CS425A: Computer Networks

Assignment 2

Aditya Tanwar
200057

February, 2023

Question 1 **(20 points)**

Write a program (in any language) that generates an n-bit frame for transmission from a k-bit data block D and a $(n-k+1)$ bit CRC pattern P. Compile and run the program with at least two set of inputs to confirm that this program is generating CRC patterns correctly.

Now, modify the program that performs the following steps:

(a) Generates a message of $k = 10$ bits.

(b) Uses the previous code with $P = 110101$ to generate the corresponding 15-bit frame T for transmission.

(c) Generates transmission errors at any bit positions of T.

(d) Applies CRC to the received frame (i.e. frame T after introducing errors) to determine if the frame should be accepted or discarded.

Attach your code with your report, and also put comments stating the instructions of how to run it.

Solution. The file is named "`A2.cpp`" (without the quotes). The program only uses the STL (Standard Template Library) of C++ and can be compiled using g++ compiler using the command "`g++ A2.cpp`" (without the quotes) which produces an executable.

On running this executable with "`./a.out`" (without the quotes), the program asks if the generation of CRC patterns is to be verified (i.e., the first statement), or introduce some errors in a 15-bit frame test its validity (i.e., the latter parts). Enter 1 for the first statement and 2 for the latter part of the question.
If 2 is entered, the program automatically produces a message of length 10 bits, calculates its FCS with respect to P, introduces some errors to its bits (independently and with some fixed error probability), and tests the new frame for validity.
However, on entering 1, the user is given a choice on whether they want to give input to the values of n and k. If not, the program continues with the default values $n = 10$ and $k = 6$.

The program can be modified easily to generate errors for frames (or messages or polynomials) of different sizes.

Question 2 **(10 points)**

In the Go-back-N ARQ mechanism using k-bit sequence numbers, why is the window size limited to $2^k - 1$ and not 2^k?

Solution 2.
Suppose the window size is $n = 2^k$, and the sender sends a window containing the frames in the order $(0, 1, \ldots, n-1)$ (without loss of generality). The receiver receives all of these successfully, sends an acknowledgement signal and expects to receive the next window also containing the frames in the same order.
Let the acknowledgment signal be lost due to noise or some other reason, so that the sender runs out of time. Thus, it sends frames corresponding to the previous window again. However, since the receiver was expecting frames of the next window, it accepts them into the next window. Now, the receiver has a duplicate of the first window, which was not intended by the sender. Hence, we run into an error.

In the case of window size $n = (2^k - 1)$ however, this problem does not occur. Suppose the sender sends the frames $(0, 1, 2, \ldots n-1)$ and the exact same situation as described above takes place. This time, the receiver will be expecting frame $\#n$, while the sender starts to resend the first window, i.e., it sends a 0. So, we do not run into the same error as in the case when the window size is 2^k.

Thus, the window size is limited to $2^k - 1$ instead of 2^k.

Question 3. **(10 points)**
What is the maximum window size that can be used in the Selective-Reject ARQ mechanism that uses k-bit sequence numbers? Explain your answer.

Solution 3.
Using k-bit sequence numbers, there can be at most 2^k unique frame numbers. Let the window size be w. We discuss the window sizes in two cases, when two consecutive windows can share an overlap, and when they cannot.

In the case of an overlap, without loss of generality, we assume that the first window has frame numbers $f_1 = (0, 1, 2 \ldots, w-1)$, and the next window has frame numbers $f_2 = (w, w+1, \ldots 0 \ldots)$ (since there is at least one overlap in f_1 and f_2). Now, suppose the receiver receives f_1 successfully, acknowledges with RR_w and forwards its window to expect f_2.
However, the sender is not able to receive this acknowledgement signal, and runs into a timeout due to frame 0. Therefore, it will repeat the window f_1 from its end; specifically, it will send frame 0 corresponding to f_1 again. But, as the receiver was expecting the next frame f_2, it will accept the frame 0 into its buffer, and *assume* that the preceding frames of the window f_2 were lost.
Thus, we run into a problem when there is an overlap, because there is ambiguity regarding exactly what window an overlapping frame belongs to. Because of this ambiguity, we cannot have a window size which causes two consecutive windows to have an overlap.

