CS 425: Computer Networks **Jan-Apr 2020**

Lecture 7: Computer Networks – January 24, 2020

Lecturer: Swaprava Nath *Scribe(s): Ankur Gautam, Bismay Swain, Rishabh Agarwal, Vipul Singhal*

Introduction

The link layer is the layer just above the physical layer. In this lecture, we will discuss how to design the link layer. We will broadly focus on these topics.

Topics in Link Layer

1. Framing
2. Error Detection and Correction
3. Retransmission
4. Multiple Access
5. Switching

Why framing?

1. Phy layer gives only a stream of bits
   - need to 'frame' them to make meaningful messages
2. Framing is the method of sending a message using those bits.

7.1 Framing

Framing creates the link layer's " messages" called frames using the PHYSICAL layer bits. A frame structure is as shown below.

Figure 7.1: A frame delimited by flag bytes

We will look at three methods of framing. These are

1. Byte Count (Simplest method and also rarely used)
2. Byte Stuffing (A little more involved, widely used)
3. Bit Stuffing (Also rarely used)

7.1.1 Byte Count-:

This 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 below for four small example frames of sizes 5, 5, 8, and 8 bytes, respectively.

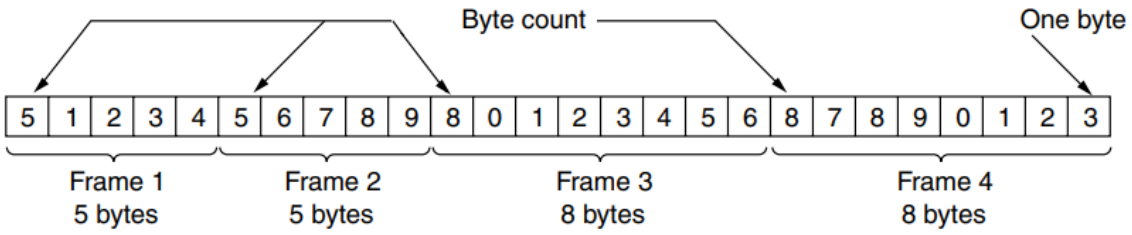

Figure 7.2: A byte stream without errors

Drawbacks-: In this method, once a length byte is corrupted, it is very difficult to get back synchronization. For example, if the byte count (5) in the second frame of the above figure becomes a 7(as shown in the figure below) due to a single bit flip, the destination will get out of synchronization. It will then be impossible to locate the correct start of the next frame. This is why this method is rarely used.

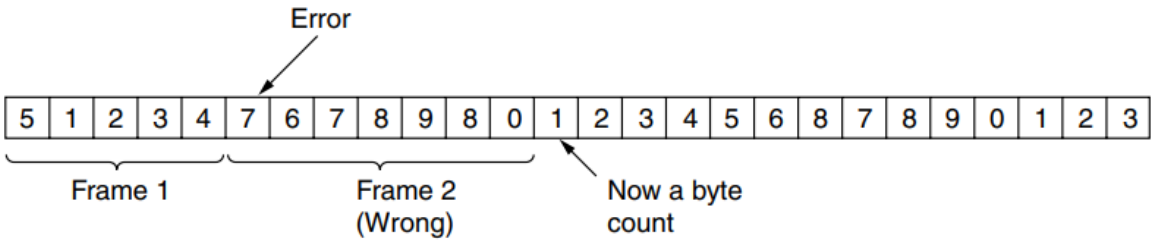

Figure 7.3: A byte stream with one error

7.1.2 Byte Stuffing-:

The second framing method uses special bytes at the start and end of the message. Often the same byte, called a FLAG byte, is used as both the starting and ending delimiter.

However, the FLAG byte can also appear as a part of the message. To resolve this problem, we add another special byte(called ESC) just before FLAG appearing in message. Now, if we come across FLAG not preceded by ESC in receiver end, we can be sure that it is the start/end of the message. If the original message also contains ESC, then we add another ESC in front of it. Some examples are shown in figure 7.4.

