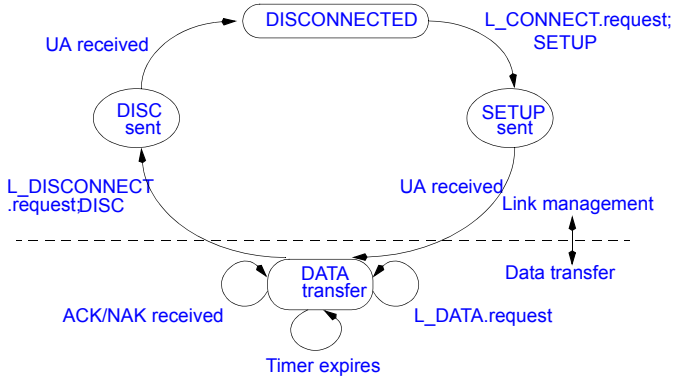
6. Connection Management

Presentation of the transmitted frame sequences

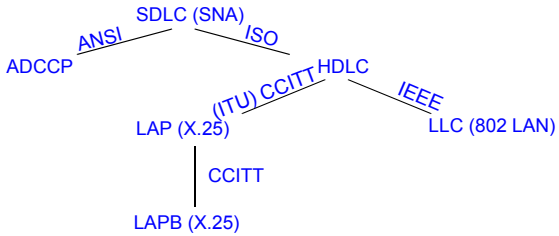


Connection Management (2)

State presentation



7. HDLC: Family



SDLC: Synchronous Data Link Control (derived from IBM System Network Architecture SNA)

ADCCP: Advanced Data Communication Control Procedure

HDLC: High-Level Data Link Control

LAP: Link Access Procedure

LAPB: Link Access Procedure, Balanced, example: L2 from OSI L3: X.25

LLC: Logical Link Control

others: Kermit, XMODEM (for modems between PCs)

⇒ **"The nice thing about standards is that you have so many to choose from" [Tanenbaum]**

HDLC: Principle

Detailed description: standards, [black]

- **a.o.: ISO 3309 HDLC Procedures - Frame Structure**

Protocol: bit-oriented, full duplex

Data transparency: bit stuffing (5* "1" always followed by "0")

Frame format (L2-PDU):

Bits	8	8	8	#(n*8)	16	8
01111110	Address	Control	Data		Checksum	01111110

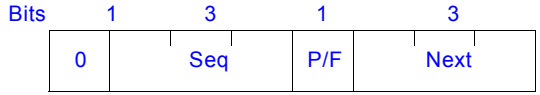
- **address:** **addressing of stations:**
  - important if several nodes on level L1
  - if point-to-point - L1: sometimes used to differentiate between command and response
- **control:** **sequence numbers, ACKs, ... (see below)**
- **data:** **any user data of any length**
- **checksum:** **Frame Check Sequence FCS variation of CRC**

3 Types of frames: (differ in control field)

Information	for data transmission
Supervisory	for control management during data transfer
Unnumbered	for connection management

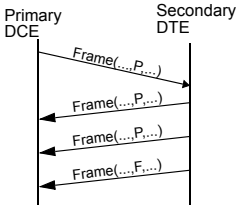
HDLC: Information Frame

Control field:



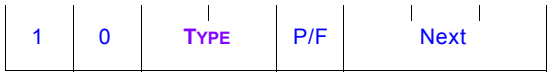
Sliding window, 3-bit sequence number

- **Seq: sequence number of the frame**
- **Next: sequence number of the frame to be expected next (instead of the last correctly received frame as described earlier)**
  - in form of "Piggybacking"
- **P/F: Poll/Final, mainly used for polling**
  - DCE (computer) requests DTEs to transmit data
  - DTE transmits (P-bit setting, last frame with F-bit)



HDLC: Supervisory Frame

Control field:

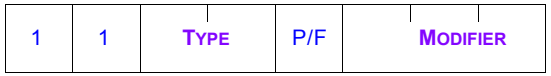


TYPES:

- **Type 00: RECEIVE READY (actually ACK)**
  - explicit acknowledgement
  - Next: next expected frame
- **Type 01: REJECT (actually NAK)**
  - negative acknowledgemet if transmission error
  - Next: first frame to be retransmitted
  - "Go-back-N" method (retransmit all up to faulty frame)
- **Type 10: RECEIVE NOT READY (actually ACK & STOP)**
  - reports that receiver has a temporary problem
  - cease acknowledgement and transmission
  - Next: frame to be expected
  - re-activate transmisison: RECEIVE READY, REJECT,...
- **Type 11: SELECTIVE REJECT**
  - retransmission of a frame (but not all previous frames)
  - Next: frame to be retransmitted
  - "Selective Repeat" method

HDLC: Unnumbered Frame

Control field:



Application:

- **control data**
- **unsecured connectionless service**

TYPES (type, modifier) a.o.

