1. Goal

To extend the design of the basic data link layer from Programming Assignment 1 and implement the simplified version of the Go-Back-N ARQ protocol.

2. Layered Architecture

For the purpose of this assignment, assume that each node in the network has three layers: physical layer, datalink layer (DLC) and application layer. Two nodes are connected to one another via a transmission link. Each layer in a node can be thought of as an abstract entity that performs certain functions. Similarly, the link is also an entity that has some functionality. Figure below outlines the three layers in each node connected by a link entity. This assignment requires development of a datalink layer entity that performs error control using the simplified Go-Back-N protocol.

3. Protocol Data Units

Layers communicate through exchange of protocol data units (PDUs). The application layer PDU is called a_pdu, the datalink layer PDU is called d_pdu, and the physical layer PDU is called ph_pdu. The formats of a_pdu and d_pdu are defined below.

```c
/* Application protocol data unit */
#define DATASIZE 128
typedef struct {
    char data[DATASIZE]; /* application data */
} A_PDU_TYPE;
```
/* DLC protocol data unit */
/* ------ Values for the type field in D_PDU_TYPE ---- */
#define D_INFO 0
#define D_RR 1
#define D_REJ 2

typedef struct {
    int address; /* address field */
    int type; /* One of D_INFO, D_RR, D_REJ */
    int number; /* Seq# for D_INFO, Ack# for D_RR, D_REJ */
    enum Boolean p_bit; /* P/F bit; YES=1, NO=0 */
    A_PDU_TYPE a_pdu; /* application PDU */
    enum boolean error; /* YES frame corrupted; otherwise NO. */
} D_PDU_TYPE;

/* ------ Values for the type field in PDU_TYPE ---- */
#define TYPE_IS_A_PDU 0
#define TYPE_IS_D_PDU 1

typedef struct {
    union {
        A_PDU_TYPE    a_pdu; /* and d_pdu as a union */
        D_PDU_TYPE    d_pdu;
    } u;
    int type; /*One of TYPE_IS_A_PDU, TYPE_IS_D_PDU */
} PDU_TYPE

Example: PDU_TYPE *pdu refers to the instance of a pdu. A pdu could be either an a_pdu, or a d_pdu, which can be classified by pdu->type. For each pdu, its content can be accessed by either pdu->u.a_pdu, or pdu->u.d_pdu.


This section describes how the Go-Back-N algorithm for this assignment should be implemented. The skeleton for the program is provided in the file dlc_layer.c.

4.1. The Go-Back-N Variables

For each node, the datalink layer maintains its DLC_Conn_Info_TYPE structure as shown below, and a pointer to the structure is automatically passed to a datalink layer entity when it is invocated.

#define MAXWIN 7
#define MAXBUFFER 16
#define MODULO 8

typedef struct {
    int snd_nxt, snd_una, rcv_nxt, rej_already_sent, window_size;
    PDU_BUFFER_TYPE pdu_buffer;
} DLC_Conn_Info_TYPE;
The DLC_Conn_Info_TYPE structure contains the following fields:

- **snd_nxt**: This is the sequence number of the next info frame to be sent. When sending an info frame, the datalink layer copies this number into the number field of the d_pdu.
- **snd_una**: This is the Ack number of the last RR or REJ that was received, and it is the sequence number of the first unacknowledged info frame sent by the datalink layer to the physical layer.
- **rcv_nxt**: This is the sequence number of the next info frame expected by the datalink layer. This determines if the incoming info frame invokes the sending of a RR or a REJ.
- **window_size**: This is the window size for the Go-Back-N protocol. The window size is set to MAXWIN.

Note: Arithmetic with snd_nxt, snd_una, and rcv_nxt is MODULO window_size arithmetic.

