Lab 4: Error handling and detection

1 Learning outcomes

After performing this lab you should have gained understanding of

- how to implement Stop-And-Wait ARQ with control of sequence numbers, time-out and frame retransmit.
- how to implement Cylic Redundancy Check (Cyclic Redundancy Check (CRC)), both creation and validation.

2 Lab tasks

The third and fourth is about the implementation of the necessary Data Link Layer (L2) mechanisms. The primary practical objectives of this lab are to:

- The *Development Node* should be able to construct a full and correct frame for transmission and detect the correctness of a received frame, including the address, frame type, sequence number and CRC.
- Achieve communication reliability by implementing Automatic Repeat Request (ARQ), including retransmission and CRC generation and checking.

3 Task 1: Automatic Repeat Request (ARQ)

The objective of ARQ is to make sure that a transmitted frame is correctly received by the remote host. The ARQ method used in this lab is *Stop-And-Wait*. A transmitted Data Link Layer (L2) data frame must be acknowledged before the next L2 frame can be transmitted. A correct Acknowledgement (ACK) frame contains the same sequence number as that of the transmitted L2 frame. If the received ACK frame contains a not expected sequence number, it must be silently dropped. If a transmitted L2 data frame is not acknowledged within a defined time, it has to be re-transmitted. If the number of re-transmissions of the same L2 frame exceeds a defined limit, that frame must be dropped, the original transmission is declared failed and the process should start a new round with a selection of Light Emitting Diode (LED) and creation of a new message. The task is done when the Development node retransmits in case of lost frames.

3.1 Hints

- Make sure that the CRC functions are de-activated both in the *Master Node* and the *Access Point*. See the *Reference Manual* for details.
- Make sure that each original transmission is given a unique sequence number. Increment the sequence number for each original transmission of a new message. Do not increment the sequence number when performing a re-transmission.
- As in previous labs, after transmission of a data frame, the *Development Node* should be prepared to receive a frame within the specified time, the time-out. If no frame addressed to the *Development Node* is received within the time-out, re-transmit the data frame you just sent, but only if the number of allowed re-transmissions are within the limit. Once a frame addressed to you is received, you can determine the frame's type. You should select which state should be the next one depending on the type of frame that is received.
- If an ACK is received, check that the ACK's source address is equal to the destination address of the just transmitted data frame and also that the sequence number is the same as that of the data frame you just transmitted. If so, you are done with a full round and the *Development Node* can start a new round with the selection of a LED. If not, re-transmit but only if the number of allowed re-transmissions is within the limit.
- There are constants and variables declared for you in the Transmit class that can be used for handling the number of re-transmissions.
- With the *Master Node*'s Dual In-line Package (DIP) switch number 3 you can choose if a correct or a faulty sequence number should be returned to the *Development Node* in order to test your code.

4 Task 2: Cyclic Redundancy Check (CRC)

Parity bits must be generated and set according to the protocol and communication specifications. CRC calculations on received L2 frames should be implemented. If the CRC fails, the frame should be silently dropped. In the final task you should implement this CRC functionality in the *Development Node*:

- The *Development Node* must be able to generate CRC parity bits calculated over the whole L2 frame and add the parity bits to the CRC field of the frame before transmission.
- The *Development Node* must be able to validate the reception of a frame by CRC calculations over the received L2 frame.

The task is done when the CRC generation and validation works properly.

4.1 Hints

- Refer to the text book(s), the *Reference Manual* and the text in reference [1] for the theoretical basis of CRC and to understand how to generate CRC code for a bit-stream with a given divisor, as well as how to check the correctness of a received bit-stream using the CRC code and the divisor. In the *Reference Manual*, you can also find specifications and details for CRC in the current application.

- The generation of CRC parity bits could be done either as one function – generate the parity bits and replace the zeros that fill the CRC field in the frame with the generated *on the run* during the transmission of the bits – or do it in a two step manner – first *generate* the parity bits from the L2 frame, where the CRC field is set to all zeros, and then *copy* the generated parity bits to the CRC field in two separated steps. For these labs, we have defined CRC at L2, thus the latter method is of choice. This method is also easier to implement. There is a function in the Transmit class, which can assist you when adding the generated CRC bits to the frame.

- The CRC validation can, as with the generation, be performed either during the reception of bits following the Start Frame Delimiter (SFD) or when a full L2 frame is available but before decomposing. Also here the second method is the preferred method of choice.

- CRC generation and check is easiest to implement using a byte as the working area and binary operators.

References

[1] Bastian Molkenthin. Understanding and implementing crc (cyclic redundancy check) calculation. `http://www.sunshine2k.de/articles/coding/crc/understanding_crc.html`. Last visited 2024-02-05.