Chapter 4

The Data Link Layer

The data link layer provides a well defined service interface to the network layer, and also provides reliable, efficient communication between adjacent machines. For this, it has to provide the functions like error control, flow control etc. among others.

To achieve all these goals, it has to take the packets received from the network layer, and encapsulate them into frames for transmission. Each frame would contain a header, the payload (which carries the packet), and a trailer as shown in figure 4.1

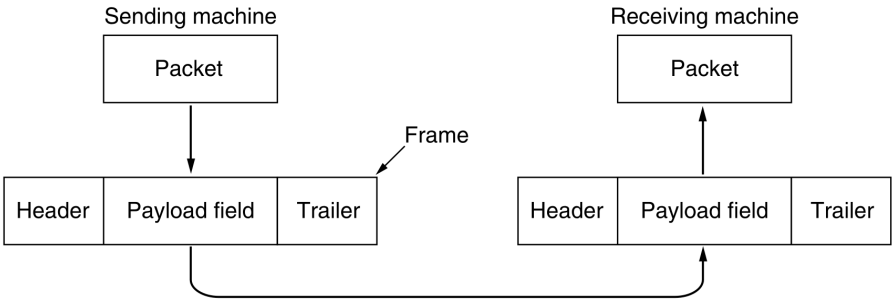

Figure 4.1: Packet to Frame Relationship (Courtesy: Computer Networks by Tanenbaum, 5th Ed.)

4.1 Services Provided to the Network Layer

(i) *Unacknowledged Connectionless Service:*

- Source machine sends independent frames to the destination machine, without destination machine's acknowledgement.
- No logical connection is established beforehand or released afterwards.
- Lost packets are not detected or recovered in this service class.
- This class of service is appropriate when transmission error rate is very low and for real-time traffic.

(ii) *Acknowledged Connectionless Service:*

- No logical connection is established beforehand or released afterwards.
- However, each frame sent is individually acknowledged (resend if not acknowledged within a specific time period).
- Useful for wireless systems.

(iii) *Acknowledged Connection-oriented Service:*

- The source and the destination establishes a logical connection before any data is transferred (i.e. the sender must make sure that the receiver is reachable, before beginning the actual data transfer[1]).
- Each frame sent is numbered and the data link layer guarantees that each frame sent is received.
- Also, each frame is received exactly once, and all frames are received in the right order.

4.2 Sublayers of the Data Link Layer

The original ISO/OSI reference model did not have any sublayers in the data link layer. However, it was added later on so that different network layer protocols such as IPv4, IPv6, IPX, DECnet etc. can work with different lower level protocols and media types, such as Ethernet, Token Ring, Bluetooth, Wi-Fi etc.

The two sublayers are:

(i) *Logical Link Control (LLC):* This is the upper sublayer of the data link layer, which defines the software processes that provide services to the network layer protocols. The primary functions of this sublayer are: *flow control, acknowledgement, error notification*, and handling logical addressing of the upper layer.

(ii) *Media Access Control (MAC):* This is the bottom sublayer in the data link layer, which defines the media access processes performed by the hardware (i.e. provides services to the physical layer). The primary functions of this sublayer are: *performing control of access to the media, hardware addressing, frame synchronization, detection of errors* etc.

4.3 Error Control

For Acknowledged Service:

- Positive acknowledgement (ACK) is sent by the destination if the frame is received without any error.
- Negative acknowledgement (NACK) means the frame needs to be retransmitted.
- In case where the frame is completely lost, or the acknowledgement is lost due to hardware problem:
  - The data link layer uses a timer.
  - When a frame is sent, the timer starts. If the timer expires the predefined time, some corrective action like retransmission needs to be done.
  - Sequence numbers must be used to avoid duplicate frames being received by the destination.

4.3.1 Error Control Strategies

Two basic strategies are used:

(i) *Error Correcting Codes:* Include enough redundant information along with each block, so that in case of an error, the destination can detect and also correct the error locally, without asking for a retransmission. Also known as *Forward Error Correction (FEC)*. This strategy is useful for wireless communication, which has high overall error rate.

(ii) *Error Detecting Codes:* Include only enough redundant information, so that the destination can figure out that some error has occurred during the transmission. Upon detection of an error, the destination has to ask the sender for a retransmission of the affected frame. Also known as *Automatic Repeat Request (ARQ)*. This strategy is useful for optical fiber communication (or in situations where the overall error rate is very low).