- **rej_already_sent**: When this variable is 1 that indicates a REJ has been sent by this datalink layer, and when it is 0 then the datalink layer, if needed, can send one REJ.
- **pdu_buffer**: This is the transmission buffer that stores the a_pdu's to be sent by the datalink layer. The size of this buffer is equal to the constant MAXBUFFER. An a_pdu must stay in the buffer until RR or REJ, that acknowledges it, is received. The buffer is an internal data structure that can be accessed only using the following functions:
  0 InsertPDUIntoBuffer()
  0 PDU_TYPE * GetPDUFromBuffer()
  0 int DataInPDUBuffer()
  0 UpdatePDUBuffer()

Descriptions of these functions are provided in Section 4.4.

4.2 The Go-Back-N Window

The Go-Back-N protocol manages two windows: the sender window and the receiver window. Both windows have a size of MAXWIN. The sender window should be managed using the variables snd_nxt and snd_una, and the receiver window is managed using rcv_nxt. The difference (modulo the sequence number space) between snd_nxt and snd_una gives the amount of window that has been used up by the sender, i.e., the number of info frames that have been sent, but not acknowledged. The snd_nxt variable moves to the right from snd_una to snd_una + MAXWIN. For every info frame acknowledged, snd_una moves to the right towards snd_nxt, but can never exceed snd_nxt. Since, snd_nxt denotes the next info frame that must be sent, setting it to snd_una may be used for retransmitting all unacknowledged info frames.

4.3 The Algorithm

The following algorithm describes the events that the datalink layer must process, and the actions that it must take. Parts of algorithm may also be outlined in the skeleton code provided in dlc_layer.c.
• The function `FromApplicationToDatalink()` of the datalink layer is called when an a_pdu from the application is received:
  o Insert a_pdu into the transmission buffer; Use `InsertPDUIntoBuffer()`.
  o Send the info d_pdu to the physical layer, if possible, i.e. if there are less than `window_size` sent but unacknowledged a_pdu’s in the transmission buffer; Hint: Use `DataInPDUBuffer()` to obtain the number of all (unsent and sent) a pdu’s in the buffer.

• The function `FromPhysicalToDatalink()` of the datalink layer is called when a d_pdu from the physical layer is received. If the d_pdu is corrupted or its address is not correct or (only in this assignment) P/F bit is set, then simply discard it, else:
  o If the d_pdu is an RR
    • Call `UpdatePDUBuffer()` to delete a_pdu’s in the transmission buffer that may be now acknowledged.
    • Update `snd_una`.
    • Send any unsent a_pdu's from the transmission buffer to the physical layer, if there are now less than `window_size` sent but unacknowledged a_pdu’s in the buffer.
  o If the d_pdu is an REJ
    • Call `UpdatePDUBuffer()` to delete a_pdu’s in the transmission buffer that may be now acknowledged.
    • Update `snd_una` and `snd_nxt`.
    • Send up to `window_size` a_pdu's from the transmission buffer to the physical layer, if there are still a_pdu’s in the buffer. Note that for some of those a_pdu’s being sent this might be retransmission.
  o If the d_pdu is an info frame
    • Check the sequence number of the info d_pdu. If this info frame is out of sequence then discard the d_pdu and if rej_already_sent is 0 then set rej_already_sent to 1 and send one REJ. If rej_already_send is 1, then send appropriate RR.
    • If the d_pdu is the one expected, then increment `rcv_nxt`, send an RR, reset rej_already_sent and send the a_pdu to application by `send_pdu_to_application_layer()`, as you did in Assignment 1.

• To send an info d_pdu to the physical layer, do the following
  o Get a pointer to an a_pdu from the transmission buffer; use `GetPDUFromBuffer()`.
  o Create a d_pdu and copy the contents from the a_pdu to it.
  o Set the remaining fields of the d_pdu; Remember that info frames are always requests; Use `GetReceiverID()` to obtain the address.
  o Start a retransmission timeout; Use `SetTimer()`.
  o Send the d_pdu to the physical layer, as you did in Assignment 1.
  o Increment `snd_nxt`.

