Chapter 3: The Data Link Layer

3.1 Data Link Layer Design Issues

The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

1. Providing a well-defined service interface to the network layer.

2. Dealing with transmission errors.

3. Regulating the flow of data so that slow receivers are not swamped by fast senders.

To accomplish these goals, the data link layer takes the packets it gets from the **network layer** and encapsulates them into **frames** for transmission. Each frame contains a frame header, a payload field for holding the packet, and a frame trailer, as illustrated in Figure 3.1. Frame management forms the heart of what the data link layer does. In the following sections we will examine all the abovementioned issues in detail.

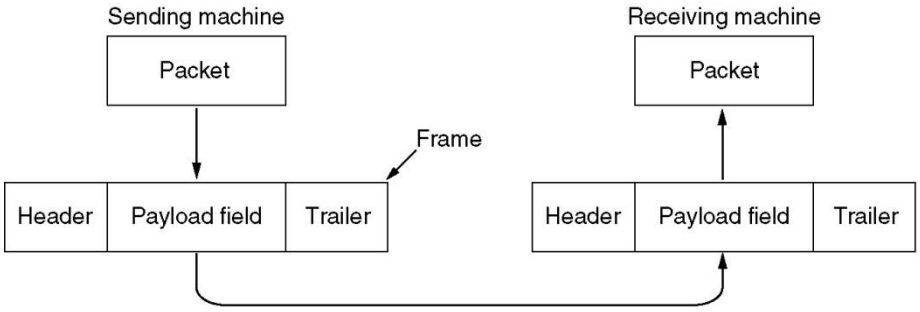


Figure 3.1: Relationship between packets and frames.

3.1.1 Services Provided to the Network Layer

Data link: describes how a shared communication channel can be accessed, and how a data frame can be reliably transmitted.

Data link layer is responsible for transmitting data from source network layer to destination network layer

Source network layer passes a number of bits to data link layer, who then packs them into frames and relies on physical layer to do actual communication. Receiving data link layer unpacks frames and passes bits to its network layer as shown in Figure 3.2 and Figure 3.3.

The basic services commonly provided are:

1. **Unacknowledged connectionless**: no attempt to recover wrong or lost frame in layer 2, having least overhead, appropriate when error rate is very low (LANs) so recovery is left to higher layers
2. **Acknowledged connectionless:** sender knows a frame has arrived safely or not, useful over unreliable channels (wireless systems)
3. **Acknowledged connection-oriented:** highly reliable but overhead is very high, usually for WANs.

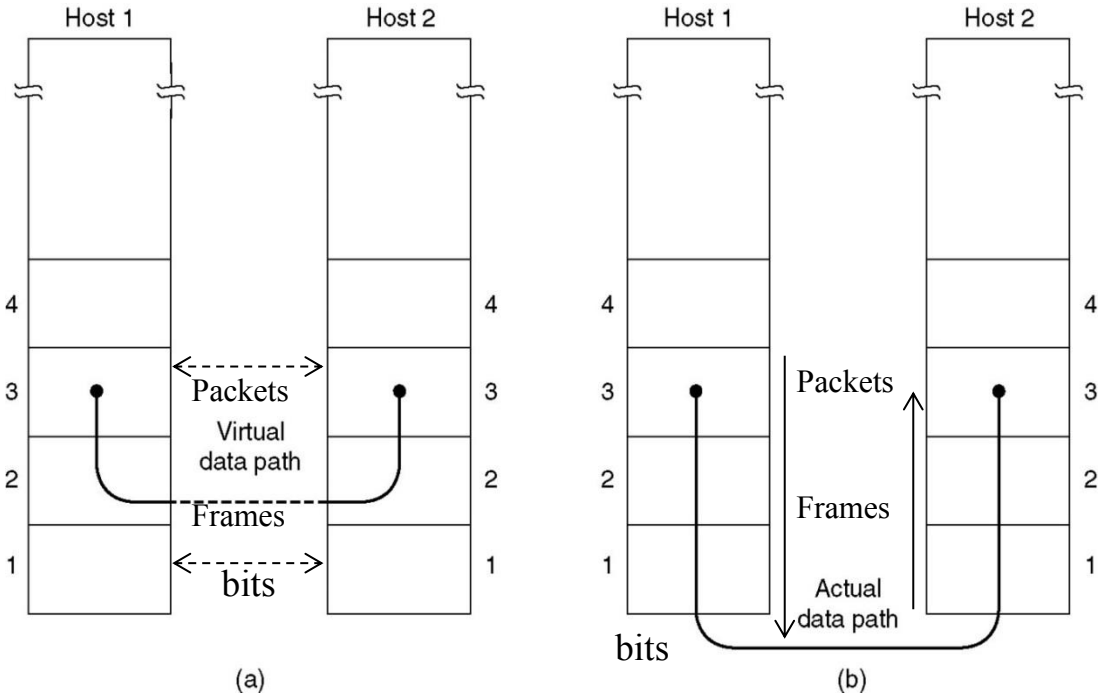


Figure 3.2: (a) Virtual communication. (b) Actual communication.

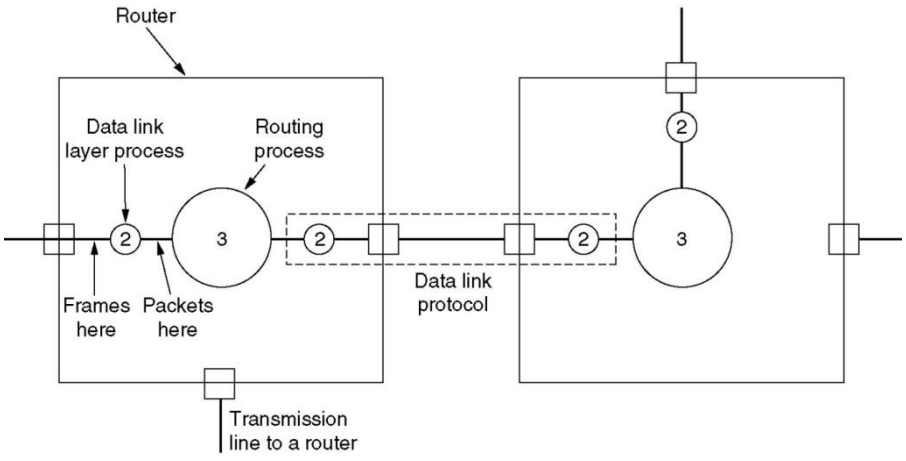


Figure 3.3: Placement of the data link protocol

3.1.2 Framing

Data link layer is responsible for making physical link reliable and, to do so, it breaks up network layer data stream into small blocks, a process called **segmentation**, and adds header and frame flag to each block to form a frame, a process called **encapsulation.**

Figure 3.4: frame structure.

- Header generally contains three parts or fields
  1. Address: address of sender and/or receiver
  2. Error detecting code: a checksum of the frame for error detection
  3. Control: additional information to implement protocol functions
- The receiving data link layer must know the start and end of a frame according to the frame flag. A good design must make it easy for a receiver to find the start of new frames while using little of the channel bandwidth. We will look at four methods:
  1. **Byte count**: The first framing method uses a field in the header to specify the number of bytes in the frame

