Networks Notes - Week 2

Srishti Agarwal

September 2018

1 Bits to Digital Encoding

This encoding takes place as the first encoding in the NIC (Network Interface Card). This encoding is responsible for synchronizing the two devices communicating, mainly their clocks. Generally, the receiving device uses changes in signal to find the clock boundary. Thus, synchronization must be a continuous process, to prevent asynchronization from occurring in the middle of data transfer, i.e. to prevent clock drifts. The following are the different schemes for doing the encoding:

1. NRZ: Non Return to Zero
   This scheme essentially replaces 0s with low pulses and 1s with high pulses. In this case, besides the clock boundary synchronization, the receiving device also uses the values of the high pulses and low pulses to average out a boundary point such that all values higher than the obtained value are read as 1 and all values lower than that as 0. Thus, the main problem with this method is that consecutive 1s and 0s can skew the synchronization, both for the clock and for the average pulse value.

2. NRZI: Non Return to Zero Inverted
   This encoding replaces 0s with keeping the signal at the same level and 1s with changing the level of the signal. Thus it doesn't encounter the problem of averaging a pulse value and it solves the problem of consecutive 1s, but not that of consecutive 0s.

3. Manchester Encoding
   This encoding replaces 0s with a low to high signal change and 1s with a high to low signal change. As the signal is constantly changing, the clock boundary is always picked perfectly. But the efficiency of this method is only 50% as for consecutive 0s or consecutive 1s the pulse has to reset itself.

These processes can be visualized for a specific example as follows:

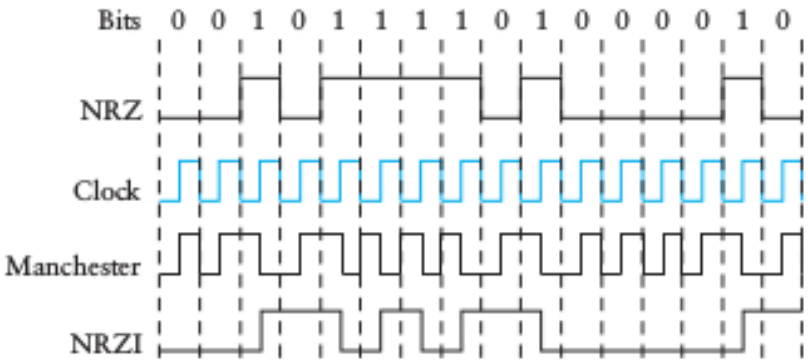

Figure 2.7 Different encoding strategies.

4. 4B/5B
   This encoding replaces blocks of 4 bits with a 5 bit combination such that the 5 bit combination does not have more than 1 leading 0 and more than 2 trailing 0s. More formally,

   $f : \{0,1\}^4 \rightarrow \{0,1\}^5$ *where* $\forall x \in \{0,1\}^4, f(x) = \{x_1x_2x_3x_4x_5 \mid \leq 1$ *leading* $0 ; \leq 2$ *trailing* $0s\}$

   This newly created series of bits limits consecutive 0s to at most 3 and is then encoded using the NRZI encoding which resolves the issue of consecutive 1s. Thus, this scheme is largely effective is eradicating the synchronization issues created by consecutive 0s and 1s in data. Moreover, as for every 4 bits of information, the bit stream has 5 bits of data, this encoding works at 80% efficiency. The exact table used for this encoding is as below:

4-Bit Data Symbol	5-Bit Code
0000	11110
0001	01001
0010	10100
0011	10101
0100	01010
0101	01011
0110	01110
0111	01111
1000	10010
1001	10011
1010	10110
1011	10111
1100	11010
1101	11011
1110	11100
1111	11101

Table 2.4 4B/5B encoding.

Thus, for example, 000011110011 will become 111101110110101, which will be coded using NRZI.

2 Framing

This is the concept of dividing data into chunks. This process takes place in a driver of the operating system, before the encoding of the bits in to digital signals. The main problem that arises with doing so is that the receiving device must have a way to know when a block of data, known as a frame, begins and when it ends. The following are the various approaches taken towards breaking up the data and the ways in which these processes let the receiver know where a frame begins and ends.

1. Byte Oriented
   These methods divide the bit stream into bytes.

   (a) Sentinel Approach/Character Stuffing/Byte Stuffing
   This method indicates where a frame of bytes starts by using a special byte called a Flag before every frame. If the Flag exists in the data, another special character called the Esc (escape) character is inserted before it. If the Esc character exists in the data, another Esc character is added before it. This method works because essentially when the receiving device reads the Esc character, it does not read the next byte as an executable command, but just as data. Thus, using the Esc character before all special characters that need to be read only as data solves the problems of using special characters to demarcate the start of a frame.