• To send a RR or a REJ, simply create a d_pdu, fill in the needed fields, and send it to the physical layer; Remember that REJ is always response, and (in this assignment only) RR is also always response;
• In this assignment, P/F bit for all frames is always set to 0;

• Remember to free every pdu, received either from the application or from the physical layer, after its processing is done.

• When the retransmission timeout expires, the function DatalinkTimerExpired() is called. This function retransmits all unacknowledged a_pdu's. In order to do this, snd_nxt must be set to snd_una and then the pointers to a_pdu's must be extracted from the transmission buffer.

Note: This is the change (or simplification) from the go-back-N ARQ, which requires sending the request RR with P/F=1 when timeout expires.

4.4 Descriptions of Provided Functions

DLC_LAYER_ENTITY_TYPE *dlc_layer_entity refers to the dlc_layer instance of a node that calls any function in dlc_layer.c.

• pdu_free(PDU_TYPE *pdu)
  This function frees a pdu.

• PDU_TYPE *pdu_alloc()
  This function allocates a pdu and returns a pointer to it. A pdu is guaranteed to contain garbage.

• UpdatePDUBuffer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)
  This function uses the number field of the RR or REJ pdu and the snd_una field in the dci, to delete the acknowledged pdu's from transmission buffer, and informs the application layer that thedatalink layer is now ready to receive more pdu's, since there is additional space in the buffer. This function must be called when RRs or REJs are received.

• int DataInPDUBuffer(DLC_Conn_Info_TYPE *dci)
  This function returns the number of a_pdu's in the transmission buffer.

• PDU_TYPE * GetPDUFromBuffer(DLC_Conn_Info_TYPE *dci)
  This function returns a pointer to a pdu that was stored in the buffer. It can be assumed that the buffer always returns a pointer to the correct pdu that the datalink layer should send. This function does not remove the pdu from the buffer.

• InsertPDUIntoBuffer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)
  This function is used by the datalink layer to insert an a_pdu into the transmission buffer when it receives one from the application. It also informs the application to stop sending more a_pdu's if the buffer is full i.e. if there are now MAXBUFFER a_pdu’s in the transmission buffer.
- **SetTimer**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, DLC_Conn_Info_TYPE *dci)
  This function sets a timer that goes off after approximately one round trip time for the connection. Note: Each datalink layer entity has only one timer and calling SetTimer() will cancel any old timer and start a new one. Thus, when multiple info pdu’s are sent one after another, a timer will in effect be the last one set. Also, note that there is no function that clears a timer, thus once a timer is set it will go off sooner or later.

- **DatalinkTimerExpired**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, DL_Conn_Info_TYPE *dci)
  This function is called when the timer expires. In this assignment and this is the simplification of the Go-Back-N protocol, the function must retransmit all unacknowledged sent info frames. It can do so by setting sn_nxt to snd_una and then sending pdu's from the transmission buffer.

- **send_pdu_to_physical_layer**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PD_TYPE *pdu)
  This function sends pdu to physical layer. The content of the pdu must be that of a d_pdu.

- **send_pdu_to_application_layer**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu)
  This function sends pdu to application layer. The content of the pdu must be that of an a_pdu.

- **int GetNodeID**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity)
  This function obtains the address of this node.

- **int GetReceiverID**(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity)
  This function obtains the address of the receiving (i.e. another) node.

- **cse_debug**(int d_level, "format", variables)
  This statement will print if the debug level is set greater than or equal to d_level. The debug level (initially set to zero) can be set from within the simulator interface. The statement is otherwise equivalent to printf statement.