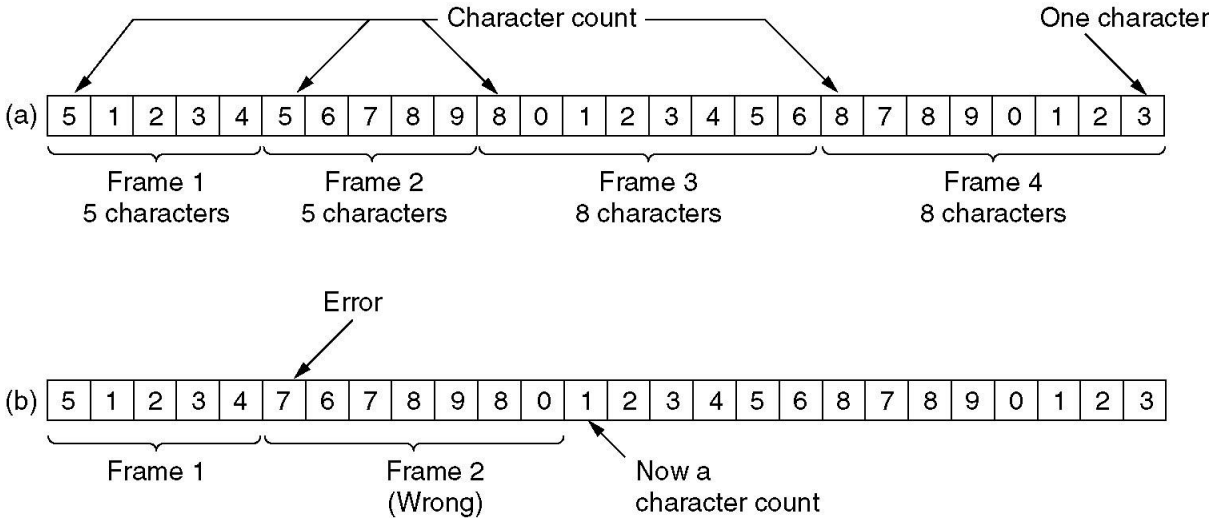


Figure 3.5: A character stream. (a) Without errors. (b) With one error.

2. **Flag bytes with byte stuffing: Each frame start and end with special bytes**. Often the same byte, called a flag byte, is used as both the starting and ending delimiter. This byte is shown in Figure 3.6 (a) as FLAG.

   It may happen that the flag byte occurs in the data. One way to solve this problem is to have the sender's data link layer insert a special escape byte (ESC) just before each ''accidental'' flag byte in the data. This technique is called **byte stuffing**, Figure 3.6 (b).

3. **Flag bits with bit stuffing**. Frame flag must be a special bit pattern that never appears any other place inside a frame. Bit stuffing: Use 01111110 as frame flag and this bit pattern must be made special. So sender adds a 0 bit whenever it encounters five consecutive 1 bits in data, and receiver deletes the 0 bit that follows five consecutive 1 bits in the received data.

4. **Physical layer coding violations.** Recall line coding defines how 0 and 1 bits are transmitted in voltage pulses, and deliberately violating the rule can be used to signify something special

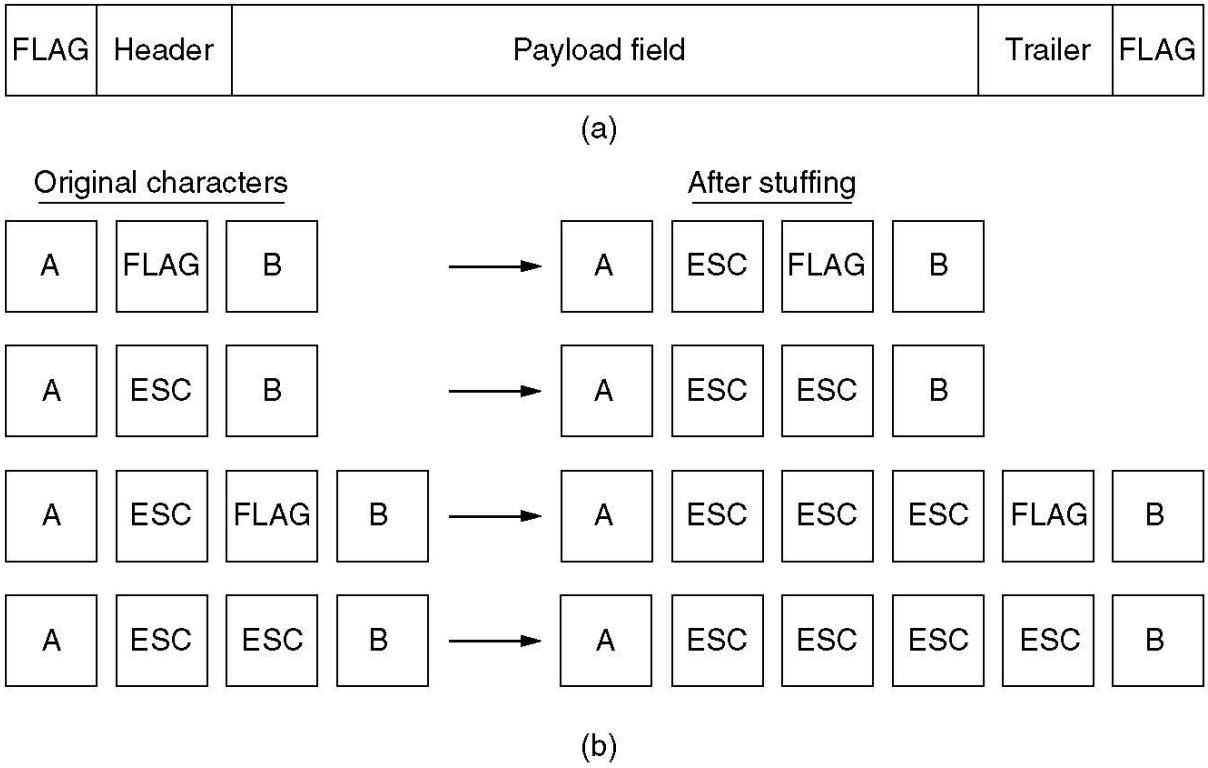


Figure 3.6: (a) A frame delimited by flag bytes.

(b) Four examples of byte sequences before and after stuffing.

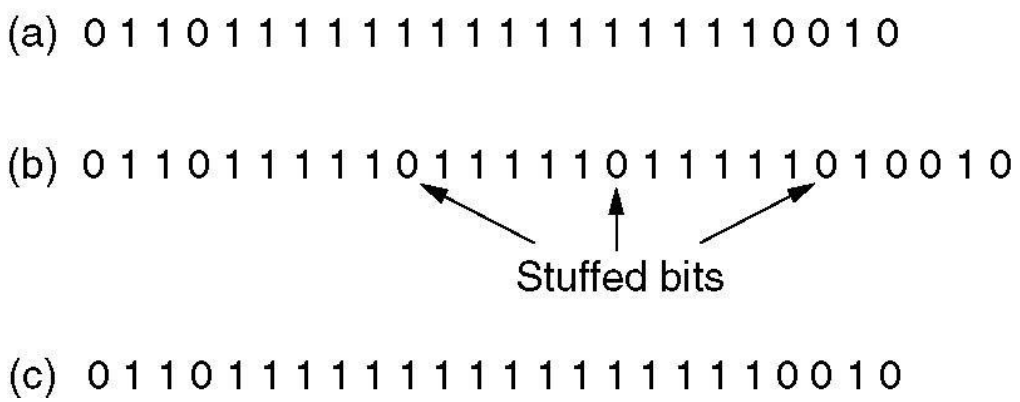


Figure 3.7: Bit stuffing
(a) The original data.
(b) The data as they appear on the line without the frame flag (01111110).
(c) The data as they are stored in receiver's memory after de-stuffing.