[1] Also called Handshaking

4.3.2 Types of Error

Primarily, the error control techniques are designed to handle two types of errors:

(i) *Single Bit Error:* Most of the errors that occur in a modern computer networks are of this type.

(ii) *Burst Error:* When more than one bit are affected by error, it is known as burst error.

4.3.3 Codeword

In error handling techniques, a codeword is constructed by appending the redundant or check bits (used to detect or correct errors) to the original data bits. So:

$$n = m + r$$

Where, n: Codeword, m: Data Bits, and r: Redundant or Check Bits.

4.3.4 Hamming Distance

In any two given strings with equal length, the hamming distance tells us how many symbols at the corresponding positions differ.

To find out how many bits in two given codewords differ, simply XOR them and count the number of 1s in the result, as shown below:

```
        1000 1001
        1011 0001
        ---------
(XOR): 0011 1000
```

In the above example, the hamming distance is 3, since in the result we get three 1s. When a codeword is transmitted, if the hamming distance between the transmitted codeword and the received codeword is *zero*, then no error has occurred during the transmission.

4.3.5 Few Error Detecting Methods

(a) *Parity Checking*

(b) *Checksum Error Detection*

(c) *Cyclic Redundancy Check (CRC)*

Parity Checking

- In this method, a single additional bit called the parity bit is appended to the data bits to generate the codeword.
- Both the sender and the receiver has to agree on a common parity type in advance.
- The parity type can either be *even parity* or *odd parity*.
- In case of even parity, the number of 1s in the data bits are calculated, and if it is odd, the parity bit is set to '1' to make number of 1s in the codeword even. If the total number of 1s in the data bits is even, the parity bit is set to '0' to keep the number of 1s in the codeword even.
- The same approach is taken in case of odd parity. The parity bit in this case is set such that the total number of 1s in the codeword always remains odd.
- The advantage of this method is that only one addition bit space is required per codeword. Moreover, the calculation can also be done very fast using simple hardware.
- However, it can detect only single bit errors, or odd-number bit errors. It fails to detect even-number bit errors.

- The variation of this method is called *Two-Dimensional Parity Check* method. In this method, multiple data bits are arranged in a tabular fashion, and parity bits are calculated for both the horizontal (called *longitudinal redundancy check (LRC)*) and vertical (called *vertical redundancy check (VRC)*) directions (i.e. for each rows and columns). Parity bit is also calculated for the resulting parity bit rows and columns, and placed at the diagonal position. The resulting codeword table is then transmitted. The method can detect single burst errors, as well as correct single bit errors.

Checksum Error Detection

- In this method, as each word is transmitted, it is added to the previously sent word, discarding the final carry.
- Finally, at the end of the transmission, the resulting sum (called the *checksum*) is also transmitted.
- The destination also does the same, and compares the locally calculated sum with the received checksum. If they match, the transmission is assumed to be error free.
- The source may also send the 2's complement of the checksum. In that case, the destination simply adds up all the received words and if the final result is not zero, it assumes that an error has occurred.
- Probability that a two-bit error will go undetected is a little less than $1/n$, where n is the number of bits in each of the words.

Cyclic Redundancy Check (CRC)

- CRC is a robust method which can detect a large number of errors.
- CRC is a type of polynomial code in which a bit string is represented in the form of polynomials with coefficients of *zeros* and *ones* only.
- For example, '110001' has 6 bits and thus represents a six-term polynomial with coefficients '1, 1, 0, 0, 0, 1' i.e. $x^5 + x^4 + x^0$ or $x^5 + x^4 + 1$
- Polynomial arithmetic uses a modulo-2 arithmetic i.e. addition and subtraction are done using XOR.
- There are no carries for addition, or borrows for subtraction.
- CRC is based on binary division.
- Long division is done same as in binary except that the subtraction is done modulo-2.
- A divisor is said *to go into* a dividend, if the dividend has as many bits as the divisor.
- The sender and the receiver must agree upon a generator polynomial $G(x)$ (i.e. the divisor) in advance.
- Both the high and low order bits of the generator $G(x)$ must be '1'.

Procedure to Obtain CRC:

- Say $G(x)$ contains n bits. Append $n-1$ number of 0s to the Data bits $M(x)$, meant for transmission.
- Divide the newly created data bits by the predetermined divisor (i.e. the generator $G(x)$) using modulo-2 method.
- Obtain the remainder — this is the CRC.
- Append the CRC to the Data bits $M(x)$. The resulting codeword $T(x)$ can now be transmitted. If the CRC contains less than $n-1$ bits, prepend with 0s to make its length equal to $n-1$.