  Example:
  ```
cse_debug(2, "number = %d\n", pdu_to_physical->u.d_pdu.number);
  ```

  If the debug level is 2 or greater, this code is executed and a value of d_pdu.number is printed.
5. Methodology

This assignment requires a design (somewhat simplified) of the Go-Back-N protocol for the datalink layer. The skeleton of this code is given in the Appendix, and provided in the file dlc_layer.c.

a. In directory /class/cis677/new_linux/Lab2/Files/, you will find these files:
   • dlc_layer.c: file containing the outline for the lab.
   • Makefile: makefile for the assignment.
   • Four configuration files: Each configuration file specifies a different error rate for the transmission link.
     o 2nodes.config: no errors in transmission.
     o 2nodes_error1.config: the link error probabilities of 0.1.
     o 2nodes_error2.config: the link error probabilities of 0.2
     o 2nodes_error3.config: the link error probabilities of 0.3
       Note: 30% of errors are address errors and the rest are errors in other parts of the frame.
   • lab2_demo: Demo program for this laboratory that you can experiment with.

b. Copy the above files to your directory.

c. Experiment with lab2_demo.

d. Study Go-Back-N-ARQ and these instructions carefully.

e. Now you are ready to write your program for the datalink layer. You have to fill in the appropriate code in dlc_layer.c; don't worry about the configuration files.

f. To compile your program, type make. This will produce an executable called lab2_exec in your working directory.

g. Now execute your version of the code and use the configuration files to make sure it works.

6. Submissions

- You must submit your source code file dlc_layer.c electronically by the command:
  submit c3461ax lab2 dlc_layer.c

  where 'x' depends on the class section you are in.

- You must also submit hard copies of the following:
  a. A short summary of the laboratory
  b. Your source code.
/*----------- PROGRAMMING ASSIGNMENT 2 -----------*/
* Name          :
* e-mail        :
* CSE account   :
* ---------------------------------------------------- */

/* --- DO NOT REMOVE OR MODIFY  #include STATEMENTS BELOW --- */
#include "cisePort.h"
#include "sim.h"
#include "component.h"
#include "comptypes.h"
#include "list.h"
#include "eventdefs.h"
#include "main.h"
#include "route_activity.h"
#include "sim_tk.h"
#include "dlc_layer.h"

/**************************************************************/
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static int
window_open(DLC_Conn_Info_TYPE *dci)
{
    int result;
    /* Based on a number of a_pdu's in the transmission buffer
       and values for snd_nxt, snd_una and window size
       determine if there is a new a_pdu ready to be sent */

    return result; // result = 1, there is an a_pdu ready
                // result = 0, there is not
}

/**************************************************************/
/* --- DO NOT REMOVE OR MODIFY THIS FUNCTION --- */
static
dlce_layer_receive(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                   GENERIC_LAYER_ENTITY_TYPE *generic_layer_entity,
                   PDU_TYPE *pdu)
{
    /* Gets the appropriate  DLC_Conn_Info_TYPE structure */
    DLC_Conn_Info_TYPE *dci;
dci = Datalink_Get_Conn_Info(dlc_layer_entity, pdu);
    if (DatalinkFromApplication(generic_layer_entity)) {
        FromApplicationToDatalink(dlc_layer_entity, pdu, dci);
    } else if (DatalinkFromPhysical(generic_layer_entity)) {
        FromPhysicalToDatalink(dlc_layer_entity, pdu, dci);
    } return 0;
}
/**************************************************************/
/* --- YOU MUST HAVE THIS FUNCTION --- */
static
FromApplicationToDatalink(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu_from_application,
   DLC_Conn_Info_TYPE *dci)
{
    /* Insert the pdu from the application layer to the
        transmission buffer. Use:
        InsertPDUIntoBuffer(dlc_layer_entity, pdu, dci); */
    /* If possible send info frame. You may use:
       SendInfo(dlc_layer_entity, dci); */
}