When there is no overlap, we can safely assume that there shall be no ambiguities because we can know exactly to which window a frame belongs. For example, the first w frames belong to the window f_1, and the next w frames (with no overlap with the first w frames) belong strictly to window f_2.
So, it boils down to finding out what window sizes ensure **no overlap**. Since, two consecutive windows are disjoint, they contain exactly $w + w = 2w$ distinct frames with them. Moreover, with k-bit sequence numbers, the maximum number of frames is 2^k. Thus, we have that, $2w \leq 2^k \Rightarrow w \leq 2^{k-1}$.

In summary, the maximum window size that be used in Selective-Reject ARQ mechanism that uses k-bit sequence numbers is 2^{k-1}.

Question 4 **(10 points)**
A channel has a data rate of 4 kbps and a propagation delay of 20 ms. For what range of frame sizes does stop-and-wait give an efficiency of at least 50%?

Solution 4. We are given that U must be at least 50%. We find a constraint on a using the

efficiency relation for stop-and-wait flow control which is given by $U = 1/(1+2a)$:

$$\begin{aligned} U \geq 50\% = \frac{1}{2} & \\ \frac{1}{1+2a} \geq \frac{1}{2} > 0 & \\ 1 + 2a \leq 2 & \qquad \text{(Taking reciprocal)} \\ 2a \leq 1 & \\ a \leq \frac{1}{2} & \end{aligned}$$

Now, given this constraint on a, we find a constraint on the *transmission time*$= t_{\text{frame}}$ using the definition of $a = t_{\text{prop}}/t_{\text{frame}}$ and the fact that we are given $t_{\text{prop}} = 20$ ms. Essentially, we want $20 \text{ ms.}/t_{\text{frame}} \leq 1/2 \Rightarrow 40 \text{ ms.} \leq t_{\text{frame}}$.
Finally, let S_F be the size of a frame (in bits). Then we have,

$$\begin{aligned} 40 \text{ ms.} \leq t_{\text{frame}} &= S_F/(4 \text{ kbps}) \\ \Rightarrow 40 \cdot 10^{-3} &\leq S_F/(4000 \text{ bps}) \\ 160 \text{ bits} &\leq S_F \end{aligned}$$

Thus, we need the size of the frame to be at least 160 bits in order for the efficiency to be at least 50% in stop-and-wait.

Question 5 **(10 points)**

Consider a frame consists of one character of 4 bits. Assume that the probability of bit error is 10^{-3} and that it is independent in each bit.

(a) What is the probability that the received frame contains no errors?

(b) What is the probability that the received frame contains at least one error?

(c) Now assume that one parity bit is added. What is the probability that the frame is received with errors that are not detected?

Solution 5.

(a) Since the probability of bit error is $10^{-3} = p$, we get that the probability of no error for a particular bit is $1 - p = 0.999$. Let E_i be the event that the i^{th} bit is flipped. Then, we have,

$$P(E_i) = p \Leftrightarrow P(\overline{E_i}) = 1 - p$$

We are interested in the probability $P(\overline{E_1} \cap \overline{E_2} \cap \overline{E_3} \cap \overline{E_4})$, i.e., there is no error in the 1^{st} bit, **and** there is no error in the 2^{nd} bit, **and** there is no error in the 3^{rd} bit, **and** there is no

error in the 4^{th} bit:

$$\begin{aligned} P(\overline{E_1} \cap \overline{E_2} \cap \overline{E_3} \cap \overline{E_4}) &= \prod_{i=1}^{i=4} P(\overline{E_i}) && \text{(From independence)} \\ &= \prod_{i=1}^{i=4} (1-p) \\ &= (1-p)^4 \\ &= (1-10^{-3})^4 \\ &\approx 1 - 4 \cdot 10^{-3} && \text{(Binomial approximation)} \\ &= \boxed{0.996} \end{aligned}$$

