CSE 473S Lab Assignment 2

Due date: 11/14/05 by 12:00 midnight

1. Goals

- To extend the design of your *datalink* layer to implement the **Go-Back-N** ARQ protocol

2. Layered architecture

For the purpose of this lab, assume that each node in the network has three layers: *physical layer, datalink layer (DLC) and application layer*. Nodes in the network are connected to one another via links. Each layer in a node can be thought of as an abstract entity that performs certain functions. Similarly, links are also entities that have some functionality. Figure 1 outlines the three layers in a node connected by a link entity. **In this lab, you will learn how these entities communicate with one another, and will develop a DLC layer entity that performs error correction using the Go-Back-N protocol.**

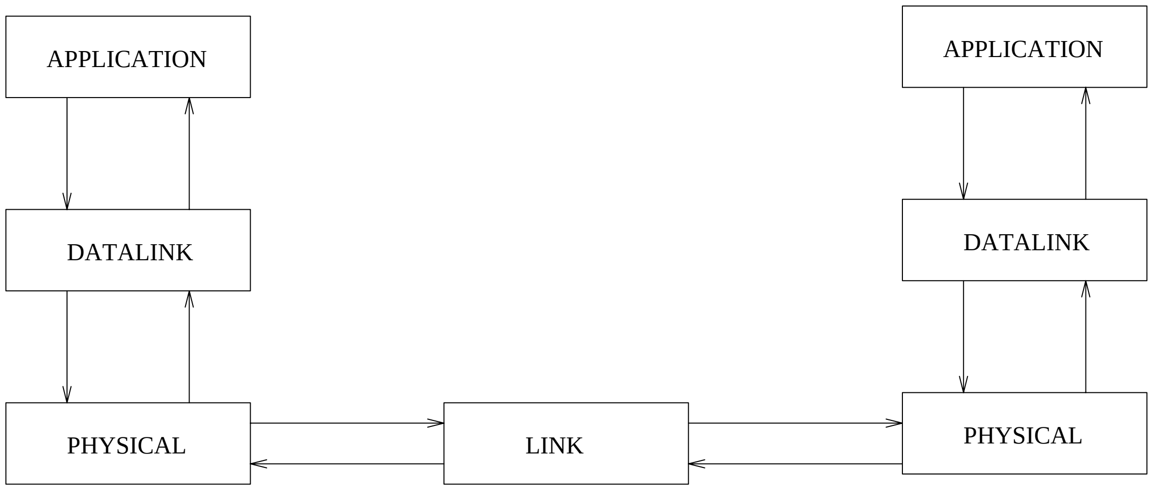

Figure 1: Layered Architecture

3. Protocol Data Units

Each layer communicates through Protocol Data Units (PDU). The application layer PDU is called **A_PDU**, the DLC PDU is called **D_PDU**, and the physical layer PDU is called **PH_PDU**. The A_PDU, D_PDU and PH_PDU formats are defined below. These definitions, together with some others are provided to you in the file `pdu.h`.

```
typedef struct {
        int  snode;                    /* source node address      */
        int  dnode;                    /* destination node address */
        char data[DATASIZE];           /* data                     */
} A_PDU_TYPE;

/* ---------- D_PDU Definition ----- */
#define D_INFO 0
#define D_ACK 1
#define D_NAK 2

typedef struct {
      int  curr_node;                  /* address of this node */
      int  next_node;                  /* address of next node */

      /* ----------- Begin: New fields -------- */
      int  type;                       /* One of D_INFO, D_ACK, D_NAK */
      int  seq_number;                 /* Sequence number of this D_PDU */
      int  ack_number;                 /* Sequence number of ACK/NAK */
      /* ----------- End: New fields -------- */

      A_PDU_TYPE  a_pdu;
      enum boolean  error; /* YES if the packet is corrupted; otherwise NO. */
} D_PDU_TYPE;

/* data unit between physical layers */
typedef struct {
      int  type;                       /* ERR, DATA */
      D_PDU_TYPE  d_pdu;
} PH_PDU_TYPE;

typedef struct {
      union {                          /* structure containing a_pdu */
            A_PDU_TYPE a_pdu;          /* d_pdu or ph_pdu as a union */
            D_PDU_TYPE d_pdu;
            PH_PDU_TYPE ph_pdu;
      } u;
      int type;    /* One of: TYPE_IS_A_PDU, TYPE_IS_D_PDU, TYPE_IS_PH_PDU */
} PDU_TYPE;
```

NOTE: The D PDU definition has three extra fields, and these are used by the Go-Back-N protocol.