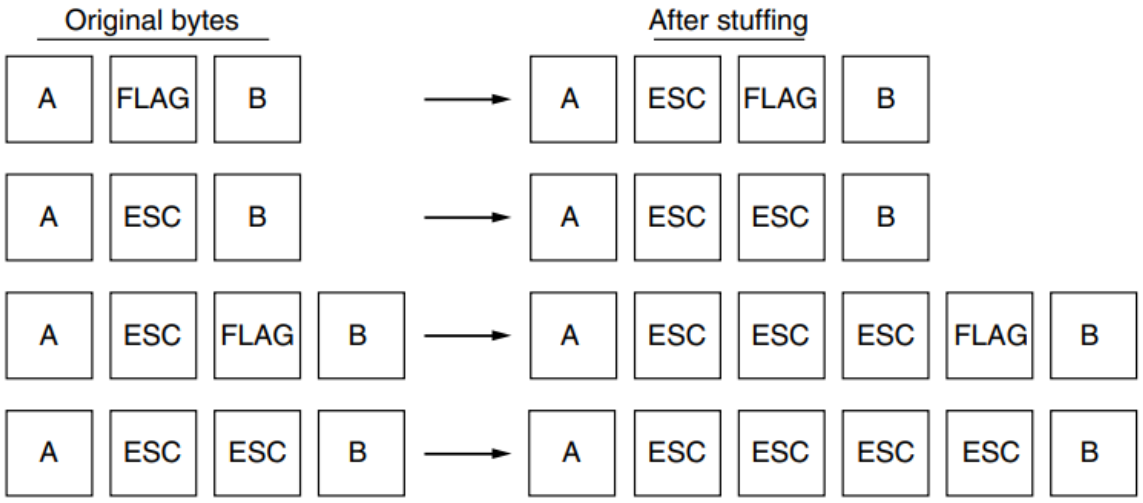

Figure 7.4: Four examples of byte sequences before and after byte stuffing

Drawbacks-: In this type of framing, there is a lot of overhead due to the extra ESC bytes.

7.1.3 Bit Stuffing-:

Bit stuffing is similar to Byte stuffing. However, as the name suggests, we now use bits instead of bits. That means, instead of a byte, the flag can be of any length. In practice, we use a sequence of 6 consecutive 1s as the FLAG. Now, if the message also has a sequence of 6 consecutive 1s, then we cannot distinguish the FLAG from the sequence in the message. To get rid of this problem, we put an extra 0 after any sequence of 5 consecutive 1's present in the message.

Example-:

Data	Encoded Message
0110**111111**0**11111**001	0110**111110**100**111110**001

Table 7.1: Table showing encoding in bit stuffing, stuffed bits are shown in bold in Encoded message

In the above example, we can see after 5 consecutive 1s we add an extra 0 so that 111111 is converted into 1111101 and 111110 is converted to 1111100. Hence, there is no chance of getting a sequence of 6 consecutive 1s in the message anymore. That means whenever the receiver comes across 111111, and it must be the flag.

Drawbacks-: In this type of framing, the receiver has to read the message bit by bit. Reading bit-by-bit is costlier. This is why this method is rarely used.

7.1.4 Application of byte stuffing-:

Byte Stuffing is the most widely used framing method. This is used in Point to Point Protocol(PPP) and many other protocols. In PPP, the FLAG byte is defined as 01111110(0x7E), and the special control escape

byte as 01111101(0x7D). The encoding of these bytes is shown below.

Byte Type	Encoding(Hex)	Encoding(Binary)
FLAG	0x7E	0111 1110
ESC	0x7D	0111 1101

Table 7.2: This is a simple table showing byte stuffing in PPP over SONET

We also encode the byte following ESC(0x7D) by XORing it with 0x20. Thus if the FLAG byte(0x7E) appears as part of the message, it is encoded as 0x7D 0x5E, where 0x5E is obtained by taking XOR of 0x7E with 0x20(just flipping the sixth bit from the right). The advantage being we don't have to check that actual FLAG(which indicates starting of a message) is preceded by ESC or not.