- In the receiver's end, to verify if the codeword $T(x)$ is received without error, simply divide the codeword $T(x)$ by the generator $G(x)$ (without appending any zeros). If the final remainder is zero, there is no error.

Requirements of CRC:
A CRC will be valid, if and only if it satisfies the following requirements:

- It should have exactly one less bit than the divisor or the generator $G(x)$.
- Appending the CRC to the end of the data unit $M(x)$ should result in the bit sequence (or codeword) $T(x)$ which is exactly divisible by the divisor or the generator $G(x)$.

Errors Detected by CRC:

- All single-bit errors.
- Two isolated single-bit errors (if the divisor is not small).
- Odd number of errors (if the divisor is selected properly, i.e. make $x + 1$ a factor of the divisor).
- All burst errors, for a polynomial code with r check bits (or redundant bits), if the burst error length is $\leq r$.
- *Summary:* Those errors that happens to correspond to the polynomials containing the divisor as a factor will slip by — all other errors will be caught.

Some Standard Divisors:

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

CRC Calculation Example: For a working example of CRC calculation and verification, refer to figure 4.2 on page number 42.

4.3.6 Few Error Correcting Codes

(a) *Linear Block Codes*

(b) *Convolutional Coding*

(c) *Hamming Codes*

(d) *Cyclic Codes*

Hamming Codes

- Hamming codes can detect and correct *single-bit* error.
- In this method, to construct the codeword, *check bits* (or parity bits) are placed at positions 2^n, where $n = 0, 1, 2, 3, ...$
- Rest of the bit positions of the codeword are used for the data bits.
- The check / parity bits are decided in advance to be either *odd* or *even*.
- For the illustration, we shall consider the example of 7-bit hamming code, though longer hamming codes (e.g. 15-bit hamming code) uses the same procedure.
- In case of 7-bit hamming codes, the data and parity bits of the codeword are placed as shown in figure 4.3

■ Example: For the data bits M(x) = 110101, calaulate the CRC and construct the codeword T(x), if the generator (or divisor) G(x) = 101. Also show how the receiver will verify the received codeword for error.

■ Solution:

● Calculation of CRC and Codeword at the sender's end:

● Step 1: Since G(x) has three bits, append two (i.e. n-1) bits to M(x) for division.

```
101 | 1 1 0 1 0 1 0 0
```

● Step 2: Start the division. Use XOR for the subtraction purpose.

```
          1
101 | 1 1 0 1 0 1 0 0
      1 0 1
XOR:  0 1 1 1
```

● Step 3: Continue the division till the last bit of M(x) to get the CRC.

```
          1 1 1 0 1 1
101 | 1 1 0 1 0 1 0 0
      1 0 1
XOR:    1 1 1
        1 0 1
          1 0 0
          1 0 1
              1 1 0
              1 0 1
                1 1 0
                1 0 1
                  1 1  <-- CRC
```

● Step 4: Append the calculated CRC (i.e. 11) to the M(x) (i.e. 110101) to get the Codeword T(x)

∴ The Codeword T(x) = 11010111

● Verification of the Codeword at the receiver's end:

● For detection of error in the receiver's end, divide the T(x) with G(x). If the final remainder is Zero, then there is no error.

```
          1 1 1 0 1 1
101 | 1 1 0 1 0 1 1 1
      1 0 1
XOR:    1 1 1
        1 0 1
          1 0 0
          1 0 1
              1 1 1
              1 0 1
                1 0 1
                1 0 1
                0 0 0
```

∵ The remainder is Zero

∴ There is no error in transmission (or, the codeword is recived without error)

Figure 4.2: Working example of CRC calculation and verification.

7	6	5	4	3	2	1
D	D	D	P	D	P	P

Figure 4.3: Format of 7-bit Hamming Code

- To find out which parity bit checks a particular data bit, rewrite the data bit position as a power of 2:

$$\begin{aligned} 7 &= 1+2+4 \\ 6 &= 2+4 \\ 5 &= 1+4 \\ 3 &= 1+2 \end{aligned}$$

- Now, from the above we can say that the parity bit $P1$ checks data bits $3, 5, 7$, since the number 1 occurs in those positions. From the working of parity check method we know that the parity bit itself is included in the calculation. As such:

$$\begin{aligned} P1 \quad &checks \quad 1, 3, 5, 7 \\ P2 \quad &checks \quad 2, 3, 6, 7 \\ P4 \quad &checks \quad 4, 5, 6, 7 \end{aligned}$$

- We can now take an example to illustrate the working of a 7-bit hamming code.

