**Date: 16.02.2024**

**CS425A: COMPUTER NETWORKS**

**Assignment-2**

**Submitted by:**

Anay Sinhal

EXY23017

**QUESTION 1:**

Please refer “Q1\_code.cpp” uploaded as a separate file.

**Compilation:**

* Ensure g++ is installed.
* In the terminal, navigate to the file's directory.
* Compile with g++ -o q1Sol Q1\_code.cpp.

**Running:**

* Execute with ./q1Sol on Unix/Linux or q1Sol.exe on Windows.
* Input the polynomial *P* and message bit-length *k* as prompted.
* For the given question, *P = 110101* and *k = 10*.

**QUESTION 2:**

The Go-back-N ARQ protocol restricts the sender's window size to rather than to prevent sequence number ambiguity. With *k*-bit sequence numbers, a window size of would allow the sequence numbers to wrap around and restart from zero within a single window, making it impossible to distinguish between new frames and retransmissions. This limitation ensures a clear separation between the current window of frames being transmitted and the next, preventing the receiver from incorrectly accepting a frame from the new window as a retransmitted frame from the previous window.

For example, if a sender transmits 7 frames (in a scenario with window size) and begins receiving acknowledgments, there's no risk of mistaking a new frame for an old one due to the sequence numbers wrapping around. This clear separation maintains the integrity of the data transmission process.

**QUESTION 3:**

In Selective-Reject ARQ using k-bit sequence numbers, the maximum window size is . This restriction is necessary to avoid confusion over whether an acknowledgment is for a new frame or a retransmitted frame when sequence numbers wrap around.

For example, consider a system using 3-bit sequence numbers, which allows for 8 distinct sequence numbers (0 to 7). With a window size of , we can have a maximum of 4 frames unacknowledged at any time. If we were to use all 8 sequence numbers, frame 0 could be confused as a new frame or a retransmission of the 8th frame due to sequence number wrap-around. Therefore, a window size of ensures a safe margin to distinguish between frames within the sequence number space.

**QUESTION 4:**

For a channel with a data rate of 4 kbps and a propagation delay of 20 ms, to achieve an efficiency of at least 50% using the stop-and-wait ARQ protocol, we can derive the minimum frame size as follows:

Efficiency (U) in stop-and-wait ARQ is given by:

where *a* is the ratio of the propagation delay to the transmission time.

Given that the minimum efficiency is 50%, we have:

The propagation delay (τ) is 20 ms (or 20 × 10−3 seconds), and the bit rate (R) is 4 kbps (or 4 × 10−3 bits per second). If *x* is the number of bits in a frame, then the transmission time () is . Therefore, .

Substituting *τ* and *R* into *a* gives us:

Substituting *a* into the efficiency formula and solving for *x* yields:

Therefore, the frame size must be at least 160 bits to maintain an efficiency of 50% or higher​​.

**QUESTION 5:**

For a frame consisting of one character with 4 bits and a bit error probability of , we can calculate the probabilities as follows:

(a) The probability that the received frame contains no errors is the probability that each bit is received without error:

(b) The probability  that the received frame contains at least one error is the complement of :

(c) The probability that a frame is received with errors that are not detected, given a parity bit is added for error detection, is calculated by considering even number of errors in a 5-bit frame (4 data bits + 1 parity bit), since an odd number of errors would be detected by the parity bit. The formula for undetected errors with a parity bit, based on the binomial probability distribution, is given as follows:

Where,

When we calculate these probabilities and sum them up, we get:

**QUESTION 6:**

Given the polynomial *P* = 110011 and the message *M* = 11100011, we append zeros equivalent to the degree of *P 1* to the message *M* and perform a binary division to find the CRC.

The steps are as follows:

1. Append five zeros to *M*: 11100011 → 1110001100000.
2. Divide by *P* using binary long division (XOR operation).

The binary long division process:

|  |  |  |
| --- | --- | --- |
|  | 10110110 |  |
| P = 110011 | 1110001100000  110011 |  |
|  | 010111 |
|  | 000000 |
|  | 101111 |
|  | 110011 |
|  | 111000 |
|  | 110011 |
|  | 010110 |
|  | 000000  101100 | |
|  | 110011 | |
|  | 111110 | |
|  | 110011 | |
|  | 011010 | |
|  | 000000 | |
|  | **11010** | |

**CRC = 11010**

**QUESTION 7:**

(a) The initial step in encoding the message using CRC involves translating the binary sequence into polynomial notation. For the message , this translates to the polynomial expression .

Next, we must prepare the message for the division by multiplying it by , which corresponds to padding the original message with four zeros to account for the degree of the generator polynomial .

The multiplication yields . We then proceed with the division of by , which is the core step in determining the CRC.

A math equations on a white background

Description automatically generated

* + The dividend is expanded to .
  + The divisor remains .
  + The result of the division provides us with the remainder polynomial .

Upon converting back to binary form, we obtain . The final encoded message, which combines the original message with the calculated remainder, is given by . This encoded sequence integrates both the data and error-checking elements, ready for transmission.

(b) To ascertain the received sequence, we invert the first and fifth bits of the encoded message , yielding:

*W = 000110110111100*

Alternatively, this result could be achieved by performing an XOR operation between the error pattern and . This method leverages the principle that XOR-ing with 1 flips the bit, whereas XOR-ing with 0 leaves it unchanged.

The next step is to validate the presence of an error by dividing the received sequence *W* by the polynomial *P(x)*. For this calculation, we use the binary representation of *P(x)*, which is *P = 10011*, simplifying the polynomial to its binary equivalent.

Executing the binary division (or modulo 2 division) and observing the remainder, we find:

11001110

*P* = 10011 000110110111100

10011

10000

10011

11111

10011

11001

10011

10100

10011

**1110**

Given the **non-zero** remainder , which in polynomial form is , we conclude that an **error** is present in the received sequence.

(c) Determining the received sequence involves XOR-ing the transmitted pattern with the given error pattern. The operation is as follows:

*W = (100100110111100) ⊕ (100110000000000)*

Hence, the result is:

*W = 000010110111100*

To verify the existence of an error, we divide *W* by the binary equivalent of *P(x)*, which is *P = 10011*. This step is identical to the procedure in the previous part, using binary strings for brevity and clarity.

Performing the binary division, we calculate:

1010100

*P* = 10011 000010110111100

10011

10111

10011

10011

10011

**0000**

The division yields a remainder of *R = 0000*, which translates to *R(x) = 0* in polynomial terms. This indicates that the error introduced does not affect the remainder and thus remains undetected by this error-checking method.