7.2 Error Detection and Correction

Till now, we have assumed that everything transmitted by the server is received as it is by the receiver. However, it is rarely the case, due to the noise in the physical layer, which can introduce erratic bits in the message. Hence, framing is usually followed by Error detection and correction.

Here, we assume that the errors introduced in the message are randomly added and not adversarial. (Someone is not intercepting the message while it is transmitted and manipulating it.)

If we can only **detect** the error, we need to re-transmit the message from the server. However, if the message is carefully constructed, we can also **correct** the detected error in the receiver end which eliminates the requirement of re-transmitting the message.

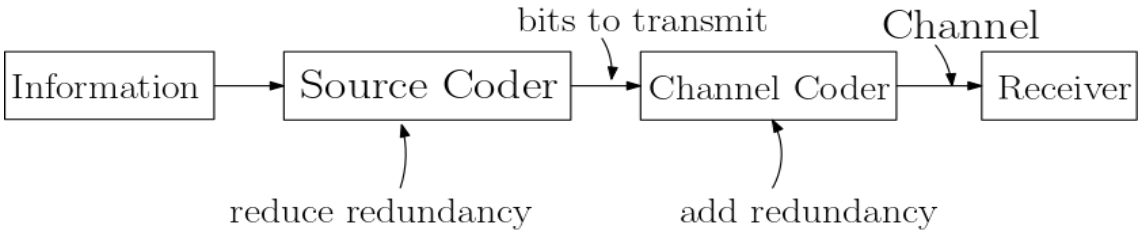

Figure 7.5: Redundancy is added at Channel Coder

To successfully detect and correct the errors, the receiver must have some extra information about the message. The server transmits this redundant information along with the message.

7.2.1 Ways of adding redundancy

In this subsection, we will talk about some very simple ways to add redundancy to our code.

7.2.1.1 Copying

In this method, we make two copies of the data and send them one after another(i.e. 0110 is sent as 0110 0110). Since we are doing this to detect and correct errors, these two questions come immediately.

Q: How many errors can it detect?

Ans: If one bit is corrupted, there will be a mismatch in the two segments of the received message, and we will detect the error. However, if there are two errors and the errors occur at corresponding bits in both the segments(i.e., we receive 0100 0100 instead of 0110 0110), we cannot detect the error. Hence maximum error that can be detected successfully by this method is 1 bit.

Q: How many errors can it correct?

Ans: If one bit mismatch, you can detect an error. However, there is no way you can know which one of them is correct. So by copying, we cannot correct any error.

7.2.1.2 Triple repetition (bitwise)

As the name suggests, we send three repetitions of each bit. For example, 0110 is sent as 000 111 111 000.

We can ask the same questions as above for triple repetitions. As we can observe, the correct message received by the receiver must have only 111 and 000. Say if 000 is corrupted into 011, we can still detect an error. However, if 000 gets changed to 111, we cannot detect an error. So maximum error it can detect is 2 bits. Also, if we change the erratic bit sequence (say 100) to the closest correct bit sequence (000), we can correct one bit of error. However, if two bits are erratic(say 000 converted into 110), the error correction code will wrongly change it into 111. Hence, we can correct only one bit of code.

Above two methods are specific examples of a general approach in networks called CODING. We formally define coding below.

7.2.2 Coding

Definition-: CODING is a mapping from a sequence of D data bits to D+R coded bits. We can see that the cardinality of the domain is 2^D and that of the range is 2^{R+D}.

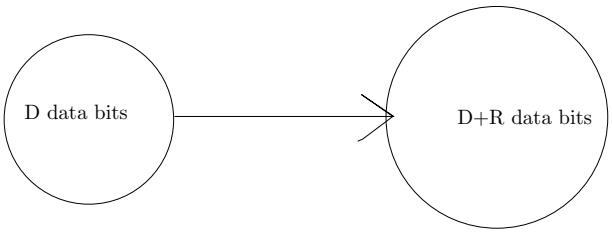

Figure 7.6: Coding is a function mapping from D bits to D+R bits

We will use this CODING function to map our data into a new encoded message which can be used in error detection and correction. Below, we discuss the required properties of a GOOD coding function which will enable us to create such a function.