(b) The event that is there at least one error, is simply the complement of the event that there is no error, i.e., the first part.
Let A be the event that there is at least one error. Then, we have found $P(\overline{A})$ in the first part. We are interested simply in $P(A)$:

$$P(A) = 1 - P(\overline{A}) \approx 1 - 0.996 = 0.004 = \boxed{4 \cdot 10^{-3}}$$

(c) With the addition of a parity bit, we shall now be receiving 5 bits in a frame. However, with parity checks, errors cannot be detected if there are an even number of bit inversions (i.e., errors), as discussed in lectures. Thus, we need to find the probability of cases where there are 2 errors or 4 errors (obviously, the case with 0 errors are omitted). Cases with 1, 3, or 5 errors can be detected by checking the parity of the 5 bits.

In the case of exactly 2 errors, we need to first find the number of pairs of 2 bits. This is given by $\binom{5}{2}$, since there are 5 bits in total (including the parity bit; treating it as any other bit). Further, since each bit has the same probability of being erroneous, and each bit is erroneous independently of every other bit, the probability of bits i and j being erroneous is:

$$P(E_i \cap E_j)\Big|_{i \neq j} = P(E_i) \cdot P(E_j) = p \cdot p = p^2$$

However, in order to get **exactly** 2 errors, we also need to ensure that there is no error in any other bit. In essence, we need to also keep in mind, the probability:

$$P(\bigcap_{k \notin \{i,j\}} \overline{E_k}) = \prod_{k \notin \{i,j\}} P(\overline{E_k}) = (1-p)^3$$

Finally, we have that,

$$\begin{aligned} P(\text{2 errors}) &= \prod_{i \neq j} P(E_i \cap E_j) \cdot P(\bigcap_{k \notin \{i,j\}} E_k) \\ &= \prod_{i \neq j} p^2 \cdot (1-p)^3 \\ &= \binom{5}{2} \cdot p^2 \cdot (1-p)^3 \\ &\approx 10 \cdot 10^{-6} \cdot 0.997 \\ &= 9.97 \cdot 10^{-6} \end{aligned}$$

In the case of **exactly** 4 errors, the number of ways of picking 4 bits out of 5 is $\binom{5}{4}$. Thus, we obtain the probability as:

$$\begin{aligned} P(\text{4 errors}) &= \binom{5}{4} \cdot p^4 \cdot (1-p)^1 \qquad \text{(Since there are 4 errors, 1 correct bit)} \\ &= 5 \cdot p^4 \cdot (1-p) \\ &= 4.995 \cdot 10^{-12} \end{aligned}$$

Finally, we get the probability of error not being detected as,

$$\begin{aligned} P(\text{2 errors}) + P(\text{4 errors}) &= 9.97 \cdot 10^{-6} + 4.995 \cdot 10^{-12} \\ &\approx 9.97 \cdot 10^{-6} \end{aligned}$$

Question 6 **(10 points)**

For $P = 110011$ and $M = 11100011$, find the CRC.

Solution 6. The message M is first padded on the right by five ($= |P| - 1$) 0's (written in a different colour for the sake of distinction) as is done generally in the algorithm. Thereafter, the dividend ($M00000$) is simply divided (modulo 2) by the divisor (P), and 10110110 and 11010 are obtained as the quotient and remainder respectively. The complete long division has been shown on the right.
Since CRC is the remainder, we have,

$$CRC = 11010$$

$$\begin{array}{r} 10110110 \\ P = 110011 \overline{)\ 11100011\color{orange}{00000}} \\ \underline{110011} \\ 010111 \\ \underline{000000} \\ 101111 \\ \underline{110011} \\ 111000 \\ \underline{110011} \\ 010110 \\ \underline{000000} \\ 101100 \\ \underline{110011} \\ 111110 \\ \underline{110011} \\ 011010 \\ \underline{000000} \\ R = 11010 \end{array}$$

Question 7 **(10 points)**

(a) In a CRC error-detecting scheme, choose $P(x) = x^4 + x + 1$. Encode the bits 10010011011.