3.2 Error Detection and Correction

- At the receiver, data contained in frame is an arbitrary bit string

This means that the receiver is not allowed to know which bit strings correspond to "legal" data and which don't (in accordance with layered encapsulation principles)

- Therefore some additional bits must be added to the frame to allow errors to be detected(and possibly corrected)
- Note that it is not possible to detect ***all*** errors.
- Example:

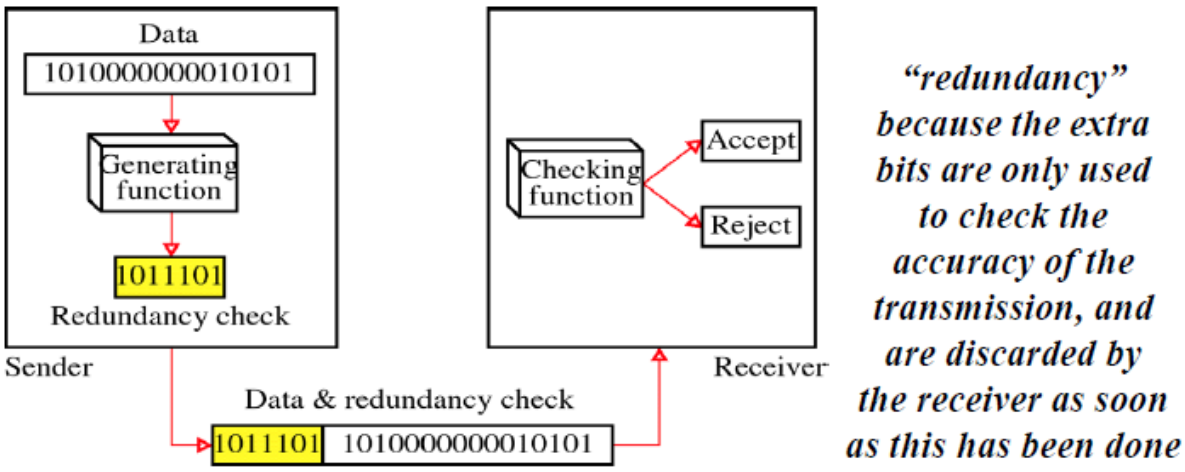


Figure 3.8: Data and redundancy check.

3.2.1 Error-Correcting Codes

- Error correction takes two forms:
- Forward Error Control (FEC)-each block transmitted contains extra information which can be used to detect the presence of errors *and* determine the position in the bit stream of the errors.
- Backward (Feedback) Error Control (BEC)-extra information is sufficient to only detect the presence of errors. If necessary, a retransmission control scheme is used to request that another copy of the erroneous information be sent.
- This is a more common method.

To either detect or correct errors at the receiver requires adding ***redundancy*** (extra bits) to each packet accepted from the network layer. For example, suppose that a packet contains m bits and each of the 2^m m bit sequences is a possible packet. Without adding any redundancy, it would be impossible to tell if an error occurred or not.

Still assuming that each packet contains m bits, suppose we encode that packet into a sequence of n bits where $n>m$. This encoding is often done by adding $n-m$ additional bits to the original packet. For each packet of ***m** bits*, the resulting sequence of n bits is called a ***codeword***; the set of all 2^m codewords along with the encoding rule is called a ***block code***. Note there are 2^n possible sequences of n bits, but only 2^m of them are codewords. Thus if the received frame contains a sequence which is not a codeword, we know an error has occurred.

The following codes are used for error correcting; in next sections we will discuss the hamming codes.

1. Hamming codes.
2. Binary convolutional codes.
3. Reed-Solomon codes.
4. Low-Density Parity Check codes.

Hamming Distance:

The number of bit positions in which two code words differ is called the **Hamming distance (*d*)**.

Example 1: W1=10001001, W2=10110001, then *d(W1,W2) = 3*

- For any error detection/correction scheme, we can define:
- **The minimum Hamming distance (or "minimum distance")** of the scheme as the ***smallest number*** of bit errors that changes one valid codeword into another

- m data bits, r check bits, n = m+r total bits all 2^m possible data strings are (usually) valid; but since the check bits are determined from the data, not all 2^n possible codewords are valid.
- if the way of computing the check bits is known, a list of all the valid codewords can be compiled and stored at the receiver, When a word W is received, the receiver finds the ***closest valid codeword to W(in Hamming distance)***and takes this codeword as the transmitted codeword
- If the minimum distance of an error-handling scheme is D, this scheme can **detect** any combination of ≤D-1 bit errors and **correct** any combination of strictly less than D/2 bit errors
- Alternatively: if you want to detect B bit errors, use a scheme with minimum distance at least B+1; if you want to correct B bit errors, use a scheme with minimum distance at least 2B+1.

Example 2: suppose only 4 valid codewords are:

00000 00000, 00000 11111, 11111 00000, and 11111 11111

- Minimum distance is D=5, so any combination of ≤4 bit errors can be detected and any combination of ≤2 bit errors can be corrected, but 3 bit errors can't be properly corrected.
- If 00000 00000 transmitted, W=00000 000**11** received: receiver knows that the received word is not a valid codeword ***(errors detected)***and computes Hamming distance between W and all valid codewords: d(W, C1)=2, d(W, C2)=3, d(W, C3)=7, and d(W, C4)=8 ⇒receiver takes C1=00000 00000 as transmitted codeword ***(errors corrected)***
- If 00000 00000 transmitted, X=00000 00**111** received: receiver knows received word is not a valid codeword ***(errors detected)***and computes Hamming distance between X and all valid codewords: d(X, C1)=3, d(X, C2)=2, d(X, C3)=8, and d(X, C4)=7 ⇒receiver takes C2=00000 11111 as transmitted codeword ***(in this case, error correction fails)***

Hamming Code

Hamming code is used for error correction and detection. In **Hamming codes** the bits of the codeword are numbered consecutively, starting with bit 1 at the left end, bit 2 to its immediate right, and so on. The bits that are powers of 2 (1, 2, 4, 8, 16, etc.) are parity bits. The rest (3, 5, 6, 7, 9, etc.) are filled up with the *m* data bits.

Example 3: Find the parity bits of the following codeword using hamming code:

01001101

Solution:

The parity bits positions are: P; 1, 2, 4 and 8 (shaded cells), 2^0, 2^1, 2^2, 2^3,…., while the other positions are filled with the original data.

P	**1**	**2**	3	**4**	5	6	7	**8**	9	10	11	12	Selected bits	Parity bits
	a	**b**	0	**c**	1	0	0	**d**	1	1	0	1		
P1	**a**		*		*		*		*		*		**a 0 1 0 1 0**	**0**
P2		**b**	*			*	*			*	*		**b 0 0 0 1 0**	**1**
P4				**c**	*	*	*					*	**c 1 0 0 0 1**	**0**
P8								**d**	*	*	*	*	**d 1 1 0 1**	**1**
	0	**1**	0	**0**	1	0	0	**1**	1	1	0	1	**Resulting code**	

* = Selected bits for each parity

Yellow or shaded (in black and white printing) columns are the parity bits locations.

Figure 3.9: Parity check bits computation process.

Other examples; Try to verify the following codewords..

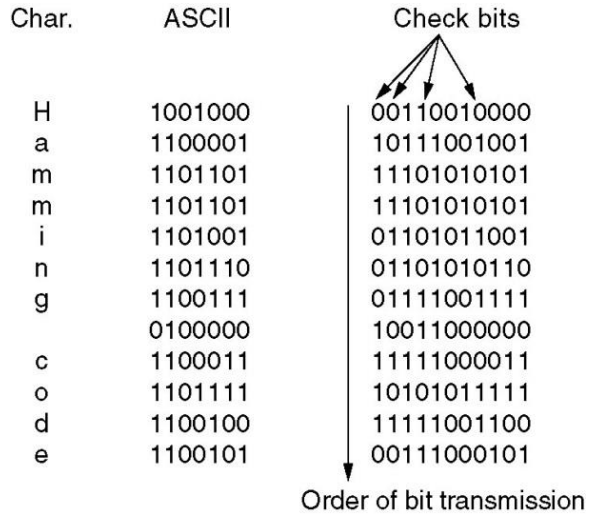


Figure 3.10: Hamming code examples.

Example 4: Check the following received codeword whether it is correct or not?

00110110000

Solution:

Re-compute the parity bits:

P	1	2	3	4	5	6	7	8	9	10	11	Computed parity bits	
	0	**0**	1	**1**	0	1	1	**0**	0	0	0	Selected bits	Parity bits
P1	**0**		1		0		1		0		0	**0** 1 0 1 0 0	0
P2		**0**	1			1	1			0	0	**0** 1 1 1 0 0	1
P4				**1**	0	1	1					**1** 0 1 1	1
P8								**0**	0	0	0	**0** 0 0 0	0

Bold face numbers are the received parity bits

Figure 3.11: Data and redundancy check.

