**CAPITULO 3**

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 several 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.

Diagram

Description automatically generatedTo 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. Frame management forms the heart of what the data link layer does.

Placement of the data link protocol:

Diagram, schematic

Description automatically generated

**3.1.1 Services provided to the data network layer**

On the source machine is an entity, call it a process, in the network layer that hands some bits to the data link layer for transmission to the destination. The job of the data link layer is to transmit the bits to the destination machine so they can be handed over to the network layer there.

The data link layer can be designed to offer various services. The actual services that are offered vary from protocol to protocol. Three reasonable possibilities that we will consider in turn are:

1. Unacknowledged connectionless service.

2. Acknowledged connectionless service.

3. Acknowledged connection-oriented service.

Unacknowledged connectionless service consists of having the source machine send independent frames to the destination machine without having the destination machine acknowledge them.

With acknowledged connectionless service, there are still no logical connections used, but each frame sent is individually acknowledged. In this way, the sender knows whether a frame has arrived correctly or been lost. If it has not arrived within a specified time interval, it can be sent again. Providing acknowledgements in the data link layer is just an optimization, never a requirement.

With a connection-oriented service, the source and destination machines establish a connection before any data are transferred. Each frame sent over the connection is numbered, and the data link layer guarantees that each frame sent is indeed received. Furthermore, it guarantees that each frame is received exactly once and that all frames are received in the right order

**3.1.2 Framing**

The bit stream received by the data link layer is not guaranteed to be error free. It is up to the data link layer to detect and, if necessary, correct errors.

The usual approach is for the data link layer to break up the bit stream into discrete frames, compute a short token called a checksum for each frame, and include the checksum in the frame when it is transmitted.

Breaking up the bit stream into frames can be done in 4 possible ways:

1. Byte count.

2. Flag bytes with byte stuffing.

3. Flag bits with bit stuffing.

4. Physical layer coding violations.

----- 1. Byte Count -----

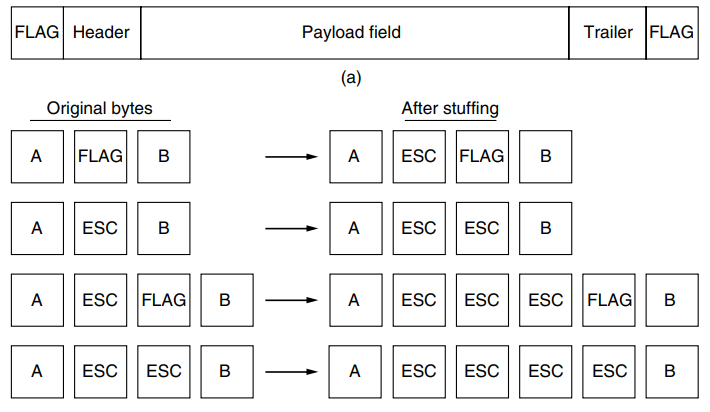
Graphical user interface, timeline

Description automatically generatedThe first framing method uses a field in the header to specify the number of bytes in the frame. When the data link layer at the destination sees the byte count, it knows how many bytes follow and hence where the end of the frame is.

The trouble with this algorithm is that the count can be garbled by a transmission error. For this reason, the byte count method is rarely used by itself.

----- 2. Flag bytes with byte stuffing -----

The second framing method gets around the problem of resynchronization after an error by having 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.

However, there is a still a problem we have to solve. 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**. If an escape byte occurs in the middle of the data it, too, is stuffed with an escape byte.

----- 3. Flag bits with bit stuffing -----

The third method of delimiting the bit stream gets around a disadvantage of byte stuffing, which is that it is tied to the use of 8-bit bytes. Framing can also be done at the bit level, so frames can contain an arbitrary number of bits.