   (b) Byte Counting
   This method gives the exact amount of data before the frame and is present at a specific place in the frame.

2. Bit Oriented
   These methods read the bit stream as bits.

   (a) Bit Stuffing
   This method also uses a Flag to demarcate the start of a frame. Specifically, it always uses 01111110 as a Flag before a frame. Thus, in order to avoid the appearance of the Flag in the data, the sender stuffs a 0 after every 5 consecutive 1s in the data. Thus, for the receiver 111110 indicates that it needs to remove the 0 and continue reading the data.

3 Error Control

1. Detection
   Noise in channels can introduce error in frames. This section will discuss techniques that can used by the receiver to detect if the received frame is erroneous or error free. Broadly, we look for a function f that takes bit messages to the message and extra information, i.e.:

$$f : \{0,1\}^k \rightarrow \{0,1\}^n \; where \; n > k$$

$$\forall x = x_1..x_k \in \{0,1\}^k, f(x) = x_1..x_k x_{k+1}..x_n$$

Thus a sender transmits a k-bit message by first converting it into $f(x)$ and then sending $f(x)$. The subset C_f of $\{0,1\}^n$ that is mapped onto by f is known as the n dimension code, and each element in it is known as a code word. Thus more formally,

$$C_f = \{f(x) \mid x \in \{0,1\}^k\}$$

Thus, the cardinality of C_f is 2^k.
Note that by definition, f will be one-to-one.

If the error in the channel is represented by a vector $e \in \{0,1\}^n$, then the message received y is, $y = f(x) \oplus e$. The receiver then checks the message by applying $f(x)$ on the first k bits to the message it receives and checking if the result is equal to the message it received. If e is a null vector, and there is no error, then $y \in C_f$ and the message is received. If e changes $f(x)$ then if,

$$y \notin C_f \implies error\ identified$$

$$y \in C_f \implies error\ not\ identified$$

Thus, the aim of all error detection methods is to reduce the probability of y being in C_f i.e.

$$P(f(x) + e = y \in C_f) \downarrow$$

One way of looking at this probability is assuming that the code words are uniformly distributed over $\{0,1\}^n$. Then:

$$P(y \in C_f) = \frac{2^k}{2^n} = \frac{1}{2^{n-k}}$$

But this is not entirely possible as a uniform distribution would need a random distribution which would need a look up table to be implemented which would slow process down considerably.
Thus, in considering the control for the probability of an error going undetected, another measure is defined, $d(C_f)$, the distance of the code words:

$$d(C_f) = min\{HD(x,y)\}\ where\ x \neq y \in C_f$$

$$\forall x, y \in \{0,1\}^n, HD(x,y) = x \oplus y$$

Thus,

$$d(C_f) = d\ where\ 1 < d < n$$

Claim: Any error that affects less than d bits will be identified for sure.

Proof:
Assume it is not detected.
Then $f(x) + e = y \in C_f$
Thus, $\exists x, y$ such that $HD(x, y) < d$.
This is a contradiction.
Hence proved.

Some specific forms of the f functions that are used are as follows:

(a) Parity

$$f : \{0,1\}^k \rightarrow \{0,1\}^{k+1}$$

$$f(x_1...x_k) = x_1...x_k x_{k+1} \; where \; x_{k+1} = (\sum_{i=1}^{k} [x_i]) \; mod \; 2$$

Here, $d(C_f) = 2$ as even in the worst case, $x_1, x_2 \in \{0,1\}^k$ will differ by 1 bit which means their modular sums will be different bits, and since f is one-to-one, this will be true even vice versa.

(b) IC: Internet Checksum

$$f : \{0,1\}^k \rightarrow \{0,1\}^{k+l} where \; l \; is \; fixed$$

The data is padded such that the number of bits are a multiple of l and divided into r blocks of length l.Thus,

$$x_1...x_l$$

$$x_{l+1}...x_{2l}$$

$$\vdots$$

$$\underline{x_{l(r-1)+1}...x_{lr}}$$

$$y_1...y_l$$

The method of the summation performed on the r blocks of l bits each varies according to the protocol being used, and thus the results and methods of checking vary.

For example, when summation is done through one's complement, the blocks are added up and then 1 is added to that summation to give $y_1...y_l$. Thus, when the receiver receives the entire message, they need to ensure that they obtain a string of 0s when they add all the blocks and $y_1...y_l$. The receiver can detect an error if this fails.