Example 1: If the given data bits are '1010', calculate the codeword for 7-bit Hamming Code with even parity.

Solution:

- First, we need to place the data bits in the codeword from left-to-right as shown in figure 4.4

D7	D6	D5	P4	D3	P2	P1
1	0	1		0		

Figure 4.4: Placing the data bits in 7-bit Hamming Code

- Next, Calculate P1. P1 checks D3, D5, and D7. Since, we need to have even parity, and there are two numbers of 1's in those data positions, we have to chose P1 as '0' to maintain even parity, as shown in figure 4.5

D7	D6	D5	P4	D3	P2	P1
1	0	1		0		0

Figure 4.5: Calculate P1

- Similarly, calculate P2. P2 checks D3, D6, and D7. There is only one '1' in those positions. So set 'P2=1' to maintain even parity, as shown in figure 4.6

D7	D6	D5	P4	D3	P2	P1
1	0	1		0	1	0

Figure 4.6: Calculate P2

D7	D6	D5	P4	D3	P2	P1
1	0	1	0	0	1	0

Figure 4.7: Calculate P4

- Finally, calculate P4. P4 checks D5, D6, and D7. There are two 1's in those positions. So set 'P4=0', as shown in figure 4.7

$\therefore$ The required 7-bit Hamming Code is: '1010010'.

Example 2: For the 7-bit Hamming Code with codeword '1010110', check for and correct the error if any, using even parity.

Solution:

- To check and correct error in case of Hamming Codes, we need to construct the *Error Word*.
- The error word consists of the recalculated outcome of the parity bit positions. For 7-bit hamming code, it is organised as shown in figure 4.8

P4	P2	P1

Figure 4.8: Error word format for 7-bit Hamming Code

- If the parity mathces with the requirement, put a Zero in that position, elese put a one.
- The given codeword is shown in figure 4.9

D7	D6	D5	P4	D3	P2	P1
1	0	1	0	1	1	0

Figure 4.9: The given codeword

- To check for transmission error, first check the parity for each bit positions corresponding to P4. Parity for P4 is calculated using the bit positions P4, D5, D6, and D7. There are two number of 1's in those position. Since the transmission was done using even parity, and the outcome is even, put zero into the P4 position of the error word, as shown in figure 4.10

P4	P2	P1
0		

Figure 4.10: Calculate error word for P4

- Next, check P2. Parity chekcing for P2 is done using the positions P2, D3, D6, and D7. There are three 1's in those positions, which is not even. As such, put 'P2=1' in the error word, as shown in figure 4.11

P4	P2	P1
0	1	

Figure 4.11: Calculate error word for P2

- Similarly, check P1. Parity checking for P1 is done using the posiitons P1, D3, D5, and D7. There are three 1's in those positions, which is not even. Hence, put 'P1=1' in the error word, as shown in figure 4.12

P4	P2	P1
0	1	1

Figure 4.12: Calculate error word for P1

- If the recalculated error word is Zero, then there is no transmission error.
- But in this case, we have got a non-zero value, which indicates transmission error.
- To find out which bit position has error, convert the error word to decimal.
- In this case, $011_2 = 3_{10}$. This indicates that the third bit from the RHS in the transmitted codeword is in error.
- To correct the error, simply invert that bit, as shown in figure 4.13

D7	D6	D5	P4	D3	P2	P1
1	0	1	0	~~1~~ 0	1	0

Figure 4.13: The corrected codeword

$\therefore$ The corrected codeword is: '1010010'.

4.4 Flow Control

4.4.1 An Unrestricted Simplex Protocol

- Data is transmitted only in one direction.
- The network layer of the sender and the receiver is always ready.
- Infinite buffer space is available, and processing time can be ignored.
- The communication channel is noise and error free.
- No sequence number or acknowledgement is used.

As can be seen from the above points, this is an ideal but impractical protocol. This protocol simply states the factors to be considered in flow control. The subsequent protocols adds more and more restriction to arrive at a practical protocol.

4.4.2 Stop-and-Wait Flow Control

- The source transmits one frame.
- After receiving the frame, the destination indicates its willingness to accept another frame by sending back an acknowledgement to the frame just received.
- The source must wait until it receives the acknowledgement before sending the next frame.
- The destination can thus stop the flow of data by simply withholding the acknowledgement.
- The channel is half-duplex.
- The communication channel is noise and error free.