4. The Go-Back-N Protocol

This section describes how you must implement the Go-Back-N algorithm for the lab. Several parts of the implementation have already been provided to you as part of the lab.

4.1. The Go-Back-N variables

To implement the Go-Back-N protocol, the DLC layer maintains a structure `DLC_Conn_Info_TYPE` as shown below. A pointer to this structure is automatically passed to the dlc layer entity that you need to write.

```
#define MAXWIN 7
#define MAXBUFFER 7
#define MAXCONNECTIONS 4

typedef struct {
      int    snd_nxt,      /* Sequence number of next D_PDU to be sent */
             snd_una,      /* Sequence num of the first unacknowledged D_PDU */
             rcv_nxt,      /* Seq num of next D_PDU expected to be received */
             nak_already_sent, /* 0 => Can send Nak. 1 => Can't send Nak */
             window_size;  /* Window size for go-back-N. Init to MAXWIN */

      PDU_BUFFER_TYPE pdu_buffer; /* This is the transmission buffer     */
      /* This is only accessible through the following functions:        */
      /*     InsertPDUIntoBuffer(dlc_layer_entity,pdu,dci);              */
      /*     UpdatePDUBuffer(dlc_layer_entity,pdu,dci);                  */
      /*     int DataInPDUBuffer(dci);                                   */
      /*     PDU_TYPE * GetPDUFromBuffer(dci);                           */
} DLC_Conn_Info_TYPE;
```

The `DLC_Conn_Info_TYPE` structure contains the following fields

- `snd_nxt`: This is the sender sequence of the next sequence number to be sent. When sending a `d_pdu`, the DLC copies this number into the `seq_number` field of the `d_pdu`. Note that the sequence numbers range from 0 to `window_size`, i.e., the sequence number space should be one more than the window size.

- `snd_una`: This is the Ack number of the last ack that was received, i.e., it is the sequence number of the first unacknowledged `d_pdu` sent by the dlc layer to the physical layer.

- `rcv_nxt`: This is the sequence number of the next `d_pdu` expected by the dlc layer. This determines if the incoming d pdu invokes the sending of an Ack or a Nak.

- `nak_already_sent`: This is a boolean variable that indicates if a Nak has already been sent by this dlc.

- `window_size`: This is the maximum window size for the Go-Back-N protocol. The window size is initialized to `MAXWIN`.

- `pdu_buffer`: This is a buffer that stores the `a_pdu`'s to be sent by the dlc. The size of the buffer is equal to the size of the Go-Back-N window. pdu's must be stored in the buffer until ack's are received for them. The buffer is an internal data structure that can be accessed only using the following functions. A detailed description of the functions is also provided a later section.

    - `InsertPDUIntoBuffer()`: This is used by the dlc to insert `a_pdu`'s to the buffer when it receives them from the application. It also informs the application if the buffer is full, i.e. if there is a whole window of unacked pdu's.
    - `PDU TYPE * GetPDUFromBuffer()`: This returns a pointer to a pdu that was stored in the buffer. You can assume that the buffer always returns a pointer to the

correct pdu that the dlc must send. This function does not remove the pdu from the buffer.

- `int DataInPDUBuffer()`: Returns the number of `a_pdu`'s in the buffer.
- `UpdatePDUBuffer()`: Deletes pdu's from the buffer and informs the application if the there is space in the buffer. This function must be called when Acks or Naks are received.

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`. In your implementation, 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 pdu's that have been sent, but not acked. The `snd_nxt` variable moves to the right from `snd_una` to `snd_una + MAXWIN`. For every pdu acked, `snd_una` moves to the right towards `snd_nxt`, but can never exceed `snd_nxt`. Since, `snd_nxt` denotes the next pdu that must be sent, setting it to `snd_una` has the effect of retransmitting the unacked packets (or doing the Go-Back-N). Note that these operations are performed modulo the sequence number space, and we have provided you with functions to perform these operations.