7.2.2.1 Desirable Properties of a Coding Function

1. The mapping must be **injective**(one to one).
2. The range of the function must be as "**distant**" as possible from each other.
3. The function should be easily **invertible**.
4. The function should have as **few redundant bits** as possible.

A **systematic code** is an error-correcting code in which the input data is embedded in the encoded output. It is a special case of general block codes.

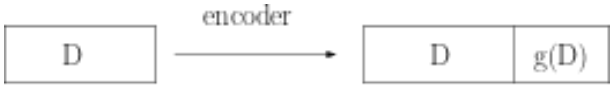

Figure 7.7: A systematic code

We stated above that encoding representations must be as distant as possible from each other. But how do we measure "distance"? One metric to measure distance is **Hamming Distance** which is defined below.

7.2.2.2 Hamming Distance

For vectors $\mathbf{v_1}, \mathbf{v_2} \epsilon \{0,1\}^k$, the hamming distance between $\mathbf{v_1}, \mathbf{v_2}$ is given by

$$d_H(\mathbf{v_1}, \mathbf{v_2}) = \text{Number of bit flips needed to change } \mathbf{v_1} \text{ into } \mathbf{v_2}$$

Examples

1. For $\mathbf{v_1} = [0010]$, $\mathbf{v_2} = [0001]$, $d_H(\mathbf{v_1}, \mathbf{v_2}) = 2$
2. For $\mathbf{v_1} = [0001]$, $\mathbf{v_2} = [1110]$, $d_H(\mathbf{v_1}, \mathbf{v_2}) = 4$

Some of the interesting properties followed by hamming distance are

1. Symmetric
2. Non-Negative
3. Triangle Inequality

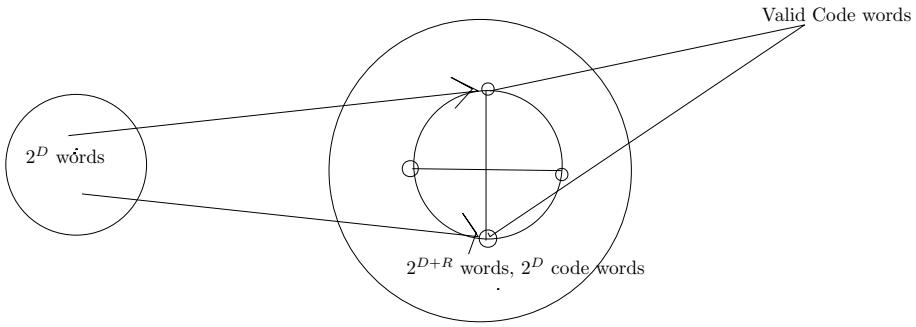

Figure 7.8: Illustration of a typical coding function

Good codes must have a larger distance between the codewords.

Hamming Distance of a Code

Hamming distance of a code is the minimum hamming distance between any pair of codewords. Henceforth, "hamming distance" is being referred to as "distance".

Example

For Triple Repetition Code, Valid Codewords are 000, 111.

$d_H = 3$ as we need 3 flips to change 000 to 111.

Some of the results regarding the distance of code are given below. Hamming gave these results in 1950.

Result 1

If the code has a distance of d+1, then errors up to d bits will always be detected.

Result 2

If the code has a distance of 2d+1, then up to d errors will always be corrected. This is because any received word with d changes can be mapped to the nearest valid codeword.

References

[CN5] AS Tanenbaum, DJ Wetherall, Computer Networks, 5th Ed., Prentice-Hall, 2010.

[WIKI] `https://en.wikipedia.org/wiki/Systematic_code`