4.4.3 Stop-and-Wait Protocol for Noisy Channel

It involves two states:

(a) *Normal Operation:*

- Works same as the regular Stop-and-Wait protocol. However, it now uses sequence number for each frame transmitted.
- Receives the same sequence numbered acknowledgement.

(b) *Time Out:*

- If the acknowledgement is not received within certain amount of time, the sender times out, and retransmits the frame.
- Timeout may occur for two reasons:
  (i) If the acknowledgement is lost.
  (ii) If the transmitted frame is lost or damaged.

4.4.4 Sliding Window Protocol

- The communication channel is full-duplex.
- Destination allocates buffer space for n frames.
- So, the destination can accept n frames, and the sender is allowed to send n frames without waiting for any acknowledgement.
- To keep track of which frames have been acknowledged, each is labelled with a sequence number.
- The destination acknowledges a frame by sending an acknowledgement that includes the sequence number of the *next* frame expected.
- This acknowledgement also implicitly announces that the destination is prepared to receive the next n frames, beginning with the number specified.
- This scheme can also be used to acknowledge multiple frames. For example, after receiving frames 2, 3, and 4, the destination can send an acknowledgement with sequence number 5.
- The sender maintains a list of sequence numbers that it is allowed to send.
- The receiver maintains a list of sequence numbers that it is prepared to receive.
- Each of these lists can be thought of as a window of frames.
- For example, say sequence number field used in the frame uses 3-bits. So, sequence numbers can range from 0 to 7.
- The actual window size need not be the maximum possible size for a given sequence number length.
- Each time a frame is sent, the window shrinks.
- Each time an acknowledgement is received, the window grows.
- The acknowledgement is sent in the form of *RR* (Receive Ready).
- The flow may be completely stopped by the destination by sending a *RNR* (Receive Not Ready). For example, 'RNR5' means acknowledgement up to frame number 4, but no more frame can be accepted. When the destination is ready to receive more frames, it send a 'RR5'.
- The working principle of sliding window protocol is shown in figure 4.14, and an working example of the protocol is shown in figure 4.15

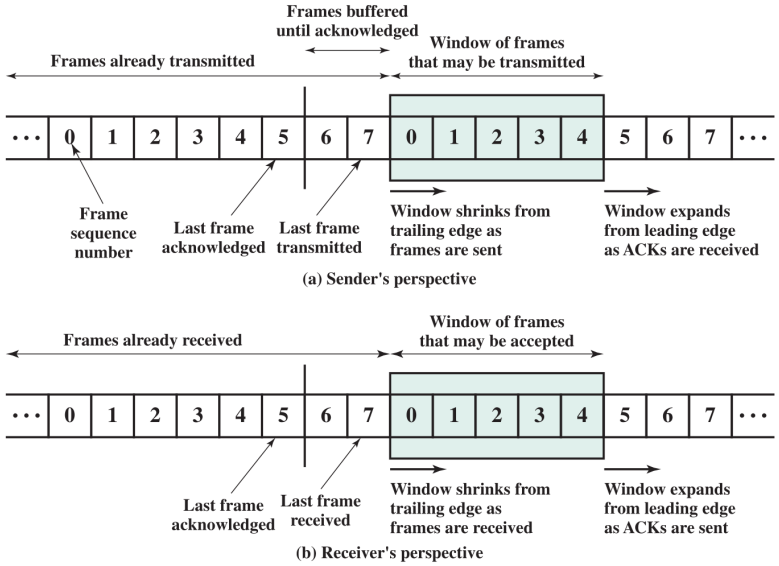

Figure 4.14: Working principle of Sliding Window Protocol (Courtesy: Data and Computer Communications by Stallings, 10^{th} Ed.)

