# DATA LINK LAYER DESIGN ISSUES

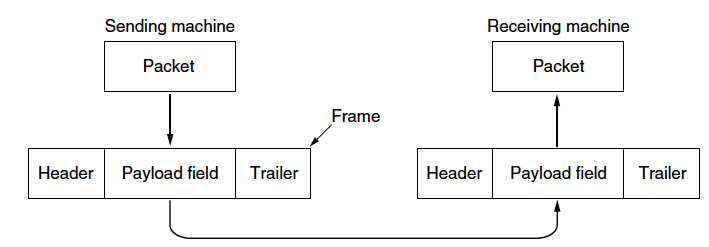
The data link layer uses the services of the physical layer to send and receive bits over communication channels. It has a number of functions, including:

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

2. Dealing with transmission errors.

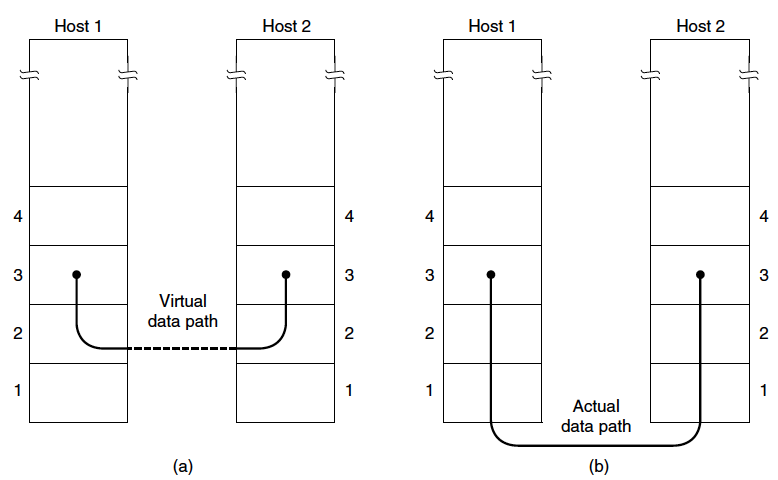
3. Regulating the flow of data so that fast senders do not swamp slow receivers.

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 below. Frame management forms the heart of what the data link layer does.



# Services Provided to the Network Layer

The function of the data link layer is to provide services to the network layer. The principal service is transferring data from the network layer on the source machine to the network layer on the destination machine. 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, as shown in Figure (a) below. The actual transmission follows the path of Figure (b), but it is easier to think in terms of two data link layer processes communicating using a data link protocol. For this reason, we will implicitly use the model of Figure (a) below.



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. **Ethernet is a good example** of a data link layer that provides this class of service. No logical connection is established beforehand or released afterward. If a frame is lost due to noise on the line, no attempt is made to detect the loss or recover from it in the data link layer. This class of service is appropriate when the **error rate is very low**, so recovery is left to higher layers. It is also appropriate for **real-time traffic**, such as voice, in which late data are worse than bad data.

The next step up in terms of reliability is **acknowledged connectionless** service. When this service is offered, 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. This service is useful over unreliable channels, such as wireless systems. **802.11 (WiFi)** is a good example of this class of service. It is perhaps worth emphasizing that providing acknowledgements in the data link layer is just an optimization, never a requirement.

The network layer can always send a packet and wait for it to be acknowledged by its peer on the remote machine. If the acknowledgement is not forthcoming before the timer expires, the sender can just send the entire message again. The trouble with this strategy is that it can be **inefficient**. Links usually have a strict maximum frame length imposed by the hardware, and known propagation delays. The network layer does not know these parameters. It might send a large packet that is broken up into, say, 10 frames, of which 2 are lost on average. It would then take a very long time for the packet to get through. Instead, if individual frames are acknowledged and retransmitted, then errors can be corrected more directly and more quickly. On reliable channels, such as fiber, the overhead of a heavyweight data link protocol may be unnecessary, but on (inherently unreliable) wireless channels it is well worth the cost.