- **DISC (Disconnect):**
  - reports inavailability (e.g. during preventive maintenance)
- **SNRM (Set Normal Response Mode):**
  - reports availability (Set SeqNo. = 0)
    (Seq.No's. are reset to 0)
    unbalanced (primary/secondary master/slave) from "old" times
- **SABM (Set Asynchronous Balanced Mode):**
  - reports availability (Set SeqNo. = 0)
    balanced (peer-to-peer) from "more recent" times
- **FRMR (Frame Reject):**
  - frame drop if protocol infraction
    e.g. frame < 32 bit, ACK from frames with invalid SeqNo's
- **UA (Unnumbered Acknowledgement):**
  - safeguards against loss/error of unnumbered frames
  - ACK, acknowledgement for unnumbered frames

HDLC and OSI Service Elements

Connect

OSI SDU use of	**HDLC PDU Unnumbered Frame:**
• Connect Request	SABM (Set Asynchronous Balanced Mode)
• Connect Response	UA (Unnumbered Acknowledgement)

Data transfer phase

OSI SDU use of	**HDLC PDU Information Frame:**
• Data Request	Information Frame itself

Disconnect

OSI SDU use of	**HDLC PDU Unnumbered Frame:**
• Disconnect Request	DISC (Disconnect)

Exception treatment

OSI SDU use of	**HDLC PDU Unnumbered Frame:**
• Abort Indication	FRMR (Frame Reject)

other HDLC PDUs

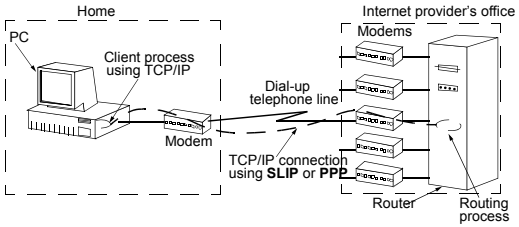
- **are exchanged within L2**
- **do not have a direct effect onto L3- or L2-SDUs respectively**

Differences HDLC, SDLC, LAPB

Supervisory Frames
- **HDLC and ADCCP**
  - know SELECTIVE REJECT
  - are more effective on L1 if many bit errors occur
- **SDLC and LAPB**
  - SELECTIVE REJECT not defined
  - i.e. only Go-Back-N

Unnumbered Frames
- **HDLC and LAPB**
  - know SABM (SET ASYNCHRONOUS BALANCED MODE)
  - know SABME and SNRME (... extended) with 7 bit SeqNo. (instead of 3 bit)
- **other**
  - actually do not have this frame specified

8. Internet L2 Protocols



Internet Connections
- **end devices to "network"**
  - approx. 10 m - 10 km: LAN & MAN: many connected to each other
  - more than approx. 10-100 km: WAN
    - point-to-point (considered here)
    - an Internet provider (AOL, T-Online, Univ.,...)
    - usually via phone line and modem
    - usually TCP/IP over **SLIP** or **PPP**
- **connection between network nodes**
  - point-to-point: (reviewed herein)
  - usually over dedicated line with router or bridge
  - often IP over **SLIP** or **PPP**

8.1 Internet: Serial Line IP (SLIP)

History
- **1984: Rick Adams connects Sun computers via modem to the Internet**
- **description in RFC 1055**

Protocol (very simple)
- **data (payload)**
  - L3 packets (here only IP packets)
- **framing:**
  - add flag byte (0xC0) at the end of the packet
  - character stuffing: if 0xC0 part of the data, replace this by 0xDB, 0xDC, etc.
  - note: some implementations insert these also as headers

Internet: Serial Line IP (SLIP) (2)

Properties
- **no error detection or error correction**
- **only IP is supported**
  - IP addresses of communicating entities have to be known in advance
  - (i.e., cannot be allocated dynamically)
  - i.e., each user would have to have his own IP address on the host -> too many addresses
- **no authentification (problem for switched line, not dedicated line)**
- **many different implementations exist because no Internet standard**
- **widely used**