4.3. The Algorithm

The following algorithm describes the events that the dlc layer must process, and the action that it must take to process these events. This algorithm is also outlined in the skeleton code provided to you in `dlc_layer.c`

- Receive pdu from application:
  - Insert pdu into pdu buffer
  - Send pdu's to physical (Use window open() to test if any more pdu's can be sent).

- Receive pdu from physical. If pdu has error, then simply discard, else:
  - If pdu is an Ack pdu
    - Call `UpdatePDUBuffer()` to delete packets that are acked.
    - Update `snd_una`
    - Send pdu's to physical
  - If pdu is a Nak pdu
    - Call `UpdatePDUBuffer()` to delete packets that are acked.
    - Update `snd_una` and `snd_nxt`
    - Send pdu's to physical
  - If pdu is an Info pdu
    - Check the sequence number of the pdu (Use `out_of_sequence()`). If the pdu is out of sequence, send a Nak and discard the pdu.

        - If the pdu is the next one expected, then increment `rcv_nxt` (Use `IncrementSequenceNumber()`), send an ack, reset `nak_already_sent` and send the pdu to application.

- To send a pdu to physical, while `window_open()`, do the following
    - Get a pointer to an a pdu from the DLC buffer (Use `GetPDUFromBuffer()`). **Note**: Do not delete the pdu from the buffer until it is acked.
    - Create a d pdu, and copy the contents of the `a_pdu` to it.
    - Set the remaining fields of the d pdu.
    - Start a retransmission timeout (Use `SetTimer()`)
    - Send pdu to the physical layer.
    - Increment `snd_nxt`

- To send an Ack or a Nak, simply create a `d_pdu`, fill in only the needed fields, and send it to the physical layer.

- When the retransmission timeout expires, a function `DatalinkTimerExpired()` is called. You must write this function to retransmit all unacknowledged 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 buffer.

4.4. List of available functions

- `IncrementSequenceNumber(int i,int N)`: Increments i modulo N.

- int `out_of_sequence_pdu(PDU_TYPE *pdu,DLC_Conn_Info_TYPE *dci)`: Returns 0 if `pdu->u.d pdu.seq_number == dci->rcv_nxt` and 1 otherwise

- `int window open(DLC_Conn_Info_TYPE *dci)`: Returns 1 if more pdu's can be sent to the physical, 0 otherwise

- `pdu_free(PDU_TYPE *pdu)`: Free a pdu

- `UpdatePDUBuffer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)`: Uses the ack number field of the pdu and the `snd_una` field in the dci, to delete the acked pdu's, and informs the application layer that the dlc is now ready to receive more pdu's.

- `int DataInPDUBuffer(DLC_Conn_Info_TYPE *dci)`: Returns the number of a pdu's in the buffer

- `PDU TYPE * GetPDUFromBuffer(DLC_Conn_Info_TYPE *dci)`: Returns a pointer to a pdu in the buffer. Does not delete the pdu from the buffer.

- `InsertPDUIntoBuffer(DLCConnectiont *cn,PDU_TYPE *pdu, DLC_Conn_Info_TYPE *dci)`: Inserts pdu into the dci buffer. Informs the higher layer to stop sending more pdu's if the buffer is full (the size of the buffer is set to the window size).

- `SetTimer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, DLC_Conn_Info_TYPE *dci)`: Sets a timer that goes off after approximately one round trip time for the connection. When the timer goes off, the first unacked pdu can be assumed to be lost, and must be retransmitted. **Note**: each dlc layer entity has only one timer. Calling `SetTimer()` will cancel any old timers and start a new one. Thus, when multiple packets are set one after another, the timer will in effect be the last one set.

- `send_pdu_to_physical_layer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu_to_physical)`: sends pdu to physical layer.

- `send_pdu_to_application_layer(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu_to_application)`: sends pdu to application layer.

- `dprintf(int debug_level, ``format'', variables)`: Debug level can be set from within the simulator interface. Use this command to print debugging information in your code. The statement will print if the debug level is set to greater than `debug_level`.