Getting back to our services, the most sophisticated service the data link layer can provide to the network layer is **connection-oriented** service. With this 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. Connection-oriented service thus provides the network layer processes with the equivalent of a reliable bit stream. It is appropriate over **long, unreliable links** such as a satellite channel or a long-distance telephone circuit. If acknowledged connectionless service were used, it is conceivable that lost acknowledgements could cause a frame to be sent and received several times, wasting bandwidth.

When connection-oriented service is used, transfers go through **three distinct phases**. In the first phase, the connection is established by having both sides initialize variables and counters needed to keep track of which frames have been received and which ones have not. In the second phase, one or more frames are actually transmitted. In the third and final phase, the connection is released, freeing up the variables, buffers, and other resources used to maintain the connection.

# Framing

To provide service to the network layer, the data link layer must use the service provided to it by the physical layer. What the physical layer does is accept a raw bit stream and attempt to deliver it to the destination. If the channel is noisy, as it is for most wireless and some wired links, the physical layer will add some redundancy to its signals to reduce the bit error rate to a tolerable level. However, the bit stream received by the data link layer is not guaranteed to be error free. Some bits may have different values and the number of bits received may be less than, equal to, or more than the number of bits transmitted. 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. When a frame arrives at the destination, the checksum is recomputed. If the newly computed checksum is different from the one contained in the frame, the data link layer knows that an error has occurred and takes steps to deal with it (e.g., discarding the bad frame and possibly also sending back an error report). Breaking up the bit stream into frames is more difficult than it at first appears.

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:

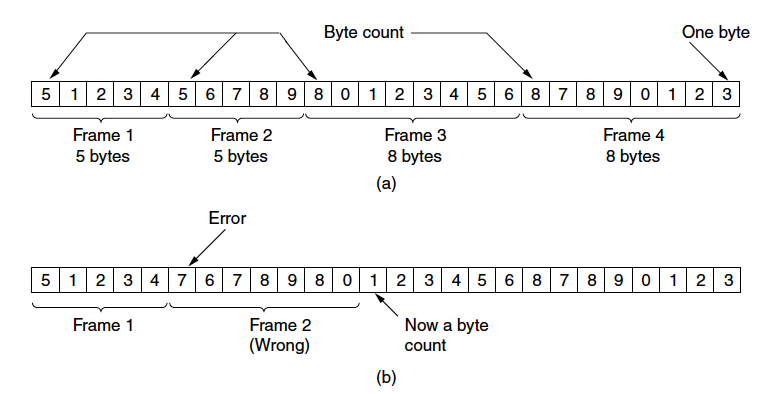
1. Byte count.

2. Flag bytes with byte stuffing.

3. Flag bits with bit stuffing.

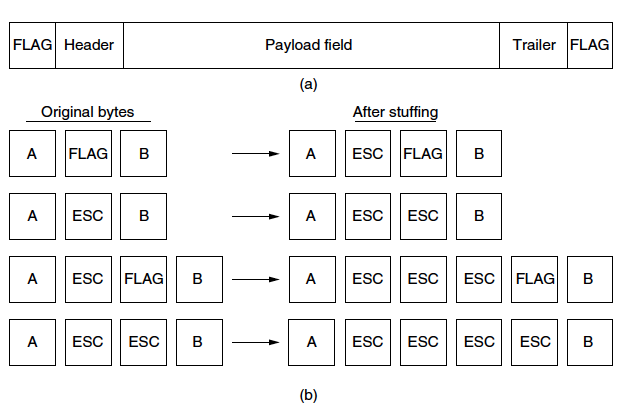
4. Physical layer coding violations.