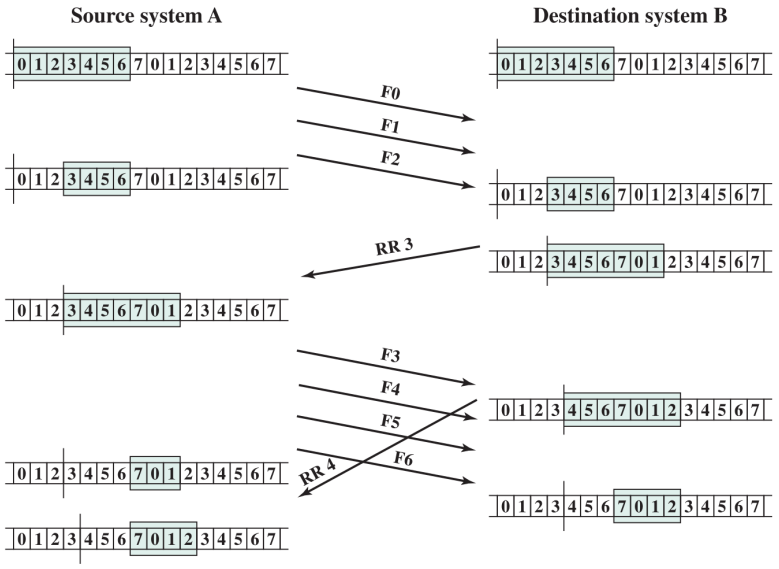

Figure 4.15: A working example of Sliding Window Protocol (Courtesy: Data and Computer Communications by Stallings, 10^{th} Ed.)

4.4.5 Error Handling in Sliding Window Protocol

There are two approaches for error handling in the sliding window protocol:

(i) *Go-Back-N:*

- This approach is shown in figure 4.16
- While no error occurs, the destination will acknowledge (RR) incoming frames as usual.
- In case of error, the destination sends a negative acknowledgement (REJ) for that frame.
- The destination will discard that frame and all future incoming frames until the frame in error is received correctly.

(ii) *Selective Repeat:*

- This approach is shown in figure 4.17
- The only frames retransmitted are those that receive a negative acknowledgement (SREJ), or that time out.
- This approach minimizes the amount of retransmission.
- However, larger buffer space, and complex logic needs to be used.

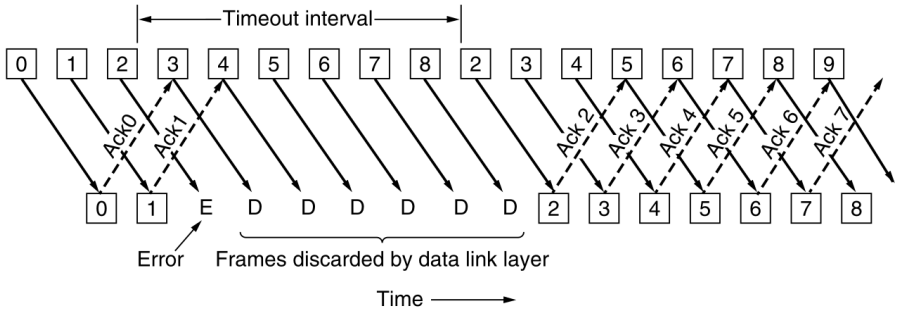

Figure 4.16: The Go-Back-N approach to error handling in the sliding window protocol (Courtesy: Computer Networks by Tanenbaum, 5th Ed.)

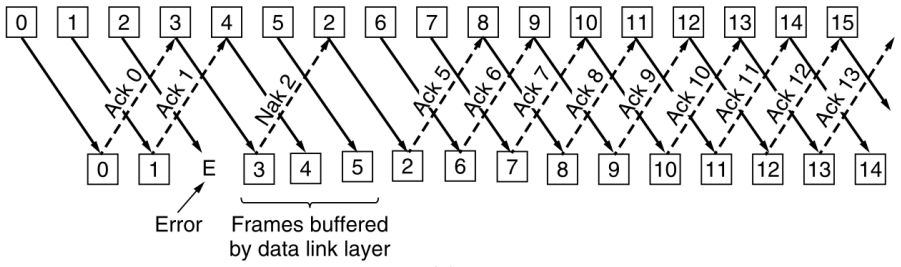

Figure 4.17: The Selective Repeat approach to error handling in the sliding window protocol (Courtesy: Computer Networks by Tanenbaum, 5th Ed.)

4.4.6 Piggybacking

- When both the stations exchange data, they must maintain two windows — one for transmit, and the other for receive.
- They also needs to send the acknowledgements.
- For efficient acknowledgement handling, piggybacking is used.

– If a station has both data and an acknowledgement to send, both of them are sent in one frame.
– If a station has no data but an acknowledgement to send, it sends a separate acknowledgement frame.
– If a station has data but no acknowledgement to send, the frame is sent with the last acknowledgement number. The other station upon receipt of a duplicate acknowledgement number simply ignores it.