Thus, the **error syndrome** is:

0 1 1 0 (0: correct, 1: incorrect)

2^0 2^1 2^2 2^3

=2+4=6

Thus, the **sixth bit in the received codeword is incorrect;**

Then the correct codeword should be:

00110**0**10000

3.2.2 Error-Detecting Codes

The following three error detecting codes will be discussed in this section:

1. Parity bit.
2. Checksums.
3. Cyclic Redundancy Checks (CRCs).

3.2.2.1 Parity bit:

Adding single bit to each codeword as shown in the following example:

Example 5: Compute the even parity bit and examine the error detection process of the following codeword; 10110101

- Five Ones in the data ⇒parity bit is **1**
- The transmitted codeword = 101101011
- If the receiver gets 101101011, parity check ok ⇒accept (OK)
- If the receiver gets 10110**0**011, parity check fails ⇒reject (OK), ask for frame to be re-transmitted
- If the receiver gets 1011**10**011, parity check ok ⇒accept (NOT OK: even number of errors undetected)
- If the receiver gets **0**01100011, parity check ok ⇒accept (NOT OK: even number of errors undetected)

3.2.2.2 Checksums

Form data into a 2-dimensional array; add single parity check bits to each row and each column; transmit row-by-row

Example 6: Examine the error detection processes using block check sum for the following data: 1110001 1000111 0011001

Solution:

- Form 3×7 array and add row and column parity bits:

1 1 1 0 0 0 1 *0*
1 0 0 0 1 1 1 *0* data bits
0 0 1 1 0 0 1 *1* *parity bits*
0 1 0 1 1 1 1 1

- The transmitted data is: 11100010 10001110 00110011 01011111
- The receiver knows to form received bit string into 4×8 array, then check the row and column parity bit
- can **detect** any odd number of bit errors in a row or column, and can detect an even number of bit errors if they're in a single row (using the column parity checks) or in a single column (using the row parity checks); and can **correct** any single bit error

To examine the error detection process for Example 6:

- suppose bit in position (1,3) is received in error (in other words, 1 bit error)

	Received Data	Computed parity for each Row	Parity check
Row1	1 1 **0** 0 0 0 1 ***0***	1	**parity check fails**
Row2	1 0 0 0 1 1 1 ***0***	0	parity check ok
Row3	0 0 1 1 0 0 1 ***1***	1	parity check ok
Row4	***0 1 0 1 1 1 1 1***	1	parity check ok
Parity for each Column	0 1 1 1 1 1 1 1		

column 1 parity check ok	column 5 parity check ok
column 2 parity check ok	column 6 parity check ok
column 3 parity check fails	column 7 parity check ok
column 4 parity check ok	column 8 parity check ok

Figure 3.12: Data and redundancy check.

Therefore the receiver can detect that the bit in position (1,3) was in error, so this bit can be corrected; i.e. the correct first codeword should be **1110001 (Row1).** Other codewords are all correct.

3.2.2.3 Checksums Cyclic Redundancy Check (CRC)

- Parity bit based methods do not, in general, provide adequate protection against error bursts.
- The most common alternative is based on the use of Polynomial Codes, which involves the computation of Cyclic Redundancy Check (CRC).
- Let M be the frame contents which is k bits long
- Let the CRC be n bits long
- Let G, the Generator Polynomial, be n+1bits long
- Divide M with n zeros appended by G and the remainder is the CRC, n bits long

Example 7: A series of 8-bit message blocks (11100110), to be transmitted across a data link using a CRC for error-detection purposes. A generator polynomial of (11001) is to be used. Illustrate the following:-

a- The FCS generation process, b- The FCS checking process.

Solution:

a- The FCS generation process.

- First, four Zeros are appended to the message, which are equivalent to multiplying the message by 2^4, since the FCS length will be four bits.
- Second, divided (modulo 2) by the generator polynomial, the modulo 2 division operation is equivalent to performing the EX-OR operation bit by bit in parallel.
- The transmitted message is generated by adding the resulting 4-bit remainder to the tail of the original message. The quotient is not used.

```
divisor      10110110   = Quotient(ignored)
  11001   | 11100110 0000        ( message )
            11001
             01011
             00000
              10111
              11001
               11100
               11001
                01010
                00000
                 10100
                 11001
                  11010
                  11001
                   00110
                   00000

                    0110 = Remainder (FCS)
```

Thus, the transmitted word should be:

11100110**0110**

b- The FCS checking process.

At the receiver, the complete received bit sequence is divided by the same generator polynomial. *Two cases are considered.*

In the first, an error burst of four bits at the tail of the transmitted bit sequence is assumed, and so the resulting remainder is non-zero, indicating a transmission error has occurred. In the second, no errors are assumed to be present, so the remainder obtained should be zero.

```
               10110110
11001    | 111001101111        error burst
           11001
            01011
            00000
             10111
             11001
              11100
              11001
               01011
               00000
                10111
                11001
                 11101
                 11001
                  01001
                  00000
                   1001        Remainder ≠ 0  error detected
```

Example 8*:* Repeat the reception process if no error burst happened.

Example 9:

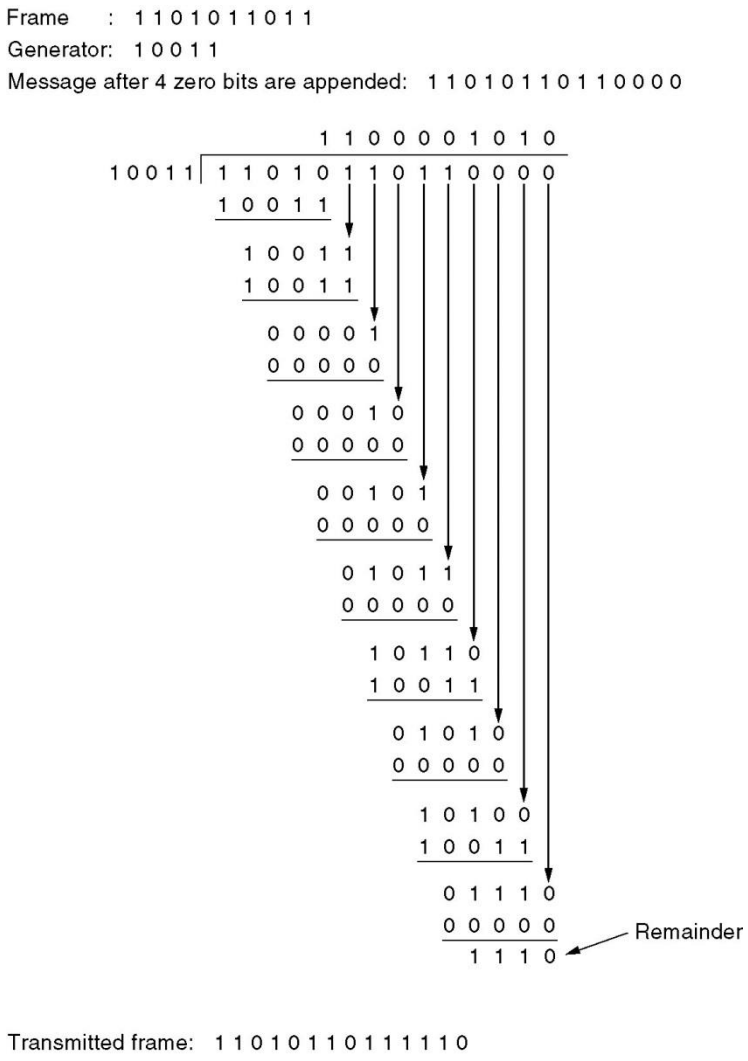


Figure 3.13: Calculation of the polynomial code checksum.