Each frame begins and ends with a special bit pattern, 01111110 or 0x7E in hexadecimal. This pattern is a flag byte. Whenever the sender’s data link layer encounters five consecutive 1s in the data, it automatically stuffs a 0 bit into the outgoing bit stream. This **bit stuffing** is analogous to byte stuffing, in which an escape byte is stuffed into the outgoing character stream before a flag byte in the data.

When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically destuffs (i.e., deletes) the 0 bit.

**3.1.3 Error Control**

How to make sure all frames are eventually delivered to the network layer at the destination and in the proper order?

The usual way to ensure reliable delivery is to provide the sender with some feedback about what is happening at the other end of the line. Typically, the protocol calls for the receiver to send back special control frames bearing positive or negative acknowledgements about the incoming frames.

An additional complication comes from the possibility that hardware troubles may cause a frame to vanish completely. In this case, the receiver will not react at all, since it has no reason to react. Similarly, if the acknowledgement frame is lost, the sender will not know how to proceed.

This possibility is dealt with by introducing timers into the data link layer. When the sender transmits a frame, it generally also starts a timer. The timer is set to expire after an interval long enough for the frame to reach the destination, be processed there, and have the acknowledgement propagate back to the sender.

However, if either the frame or the acknowledgement is lost, the timer will go off, alerting the sender to a potential problem. The obvious solution is to just transmit the frame again. However, when frames may be transmitted multiple times there is a danger that the receiver will accept the same frame two or more times and pass it to the network layer more than once. To prevent this from happening, it is generally necessary to assign sequence numbers to outgoing frames, so that the receiver can distinguish retransmissions from originals.

3.2 Error detection and correction

The former strategy uses **error-correcting codes**, and the latter uses **error-detecting codes**. The use of error-correcting codes is often referred to as **FEC (Forward Error Correction)**.

There are two types of errors: **Simple errors** (random and independent from previous error) and **Errors in Bursts** (affect neighbour bits, burst length defined by the first and last bits in error).

**3.2.1 Error-Detecting Codes**

We will examine three different error-detecting codes. They are all linear, systematic block codes:

1. Parity
2. Checksums
3. Cyclic Redundancy Check (CRC)

---- 1. Parity ----

In the first error-detecting code, a single **parity bit** is appended to the data. The parity bit is chosen so that the number of 1 bits in the codeword is even (or odd). Doing this is equivalent to computing the (even) parity bit as the modulo 2 sum or XOR of the data bits. This means that it can detect single-bit errors.

With this strategy, a megabit of data would require 10,000 check bits. To merely detect a block with a single 1-bit error, one parity bit per block will suffice.

One difficulty with this scheme is that a single parity bit can only reliably detect a single-bit error in the block. If the block is badly garbled by a long burst error, the probability that the error will be detected is only 0.5, which is hardly acceptable.

The odds can be improved considerably if each block to be sent is regarded as a rectangular matrix n bits wide and k bits high. Now, if we compute and send one parity bit for each row, up to k bit errors will be reliably detected as long as there is at most one error per row.

However, there is something else we can do that provides better protection against burst errors: we can compute the parity bits over the data in a different order than the order in which the data bits are transmitted. Doing so is called **interleaving or Bi-dimensional parity check**.

Interleaving is a general technique to convert a code that detects (or corrects) isolated errors into a code that detects (or corrects) burst errors.

---- 3. Cyclic Redundancy Check (CRC) ----

This stronger kind of error-detecting code is in widespread use at the link layer: the **CRC (Cyclic Redundancy Check)**, also known as a **polynomial code**. Polynomial codes are based upon treating bit strings as representations of polynomials with coefficients of 0 and 1 only.

For example, 110001 has 6 bits and thus represents a six-term polynomial with coefficients 1, 1, 0, 0, 0, and 1: 1x^5 + 1x^4 + 0x^3 + 0x^2 + 0x^1 + 1x^0.

Polynomial arithmetic is done modulo 2, according to the rules of algebraic field theory. It does not have carries for addition or borrows for subtraction. Both addition and subtraction are identical to exclusive OR.