The 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. This technique is shown in Figure (a) below for four small example frames of sizes 5, 5, 8, and 8 bytes, respectively. The trouble with this algorithm is that the count can be garbled by a transmission error. For example, if the byte count of 5 in the second frame of Figure (b) becomes a 7 due to a single bit flip, the destination will get out of synchronization. It will then be unable to locate the correct start of the next frame. Even if the checksum is incorrect so the destination knows that the frame is bad, it still has no way of telling where the next frame starts. Sending a frame back to the source asking for a retransmission does not help either, since the destination does not know how many bytes to skip over to get to the start of the retransmission. For this reason, the byte count method is rarely used by itself.



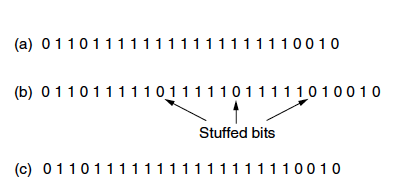
The second framing method gets around the problem of resynchronization after an error by having **each frame start and end with special bytes**. Often the same byte, called a **flag byte**, is used as both the starting and ending delimiter. This byte is shown in Figure (a) below as FLAG. Two consecutive flag bytes indicate the end of one frame and the start of the next. Thus, if the receiver ever loses synchronization it can just search for two flag bytes to find the end of the current frame and the start of the next frame.

However, there is a still a problem we have to solve. It may happen that the **flag** **byte occurs in the data**, especially when binary data such as photographs or songs are being transmitted. This situation would interfere with the framing. 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. Thus, a framing flag byte can be distinguished from one in the data by the absence or presence of an escape byte before it. The data link layer on the receiving end removes the escape bytes before giving the data to the network layer. This technique is called **byte stuffing**. Of course, the next question is: what happens if an escape byte occurs in the middle of the data? The answer is that it, too, is stuffed with an escape byte. At the receiver, the first escape byte is removed; leaving the data byte that follows it (which might be another escape byte or the flag byte). Some examples are shown in Figure (b). In all cases, the byte sequence delivered after destuffing is exactly the same as the original byte sequence. We can still search for a frame boundary by looking for two flag bytes in a row, without bothering to undo escapes.



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 be also be done at the **bit level**, so frames can contain an arbitrary number of bits made up of units of any size. It was developed for the once very popular HDLC (High level Data Link Control) protocol. 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. It also ensures a minimum density of transitions that help the physical layer maintain synchronization. USB (Universal Serial Bus) uses bit stuffing for this reason. When the receiver sees five consecutive incoming 1 bits, followed by a 0 bit, it automatically dyestuffs (i.e., deletes) the 0 bit. Just as byte stuffing is completely transparent to the network layer in both computers, so is bit stuffing. If the user data contain the flag pattern, 01111110, this flag is transmitted as 011111010 but stored in the receiver’s memory as 01111110. Figure 3-5 gives an example of bit stuffing.

With bit stuffing, the boundary between two frames can be unambiguously recognized by the flag pattern. Thus, if the receiver loses track of where it is, all it has to do is scan the input for flag sequences, since they can only occur at frame boundaries and never within the data.



With both bit and byte stuffing, a side effect is that the **length of a frame now depends on the contents of the data it carries**. For instance, if there are no flag bytes in the data, 100 bytes might be carried in a frame of roughly 100 bytes. If, however, the data consists solely of flag bytes, each flag byte will be escaped and the frame will become roughly 200 bytes long. With bit stuffing, the increase would be roughly 12.5% as 1 bit is added to every byte.

The last method of framing is to use a shortcut from the physical layer. The encoding of bits as signals often includes redundancy to help the receiver. This redundancy means that some signals will not occur in regular data. We can use some reserved signals to indicate the start and end of frames. In effect, we are using ‘‘coding violations’’ to delimit frames.

The beauty of this scheme is that, because they are reserved signals, it is easy to find the start and end of frames and there is no need to stuff the data. Many data link protocols use a combination of these methods for safety. A common pattern used for Ethernet and 802.11 is to have a **frame begin** with a well-defined pattern called a **preamble**. This pattern might be quite long (72 bits is typical for 802.11) to allow the receiver to prepare for an incoming packet. The preamble is then followed by a length (i.e., count) field in the header that is used to locate the end of the frame.