3.3 Flow and Error Control

3.3.2 Error Control

•allows the Receiver to tell the Sender about frames damaged or lost during transmission

•coordinates the *re-transmission* of those frames by the Sender

•note: since flow control provides the Receiver's acknowledgement (ACK) of correctly-received frames, error and flow control are closely linked

•*basic idea:* ACK every correctly-received frame and negatively acknowledge each incorrectly-received frame

•Sender keeps copies of un-ACKed Frames to re-transmit if required

•want: packets (inside frames) passed to Receiver's Network layer in order

3.3 Elementary Data Link Protocols

- An Unrestricted Simplex Protocol
- A Simplex Stop-and-Wait Protocol
- A Simplex Protocol for a Noisy Channel

3.3.1 Utopia/Unrestricted Simplex Protocol

Assumptions:

- Data transmission in one direction only (simplex)
- No errors take place on the physical channel
- The sender/receiver can generate/consume an infinite amount of data.
- Always ready for sending/receiving Stop-and-wait Flow Control

3.3.2 Simplex Stop-and-Wait Protocol

- Tackles the problem of preventing the sender from flooding the receiver with frames faster than the latter is able to process them (Drop assumption that receiver can process incoming data infinitely fast).
- stop-and-wait protocols where the sender sends one frame and then waits for acknowledgement.
- In this protocol, the contents of the acknowledgement frame are unimportant.
- Data transmission is one directional, but must have bidirectional line. Could have a half-duplex (one direction at a time) physical channel.

As in protocol 1, the sender starts out by fetching a packet from the network layer, using it to construct a frame, and sending it on its way. But now, unlike in protocol 1, the sender must wait until an acknowledgement frame arrives before looping back and fetching the next packet from the network layer. The sending data link layer need not even inspect the incoming frame as there is only one possibility. The incoming frame is always an acknowledgement.

3.3.3 Simplex Protocol for a Noisy Channel

What if the channel is noisy and we can lose frames (checksum incorrect). Simple approach, add a time out to the sender so it retransmits the frame after a certain period.

Scenario of what could happen:

- A transmits frame one
- B receives A1
- B generates ACK
- ACK is lost
- A times out, retransmits
- B gets duplicate copy (sending on to network layer)
- Use a sequence number. How many bits? 1-bit is sufficient because only concerned about two successive frames.
- Positive Acknowledgement with Retransmission (PAR) sender waits for positive acknowledgement before advancing to the next data item.

How long should the timer be? What if too long? (Inefficient)

What if too short? A problem because the ACK does not contain the sequence number of the frame which is being ACK'ed.

Scenario:

- A sends frame zero
- timeout of A
- resend frame A0
- B receives A0, ACKS
- B receives A0 again, ACKS again (does not accept)
- A gets A0 ACK, sends frame A1
- A1 gets lost
- A gets second A0 ACK, sends A2
- B gets A2 (rejects, not correct seq. number)
- will lose two packets before getting back on track (with A3,1)

3.4 Sliding Window Protocols

- **Sender can transmit several frames continuously before needing an ACK**
- if ACK received by Sender before continuous transmission is finished, Sender can continue transmitting
- an ACK can acknowledge the correct receipt of **multiple** frames at the Receiver
- Frames and ACKs must be numbered:
  - each Frame's number is 1 greater than the previous Frame
  - each ACK's number is the number of the *next Frame expected* by the Receiver

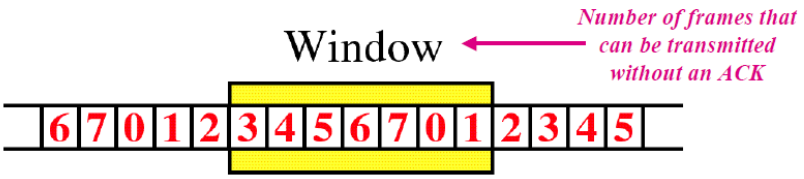


Figure 3.14: Sliding window

- Frames may be acknowledged by the Receiver at any time, and maybe transmitted by the Sender as long as the Window hasn't filled up
- Frames are numbered modulo-n, from 0 to n-1: 0, 1,…, n-1, 0, 1,…,n-1, 0, 1,…
- Size of the Window is n-1: 1 less than the number of different Frame numbers.
- Instead of sending Ack frame on its own, if there is an outgoing data frame in the next short interval, attach the Ack to it (using "Ack" field in header). Better use of bandwidth. This operation is called **Piggybacking.**