(c) CRC: Cyclic Redundancy Codes

These codes are also called polynomial codes as this method requires converting binary strings into polynomials. In general, the gist of the method is that, the data (binary string x), is converted into polynomial $M(x)$. Then $M(x)$ is turned into a larger degree polynomial $N(x)$ such that $N(x)$ is divisible by a polynomial $C(x)$ that both sender and receiver have agreed upon. Note that $deg(C(x))$ decides the name of the specific code being used. CRC-32 is the most commonly used one where $deg(C(x)) = 32$.

The exact process of encoding and decoding can be understood through the following steps:

i. Let $deg(C(x)) = l$ *and* $x = m_{k-1}...m_0 \in \{0,1\}^k$

ii. Then the polynomial will be calculated as follows:

$$M(x) = m_{k-1}x^{k-1} + ... + m_1x^1 + m_0$$

Note that for a message of k bits, the polynomial will be of at most degree $k-1$.

iii. Then the temporary polynomial $\overline{M(x)}$ will be calculated as follows:

$$\overline{M(x)} = M(x).x^l$$

$$\overline{M(x)} = m_{k-1}x^{k-1+l} + ... + m_0x^l$$

iv. Obtain Quotient $Q(x)$ and Remainder $r(x)$ such that

$$\overline{M(x)} = C(x).Q(x) + r(x)$$

using long division of polynomials such that the arithmetic is mod 2.

v. The polynomial $N(x)$ can be calculated as follows:

$$N(x) = \overline{M(x)} - r(x) = C(x).Q(x)$$

This polynomial is converted back into a bit string which is the final data that needs to be sent by the sender.

vi. On receiving the message, y, the receiver checks if y is divisible by $C(x)$. If it divisible then it is decoded by reading the first k bits as the message.

The correctness of this method, depends on whether the error is divisible by $C(x)$. This can be seen as follows:
Let the error polynomial be $E(x)$.
Thus, the message received is $y = E(x) + N(x)$.
$N(x)$ is divisible by $C(x)$ by definition.
Thus, y divisible by $C(x)$ if $E(x)$ divisible by $C(x)$.

These steps can be better illustrated through an example encoding as done on the next page.

PAGE No.
DATE

Eg. Let $C(x) = x^3 + x + 1$

$\therefore L = 3$

Consider $x = 10110101$

$\therefore M(x) = 1 \cdot x^7 + 0x^6 + 1 \cdot x^5 + 1x^4 + 0x^3 + 1x^2 + 0x + 1$

$= x^7 + x^5 + x^4 + x^2 + 1$

$\therefore \overline{M(x)} = (x^7 + x^5 + x^4 + x^2 + 1)\, x^3$

$= x^{10} + x^8 + x^7 + x^5 + x^3$

$$
\begin{array}{r|l}
 & x^7 + x^2 \\
\hline
x^3 + x + 1 & x^{10} + x^8 + x^7 + x^5 + x^3 \\
 & \underline{x^{10} + x^8 + x^7} \\
 & \quad x^5 + x^3 \\
 & \quad \underline{x^5 + x^3 + x^2} \\
 & \quad\quad x^2
\end{array}
$$

$\therefore \overline{M(x)} = (x^3 + x + 1)(x^7 + x^2) + x^2$

$\therefore Q(x) = x^7 + x^2, \quad r(x) = x^2$

or

$N(x) = \overline{M(x)} - r(x)$

$= x^{10} + x^8 + x^7 + x^5 + x^3 - x^2$

$= x^{10} + x^8 + x^7 + x^5 + x^3 + x^2$

$\therefore$ Message to be sent : 10110101100

(message | remainder)

In general, the effectiveness of this method depends on the choice of $C(x)$. For example,
Result: If $C(x)$ is divisible by $(x+1)$, then all errors $E(x)$ that are odd polynomials i.e. flip an odd number of bits, can be detected.
Proof:
Assume that $E(x)$ is divisible by $C(x)$. Then,

$$E(x) = C(x).P(x)$$

$$E(x) = (x+1).P'(x).P(x)$$

$$E(x) = (x+1).P"(x)$$

As this must hold true for all x, let $x = 1$:

$$E(1) = 2P"(1)\ mod\ 2$$

$$1 = 2P"(1)\ mod\ 2$$

$$1 = 0$$

But this isn't true, thus by contradiction, $E(x)$ is not divisible by $C(x)$.

2. Resolution
   Once the error is detected, there are two ways in which one can deal with them. They are as follows:

   (a) Error Correction
   This is a process that is carried out by error correction codes that are very expensive both in terms of the time they take and in terms of the extra information they require. This is usually only done when the information transfer is over an extremely noisy channel (like satellite communications or wireless communications) and re-transmission won't help. The exact codes used will not be discussed extensively in this course, but will be discussed in Coding Theory.

   (b) Re-transmission
   Once an an error is detected on a less noisy channel the receiver asks for the frame to be re-sent.