When the polynomial code method is employed, the sender and receiver must agree upon a **generator polynomial**, G(x), in advance. To compute the CRC for some frame with m bits corresponding to the polynomial M(x), the frame **must be longer** than the generator polynomial.

The idea is to append a CRC to the end of the frame in such a way that the polynomial represented by the checksummed frame is divisible by G(x). When the receiver gets the checksummed frame, it tries dividing it by G(x). If there is a remainder, there has been a transmission error.

The algorithm for computing the CRC is as follows:

1. Let r be the degree of G(x). Append r zero bits to the low-order end of the frame so it now contains (m + r) bits and corresponds to the polynomial (x^r) \* M(x).
2. Divide the bit string corresponding to G(x) into the bit string corresponding to (x^r) \* M(x), using modulo 2 division.
3. Subtract the remainder (which is always r or fewer bits) from the bit string corresponding to (x^r) \* M(x) using modulo 2 subtraction. The result is the checksummed frame to be transmitted. Call its polynomial T(x).

A picture containing chart

Description automatically generated

3.3 Elementary Data Link Protocols

**3.3.3 A Simplex Stop-and-Wait Protocol for a Noisy Channel**

Now let us consider the normal situation of a communication channel that makes errors. Frames may be either damaged or lost completely. However, we assume that if a frame is damaged in transit, the receiver hardware will detect this when it computes the checksum.

If the frame is damaged in such a way that the checksum is nevertheless correct—an unlikely occurrence—this protocol 2 can fail.

----- Protocol 2 -----

Protocol 2 (Stop-and-wait) also provides for a one-directional flow of data from sender to receiver. The communication channel is once again assumed to be error free. The receiver has only a finite buffer capacity and a finite processing speed, so the protocol must explicitly prevent the sender from flooding the receiver with data faster than it can be handled.

------------------------

At first glance it might seem that a variation of protocol 2 would work: adding a timer. The sender could send a frame, but the receiver would only send an acknowledgement frame if the data were correctly received. If a damaged frame arrived at the receiver, it would be discarded. After a while the sender would time out and send the frame again. This process would be repeated until the frame finally arrived intact.

Also, the receiver must be able to distinguish a frame that it is seeing for the first time from a retransmission. The obvious way to achieve this is to have the sender put a sequence number in the header of each frame it sends.

A 1-bit sequence number (0 or 1) is sufficient to achieve this. At each instant of time, the receiver expects a particular sequence number next. When a frame containing the correct sequence number arrives, it is accepted and passed to the network layer, then acknowledged.

Protocols in which the sender waits for a positive acknowledgement before advancing to the next data item are often called **ARQ** (**Automatic Repeat reQuest**) or **PAR** (**Positive Acknowledgement with Retransmission**).

3.4 Sliding Windows Protocol

In the previous protocols, data frames were transmitted in one direction only. In most practical situations, there is a need to transmit data in both directions. Instead of using two separate links for both directions, it is more efficient to use the same link for data in both directions.

The reverse channel normally has the same capacity as the forward channel. In this model the data frames from A to B are intermixed with the acknowledgement frames from A to B.

However, it is still possible to improve on this scheme. When a data frame arrives, instead of immediately sending a separate control frame, the receiver restrains itself and waits until the network layer passes it the next packet. The acknowledgement is attached to the outgoing data frame (using the ack field in the frame header). In effect, the acknowledgement gets a free ride on the next outgoing data frame. This is known as **piggybacking.**

---- Protocol 3 ----

(PAR) allows unidirectional data flow over an unreliable channel.

-----------------------

The essence of all sliding window protocols is that at any instant of time, the sender maintains a set of sequence numbers corresponding to frames it is permitted to send. These frames are said to fall within the **sending window**. Similarly, the receiver also maintains a **receiving window** corresponding to the set of frames it is permitted to accept.