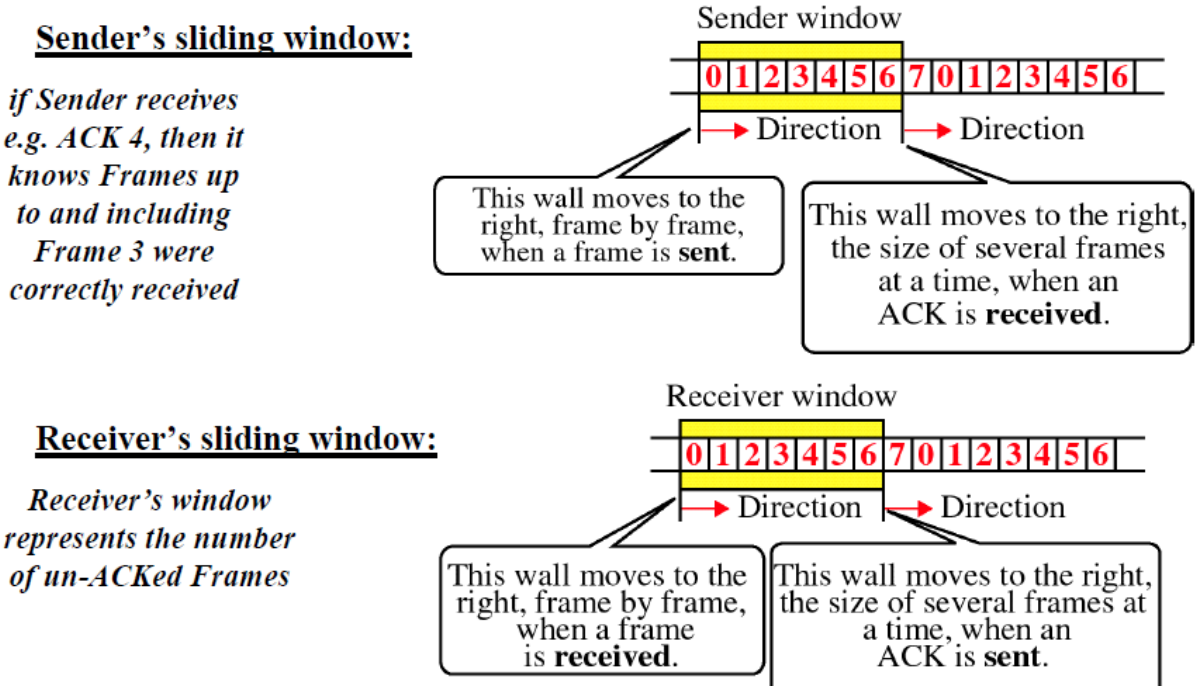


Figure 3.15: Sliding window

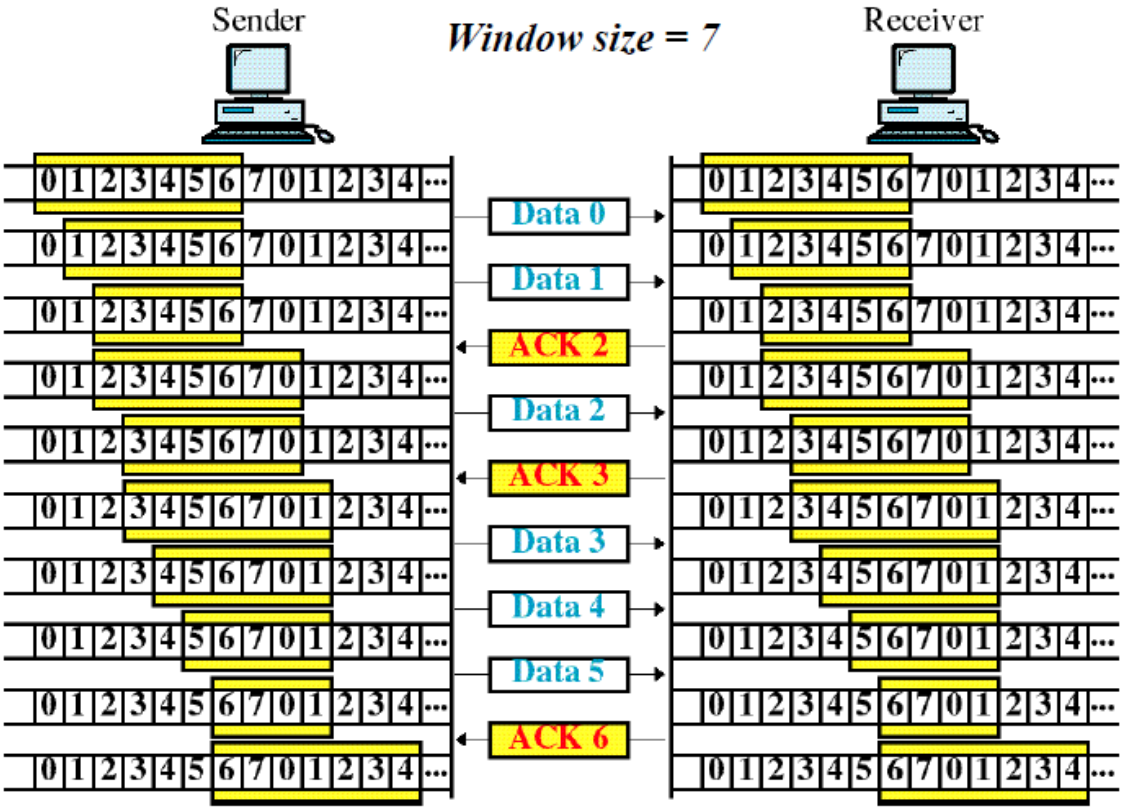


Figure 3.16: Sliding window protocol example.

3.4.1 Error Control: ARQ (Automatic Repeat Request)

- If error(s) detected in received Frame, return NAK (Not AcKnowledged) to Sender
  - NAK can be **explicit** or **implicit** (Sender's Timeout timer expires)
- Sender keeps a copy of each un-ACKed Frame to re-transmit if required
  - ACK received by Sender for Frame ⇒discard copy
  - NAK received by Sender for Frame ⇒decide how to re-transmit Frame
- Sender starts Timeout timer for each Frame when it is transmitted
  - Appropriate Timeout value = the expected delay for Sender to receive ACK for the Frame (in practice, set Timeout slightly larger than this…)
  - packet is not considered to be delivered successfully to the Receiver's Network layer until the Sender knows this (by getting ACK for it)
- 3 types of ARQ scheme:
  - Stop-and-wait ARQ–extension of Stop-and-wait flow control
  - Sliding window ARQ –extension of sliding window flow control:
    - Go-back-n ARQ–Receiver must get Frames in correct order
    - Selective repeat ARQ–correctly-received out-of-order Frames are stored at Receiver until they can be re-assembled into correct order