Optimizations:
- **RFC 1144**
- **properties:**
  - successive packets often have the same header
- **use:**
  - header compression, if successive packets are the same

8.2 Internet: Point-To-Point Protocol (PPP)

History

- **IETF initiated group to**
  - replace SLIP (not a standard) by an Internet standard
  - improve the data link protocol
- **application: login connections and dedicated lines**
- **RFC 1661 (others: RFC 1662, RFC 1663)**
- **will replace SLIP**

Protocol: character-oriented (SLIP bit-oriented) with

- **FRAMING: frame identification and error treatment**
- **Link Control Protocol LCP (PHASE 1)**
  - establish, test, release L2 connection
  - authentification: L2 entities can determine each other's identities
  - negotiate options with L2 partner entity (e.g. coordinate payload size, select NCP protocol)
- **Network Control Protocol NCP (PHASE 2)**
  - one NCP per each L3 protocol
  - NCP for IP: selects for example IP address
- **actual data transfer (PHASE 3)**

Internet: Point-To-Point Protocol (PPP) (2)

Frame Format (similar to or derived from HDLC):

Bits (of SDLC to compare):
8 8 8 no ≥0 16 8
Bytes PPP here:
1 1 1 1 or 2 ≥0 2 or 4 1

Flag	Addr.	Ctrl	Protocol	Payload	Checksum	Flag
01111110	11111111	00000011				01111110

- **(HDLC) flag:** identifying characters for L2 frame
- **address:** always addressing of all stations
- **control:** default: unnumbered frame (see above)
  without ACK, without error treatment
  otherwise: numbered mode

(Comment: Address & Control fields can be left out depending on the setup)

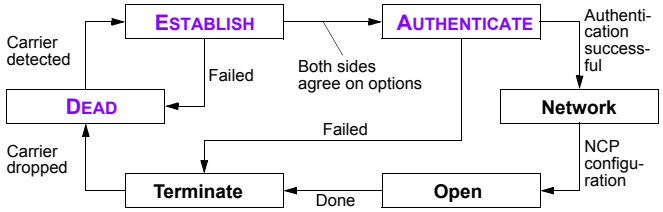
- **protocol:** designates the protocol, usually 2 byte
  - L2: LCP, NCP
  - L3: IP, OSI CLNP, Appletalk, ..
- **payload/data:** any user data
  - length: negotiated, otherwise max. 1500 byte
- **checksum:** CRC variation, usually 2 byte
- **(HDLC) flag:** bound of the L2 frame

Internet: Point-To-Point Protocol (PPP) (3)

Features

- **error treatment**
- **supports several L3 protocols/services**
- **IP addresses are determined dynamically**
- **authentification**

Internet: Point-To-Point Protocol (PPP) Phase Diagram, States (4)



Dead

- **L1 connection does not exist**

Establish

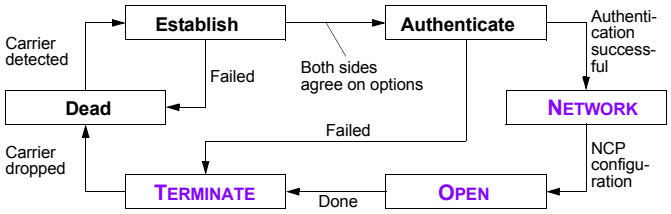
- **after L1 connect**
- **negotiation of LCP options**

Authenticate

- **authentification of both parties**

...

Internet: Point-To-Point Protocol (PPP) (5)



...

Network
- call of the desired NCP protocol
- configuration of the network layer

Open
- data transfer may begin
- e.g. transmission of IP packets in the payload field of PPP frames

Terminate
- disconnect

9. Perspective: L2 Communication Design Tasks

Service and protocol design include questions about (among others):

Specification method
- type (Petri-Netz, SDL, ESTELLE, LOTOS,..)
- utilization range (up to generating programs)
- deadlocks may be recognized (depending on method)

Implementation
- HW vs. SW
- SW environment (C, C++, class library,...)
- buffer management (logical copying)
- process segmentation

Service selection
- connection-oriented vs. confirmed connectionless vs. connectionless

Configuration/parameter selection
- sliding window size
  (among others depends on L1 error properties, L1 transmission time)
- error correction procedure