/**************************************************************/
/* --- YOU MUST HAVE THIS FUNCTION --- */
static
FromPhysicalToDatalink(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu_from_physical,
   DLC_Conn_Info_TYPE *dci)
{
    /* Check and discard the pdu when error is detected or P/F
       bit is set */
    /* If not discarded, check d_pdu.type and call an
       appropriate function:
       if RR you may use and call DataLinkProcessRR()
       if REJ you may use and call DataLinkProcessREJ()
       if INFO you may use and call DataLinkProcessINFO() */
}

/**************************************************************/
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static
DatalinkProcessRR(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu,
   DLC_Conn_Info_TYPE *dci)
{
    /* Check the address and if not correct discard frame and
       return 0; */
    /* Otherwise:
       Free up space in the retransmission buffer. Use:
       UpdatePDUBuffer(dlc_layer_entity, pdu, dci); */
    /* update snd_una */
    /* Send as many info pdu's as allowed by window. You may
       use: window_open(dci)
       and
       SendInfo(dlc_layer_entity, dci); */
    /* Free pdu */
    return 0;
}
/**YOU DO NOT HAVE TO HAVE THIS FUNCTION**/
static
DatalinkProcessREJ(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu,
DLC_Conn_Info_TYPE *dci)
{
    /* Check the address and if not correct discard frame and */
    /* return 0; */
    /* Otherwise, free up space in the retransmission buffer */
    /* update snd_una and snd_nxt */
    /* Send as many pdu's as allowed by window */
    /* Free pdu */
    return 0;
}

/**YOU DO NOT HAVE TO HAVE THIS FUNCTION**/
static
DatalinkProcessInfo(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
PDU_TYPE *pdu,
DLC_Conn_Info_TYPE *dci)
{
    PDU_TYPE *pdu_to_application;
    /* Check the address and if not correct discard frame and */
    /* return 0; */
    /* Check if the pdu has the expected sequence number */
    /* When out of sequence, then send REJ, discard pdu and */
    /* return 0 */
    /* You may use SendREJ() to send REJ*/
    /* If expected PDU, then increment rcv_nxt, */
    /* set rej_already_sent = 0 and RR is sent */
    /* You may use SendRR() to send RR */
    /* --- Send pdu to application : Same as Lab1 --- */
    /* Free pdu */
    return 0;
}

/**DO NOT CHANGE NAME OF THIS FUNCTION**/
/* The function is automatically called when the timer expires*/
static
DatalinkTimerExpired(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
DLC_Conn_Info_TYPE *dci)
{
    dci->snd_nxt = dci->snd_una;
    /* Send as many pdu's as allowed by window */
    /* You may use: window_open() and SendInfo(); */
    return 0;
}
/*************** YOU DO NOT HAVE TO HAVE THIS FUNCTION *************** /
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static
SendInfo(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
         DLC_Conn_Info_TYPE *dci)
{
    /* get a_pdu to send */
    PDU_TYPE *pdu_to_send;
    pdu_to_send = GetPDUFromBuffer(dci);

    /* Copy it to a new d_pdu and fill the remaining fields */
    /* increment snd_nxt */
    /* Set timer. Use:
       SetTimer(dlc_layer_entity,dci);  */

    /* --- Send d_pdu to physical layer: Same as Lab1 --- */
    return 0;
}

/*************** YOU DO NOT HAVE TO HAVE THIS FUNCTION *************** /
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static
SendRR(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
        DLC_Conn_Info_TYPE *dci)
{
    /* Allocate a new d_pdu and fill in the needed fields */
    /* Send to d_pdu to physical layer */
    return 0;
}

/*************** YOU DO NOT HAVE TO HAVE THIS FUNCTION *************** /
/* --- YOU DO NOT HAVE TO HAVE THIS FUNCTION --- */
static
SendREJ(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
         DLC_Conn_Info_TYPE *dci)
{
    /* Don't send REJ if rej_already_sent = 1, but send RR */

    /* Allocate a new d_pdu and fill in the needed fields */
    /* Send to d_pdu to physical layer */

    /* rej_already_sent set to 1 */
    return 0;
}