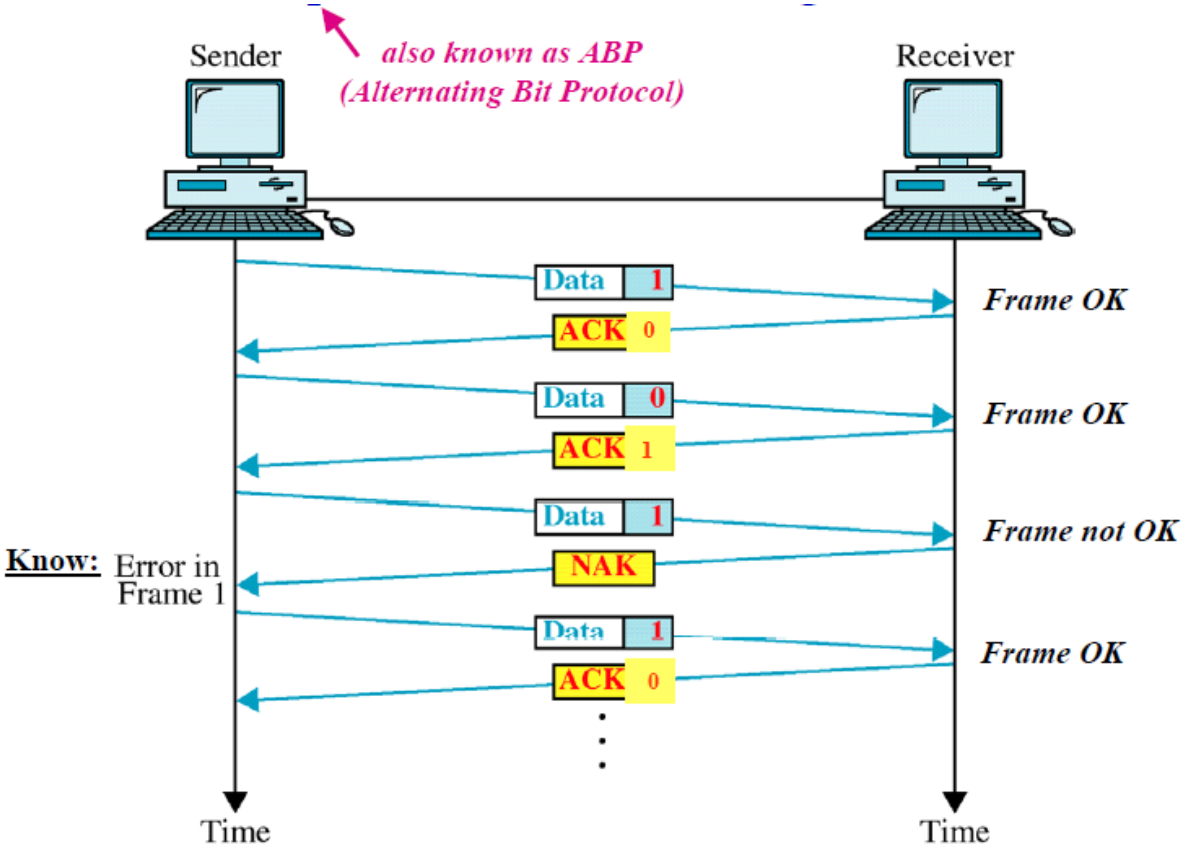


Figure 3.17: Error in Frame 1

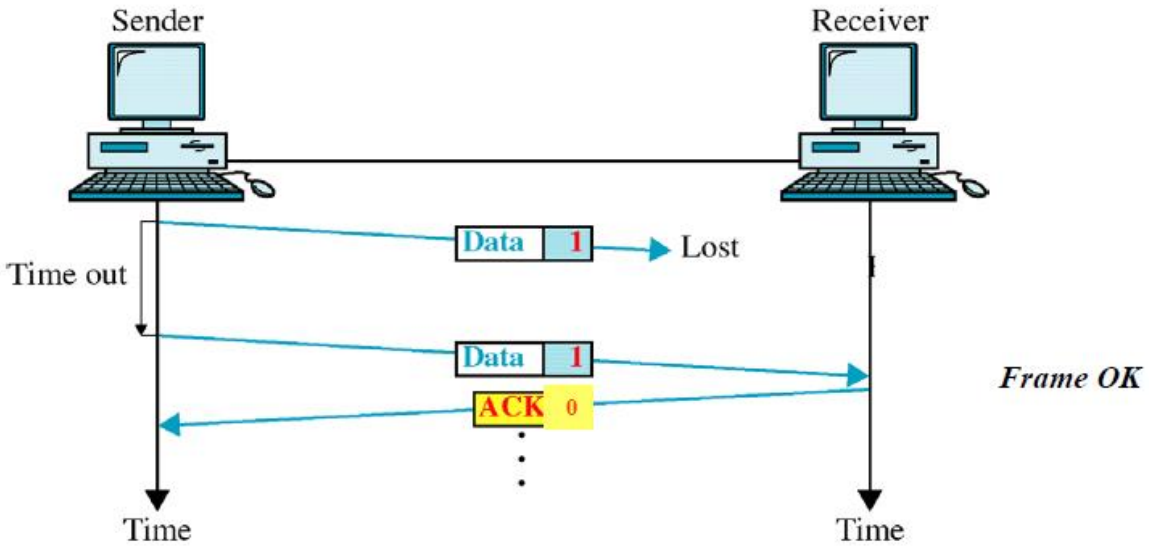
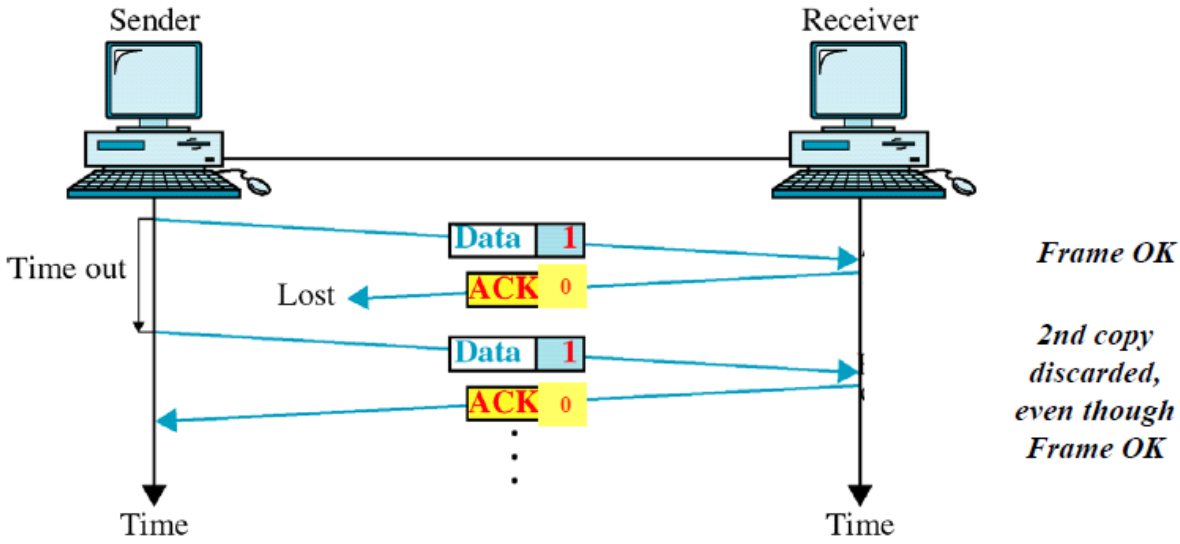


Figure 3.18: Data Lost



Q: why does Receiver send ACK 0 for copy of data Frame 1 ?

Figure 3.19: Acknowledgment Lost

To sum up there are four possible outcomes for packet transmission and acknowledgement:

- Frame received correctly at receiver and ACK received at sender
- Frame received incorrectly at receiver and NACK received at sender
- Frame lost before reaching receiver
- ACK/NACK lost before reaching receiver

3.4.2 Go-back-N ARQ, damaged data frame

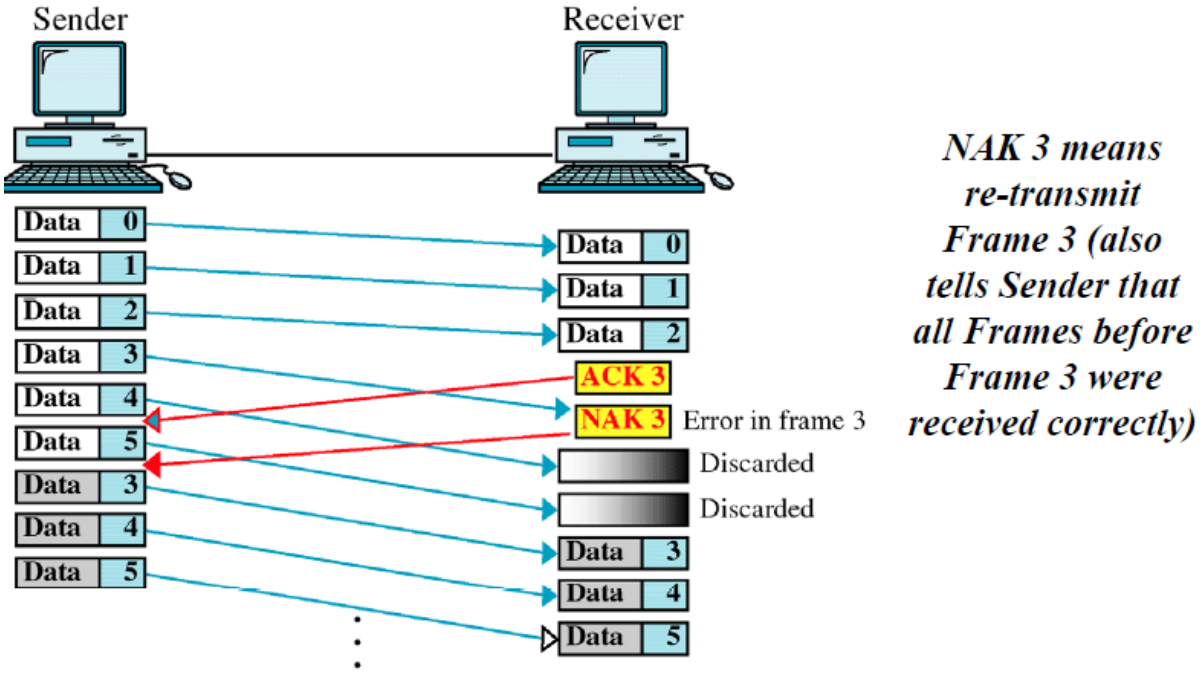


Figure 3.20: Error in Frame 3.

Receiver only accepts correctly-received Frames in the correct order (so Receiver doesn't have to buffer any Frames and re-order them…)

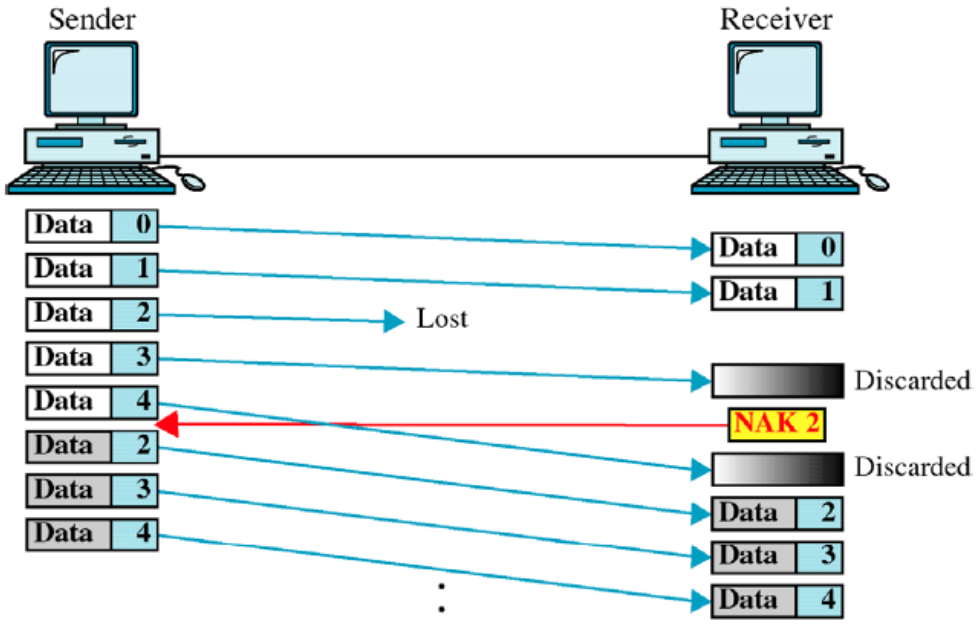


Figure 3.21: Lost Frame.