5. Methodology

In this lab, you will design the Go-Back-N protocol for the datalink layer. The skeleton of this code is given in the Appendix, and provided in the `dlc_layer.c`

1) In the file at **http://www.cse.wustl.edu/~cse473s/labs/lab2.zip** you will find the following.

    - `sim.src/pdu.h`: Header file containing some declarations and definitions. You don't need to include this file anywhere in your source code because it is already included in `dlc layer.h`. You will need to use some of the function definitions provided in this file, like `pdu_alloc()`and `pdu_free()`
    - `components.src/dlc_conn_info.h`: File defining the connection info data structure.
    - `components.src/dlc_layer.c`: File containing the outline for the lab.
    - `lab2.vcproj`: Visual Studio project file for the lab.
    - Configuration files: `2nodes_error.config`, `3nodes_error.config`. These _les specify the configuration of the network. In this lab you will only use

2 and 3 node configurations with point-to-point links. Each configuration file specifies a different error rate for the links. The configurations are shown in figures 2 and 3.

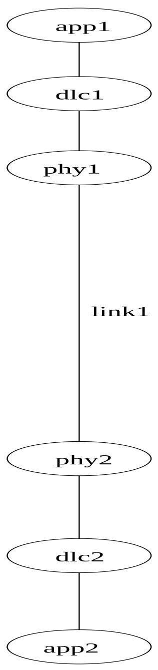

Figure 2: Two node configuration

- `2nodes_error.config` is a two node configuration with link error probability of 0.3.
- `3nodes_error.`config is a three node configuration with link errors. App1 sends to app2. App2 sends to app3. App3 sends to app1. Link1, link2, link3 have error probabilities of 0.1, 0.2, and 0.3 respectively.

- `test/lab2.exe`: A sample executable file for you to experiment with.

2) Unzip the `lab2.zip` file into your user directory at CEC. Once unzipped you should have the following folder structure. `H:/cse473s/lab2`. (You can deviate from this path but you may have to change settings in the project file to do so)

3) You need to set an environmental variable for execution. Follow these instructions closely.
   - Go to Start/Settings/Control Panel/System.
   - Go to the *Advanced* tab
   - Click on Environment Variables
   - Under *User Variables*, click *New*.
   - Enter the following;
     - Variable Name: CISE_LIBRARY_LAB2
     - Variable Value: H:\cse473s\lab2\sim.src

- Click OK for all open dialogs.

4) Experiment with `lab2.exe`. Refer to the lab1 handout for instructions on how to run your program.

5) Study the source code carefully. (Don't worry about the configuration files).

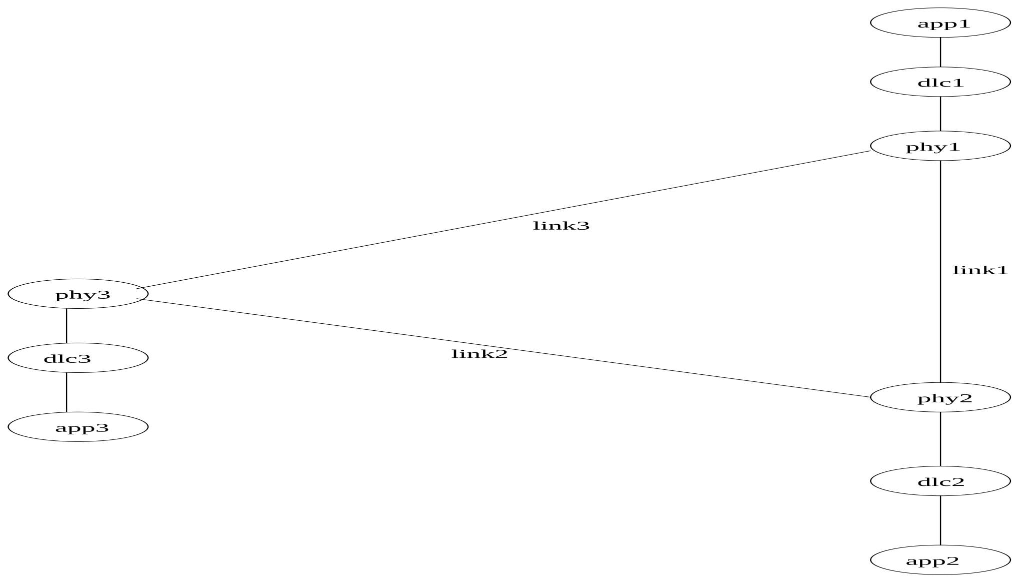

Figure 3: Three node configuration

6) Now you are ready to write your program for the *datalink* layer. All you have to do in this lab is fill in the appropriate code in `dlc_layer.c`.