**3.4.2 A protocol using Go-Back-N**

Until now we have made the tacit assumption that the transmission time required for a frame to arrive at the receiver plus the transmission time for the acknowledgement to come back is negligible. Sometimes this assumption is clearly false. In these situations, the long round-trip time can have important implications for the efficiency of the bandwidth utilization.

The problem described here can be viewed as a consequence of the rule requiring a sender to wait for an acknowledgement before sending another frame. If we relax that restriction, much better efficiency can be achieved.

A picture containing logo

Description automatically generatedTo find an appropriate value for w (number of transmitted frames) we need to know how many frames can fit inside the channel as they propagate from sender to receiver. This capacity is determined by the bandwidth in bits/sec multiplied by the one-way transit time, or the **bandwidth-delay** product of the link.

B\*D = number of bits in a frame

One option, called **go-back-n**, is for the receiver simply to discard all subsequent frames, sending no acknowledgements for the discarded frames. This strategy corresponds to a receive window of size 1. In other words, the data link layer refuses to accept any frame except the next one it must give to the network layer. If the sender’s window fills up before the timer runs out, the pipeline will begin to empty. Eventually, the sender will time out and retransmit all unacknowledged frames in order, starting with the damaged or lost one. This approach can waste a lot of bandwidth if the error rate is high.

The other general strategy for handling errors when frames are pipelined is called **selective repeat**. When it is used, a bad frame that is received is discarded, but any good frames received after it are accepted and buffered. When the sender times out, only the oldest unacknowledged frame is retransmitted.

**3.4.3 A Protocol Using Selective Repeat**

An alternative strategy, the selective repeat protocol, is to allow the receiver to accept and buffer the frames following a damaged or lost one. This is protocol 6.

In this protocol, both sender and receiver maintain a window of outstanding and acceptable sequence numbers, respectively. The sender’s window size starts out at 0 and grows to some predefined maximum. The receiver’s window, in contrast, is always fixed in size and equal to the predetermined maximum.

---- Protocol 6 ----

Selective repeat protocol accepts frames out of order but passes packets to the network layer in order. Associated with each outstanding frame is a timer. When the timer expires, only that frame is retransmitted, not all the outstanding frames.

-------------------------

**POWERPOINT**

1. Framing, Error detection and ARQ in common networks

* 1. **Ethernet**
* Framing:
  + Start of frame: preamble + SFD
  + End of frame: end of signal transitions (Manchester code), length
* Error Detection: 32-bit CRC in trailer (ITU-32)
* No ARQ is used. Bit Error Ratio is very low

**1.2 Point to Point Protocol**

* Framing: Flags - 0x7E
* Byte stuffing: ESC – 0x7D
* Error detection – can be negotiated
* No ARQ is used

**1.3 Wireless LAN**

* Framing:

Diagram

Description automatically generated

* HEC (Header Error Check)
* ARQ: modified version of Stop and Wait
  1. **High-Level Data Link Control**
* Framing: FLAGS, Bit stuffing
* Error detection – ITU-16
* ARQ – Selective Repeat and Go-Back-N ARQ (supports both)

2. Reliability in the Protocol Stack

**2.1 Reliability in the TCP/IP Reference Model**

The layered model transforms bit error in packet losses Therefore, packet losses must be repaired. ARQ solutions are an adequate problem solver.

Two strategies can be used:

1. Link-by-Link ARQ
   1. Repairs losses link by link
   2. Requires network elements to:
      1. remember information about packet flows ➔ high processing per frame
      2. store packets in case they must be retransmitted ➔ memory required
2. End-to-end ARQ
   1. Low complexity in intermediate network elements
   2. Packets may follow different end-to-end paths
   3. Not acceptable when Packet Loss Ratio is high

Capacity of one link: C\*(1-PLR)

Capacity using Link-by-Link: CLL = C (one link) = C\*(1-PLR) 🡪 PLR = packet loss ratio , C = capacity (bits/s)

Capacity using End-to-End ARQ: CEE=C\*(1-PLR)K 🡪 k = number of links