Frame 3 discarded even though it was correctly received BECAUSE Receiver was expecting Frame 2 (same for Frame 4)

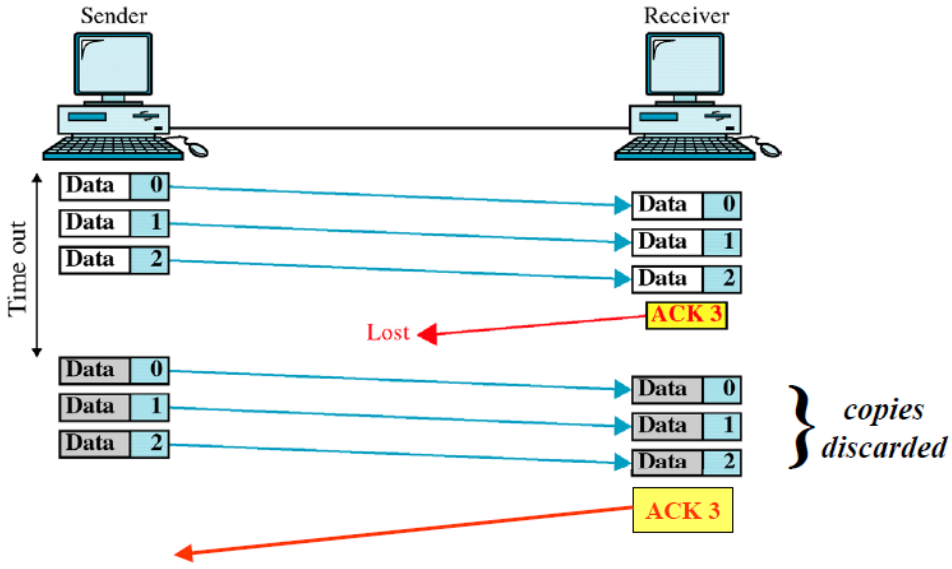


Figure 3.22: Lost Acknowledgement.

3.4.3 Selective repeat ARQ, damaged data frame

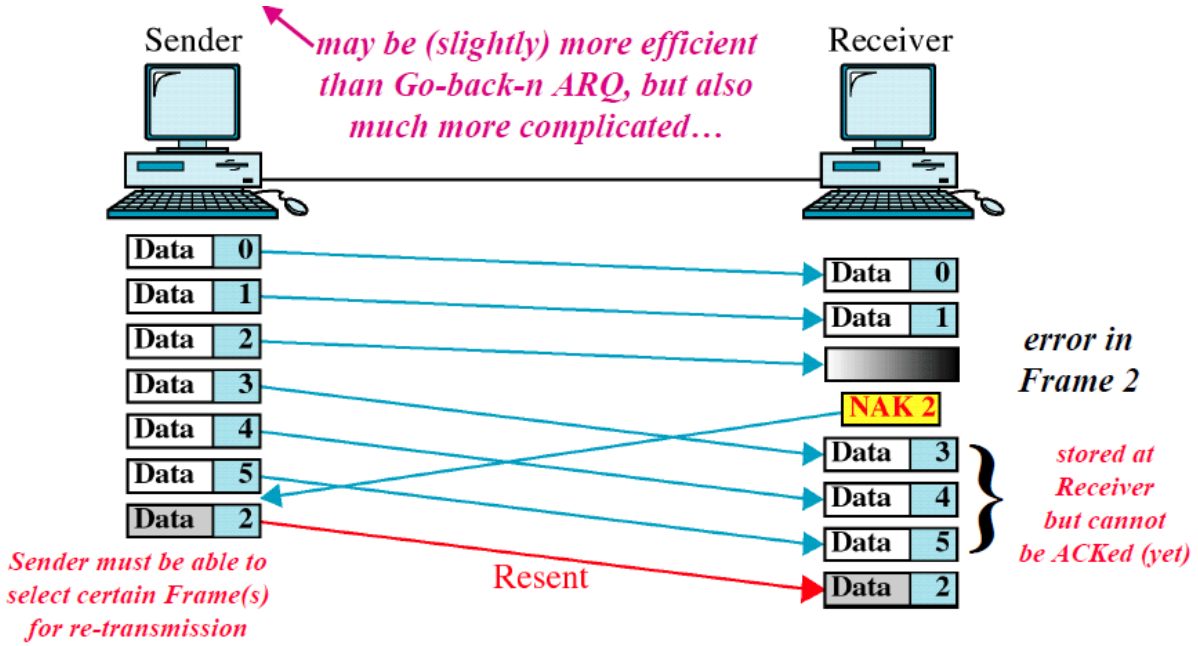


Figure 3.23: Error in Frame 2.

When frame 2 received correctly, Receiver can deliver the packets in Frames 2-5 to its Network layer and send ACK 6 back to Sender. Lost or damaged ACK/NAK handled similarly to Go-back-n ARQ.

3.5 Example Data Link Protocols

- HDLC – High-Level Data Link Control
- The Data Link Layer in the Internet

3.5.1 HDLC – High-Level Data Link Control

Features:

- bit-oriented, using bit stuffing with start and end flag 01111110.
- FCS: Arbitrarily long data field, followed by CRC checksum.
- polynomial: 16-bit CRC-CCITT
- Ack-ed service
- 3-bit frame numbers. i.e. 8 distinct numbers.
- piggybacked Acks
- Sliding window. Up to 7 un-ack-ed frames at any time.

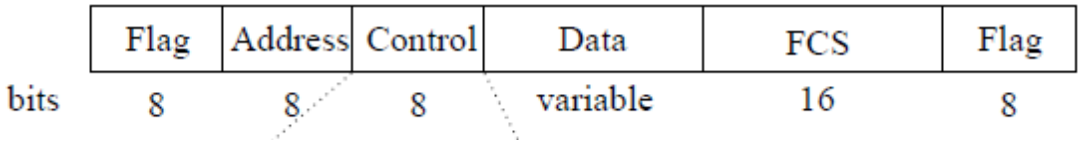


Figure 3.24: HDLC protocol format.

- Control: Three types of frames, I, S and U frames. The old protocol uses a sliding window with 3-bit sequence number and the maximum window size is N = 7. The control field is now extended to 16 bits with 7-bit sequence number

• I-frames: carry user data. Additionally, flow and error control (ACKs) may be piggybacked inside the control field of an I-frame.

• S-frames: provide flow/error control when piggybacking is not used.

• U-frames: are mainly for control purposes, i.e. establishing and terminating connections, unnumbered ACK, etc., but can also be used to carry data for unacknowledged connectionless services, and type is defined by M bits.

3.5.2 Data Link Layer in Internet

- Internet consists of various "networks", i.e. they may support different network layer protocols, and even links within may be very different. A data link layer protocol for Internet needs to be able to deal with these issues
- Internet connection, in practice, is built up on point-to-point links
  - Organisations' routers are connected to outside world's routers via point-to-point leased lines (router-router)
  - Home users connect to outside Internet routers via cable modems and dial-up telephone lines (host-router)
- Internet uses the point-to-point protocol (PPP): Its handles error detection, supports multiple protocols, allows IP addresses to be negotiated at connection time, permit authentication

Point-to-Point Protocol (PPP)

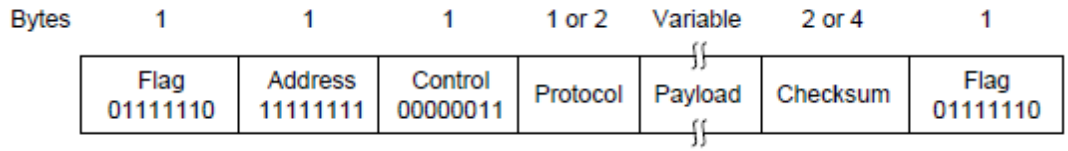


Figure 3.25: PPP format.

- PPP frame format is:
  - Frame flag: 01111110, and byte stuffing is used.
  - Address: 11111111, which means all stations can accept the frame; This avoids issue of assigning data link addresses.
  - Control: 00000011 is a default value, indicating unnumbered frame
  - Note there is no sequence number, so data link layer does not do flow/error control, but PPP has FSC and is able to detect errors
  - Protocol field: tells what kind of packet is in payload; Protocols starting with a 0 bit are network protocols such as IP, IPX, OSI, and others; starting with a 1 bit are used to negotiate other protocols.
- PPP provides link control protocol (LCP) for controlling line (setup, testing, negotiating options, shut down)