(b) Suppose the channel introduces an error pattern 100010000000000 (i.e., a flip from 1 to 0 or from 0 to 1 in position 1 and 5). What is received? Can the error be detected?

(c) Repeat part (b) with error pattern 100110000000000.

Solution 7.

(a) The first order of business is to convert the binary string into a polynomial. This is exactly what we do:

$$M = 10010011011 \Leftrightarrow M(x) = x^{10} + x^7 + x^4 + x^3 + x + 1$$

We now move on to dividing $x^4 \cdot M(x)$ by $P(x)$:

$$
\begin{array}{r}
x^{10} + x^6 + x^4 + x^2 \\
\hline
P(x) = x^4 + x + 1 \;\big)\; x^{14} + x^{11} + x^8 + x^7 + x^5 + x^4 \\
\underline{x^{14} + x^{11} + x^{10}} \\
x^{10} + x^8 + x^7 \\
\underline{x^{10} + x^7 + x^6} \\
x^8 + x^6 + x^5 + x^4 \\
\underline{x^8 + x^5 + x^4} \\
x^6 \\
\underline{x^6 + x^3 + x^2} \\
R(x) = x^3 + x^2
\end{array}
\qquad = x^4 \cdot M(x)
$$

We obtain the remainder polynomial $R(x) = x^3 + x^2$, converting it into a binary sequence (of 4 bits), we get $R = 1100$. To encode the message M, we simply concatenate R after it, i.e., the encoding is $MR = 100100110111100$

(b) To know what is received, we simply flip the 1^{st} and 5^{th} bit of MR (the intended pattern), which is, $W = 000110110111100$.
We could have also gotten W by XOR-ing the error pattern with the sent pattern MR, since, positions of flipping are indicated with 1, and XOR-ing with 1 flips a bit, while XOR-ing with 0 does not change a bit.

To check for an error, we simply divide the received pattern by $P(x)$. Since there is an error, the remainder on dividing by $P(x)$ must be non-zero. If it is indeed non-zero, we can conclude there is an error immediately. To avoid verbosity, instead of using $P(x)$, we use $P = 10011$ derived from $P(x)$. The long division (modulo 2) is shown on the right.

$$
\begin{array}{r}
11001110 \\
\hline
P = 10011 \;\big)\; 000110110111100 \\
\underline{10011} \\
10000 \\
\underline{10011} \\
11111 \\
\underline{10011} \\
11001 \\
\underline{10011} \\
10100 \\
\underline{10011} \\
R = 1110
\end{array}
$$

Since, the remainder $R = 1110 \Leftrightarrow R(x) = x^3 + x^2 + x$ is non-zero, thus an error will be **detected**.

(c) Again, to know what is received, we XOR the intended pattern with the error pattern to get,

$$W = (100100110111100) \oplus 100110000000000$$

i.e., $W = 000010110111100$. As for checking if there is an error, we follow the same procedure as before; divide the received pattern by $P(x)$. Again, for the sake of being concise, we use binary strings instead of polynomials.
The long division is towards the right.

$$
\begin{array}{r}
1010100 \\
\hline
P = 10011 \;\big)\; 000010110111100 \\
\underline{10011} \\
10111 \\
\underline{10011} \\
10011 \\
\underline{10011} \\
R = 0
\end{array}
$$

Since the remainder $R = 0000 \Leftrightarrow R(x) = 0$, the error **cannot** be detected.