7) To start, open `lab2.vcproj` file (Opens using Visual Studio .NET). To build the project, use the *Build/Build* menu option. To execute your program, use the *Debug/Start* menu option.

8) Execute your version of the code and use the configuration file to make sure it works.

9) You can use any of the Windows PCs in CEC or the **Oasis** terminal server to work on this project. If you are unfamiliar with Oasis, please get help from the CEC help desk. (or post to the news group: wu.cse.class.cse473)

6. Submissions

You must submit the following in a zip file by email to cse473s@cse.wustl.edu. The subject line of your email should contain “lab2_lastname_firstname. Please also name the zip file as lab2_lastname_firstname.zip (e.g. lab2_pallemulle_sajeeva.zip)

The zip file should contain:

- Your source code for `dlc_layer.c`.
- A Readme file that specifies the following,
    - Whether your code works as expected
    - If the project is incomplete, then what problems you ran into.

7. Miscellaneous

7.1. For questions,

- Come to office hours: The TA office hours may be found at http://www.cse.wustl.edu/~csgrader or in WUGrade.
- Post to the news group: wu.cse.class.473

8. Appendix

```
/* skeleton dlc_layer.c for lab2 */

/* ----- DO NOT REMOVE OR MODIFY  ---- */

#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"

/*****************************************************/
/* Sequence Number Manipulation/Test Macros:
   Accounts for Data Availability in Buffer and Wrap Around */

#define IncrementSequenceNumber(i,N) {(i) = ((i)+1)%(N);}
#define out_of_sequence_pdu(pdu,dci) (pdu->u.d_pdu.seq_number != dci-
>rcv_nxt)

static int
window_open(DLC_Conn_Info_TYPE *dci)
{
int result;
int data_available = DataInPDUBuffer(dci);
int occupied_window;

occupied_window = ((dci->snd_nxt >= dci->snd_una) ?
               (dci->snd_nxt - dci->snd_una) :
               (dci->snd_nxt + dci->window_size + 1 - dci->snd_una));
result = ((occupied_window < data_available) &&
         (occupied_window < dci->window_size));
return result;
}

/****************************************************/
static
dlc_layer_receive(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
              GENERIC_LAYER_ENTITY_TYPE *generic_layer_entity,
              PDU_TYPE *pdu)
{
  DLC_Conn_Info_TYPE *dci;

      dci = Datalink_Get_Conn_Info(dlc_layer_entity,pdu);
          /* Gets the appropriate  DLC_Conn_Info_TYPE structure */

 if (DatalinkFromApplication(generic_layer_entity)) {

      InsertPDUIntoBuffer(dlc_layer_entity,pdu,dci);         /* Insert
A_PDU into dci->buf */
      AttemptToSend(dlc_layer_entity, dci); /* Sends from the buffer */

 } else if (DatalinkFromPhysical(generic_layer_entity)) {

      if (pdu->u.d_pdu.error == YES){
         DatalinkProcessError(dlc_layer_entity, pdu,dci);
      }
      else if (pdu->u.d_pdu.type == D_ACK) {
         DatalinkProcessACK(dlc_layer_entity, pdu,dci);
      }
      else if (pdu->u.d_pdu.type == D_NAK) {
         DatalinkProcessNAK(dlc_layer_entity, pdu,dci);
      }
      else if (pdu->u.d_pdu.type == D_INFO) {
         DatalinkProcessInfo(dlc_layer_entity, pdu,dci);
      }


 }
   return 0;
}


/********************************************************************/
/* DO YOUR CODING FROM HERE */

static
DatalinkProcessError(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                 PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{

      /* Simply Free PDU */

     /* Optional : Send NAK */

     return 0;
}
```

```
/*******************************************************************/
static
DatalinkProcessACK(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                 PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{
     /* Free up space in the retransmission buffer */
     UpdatePDUBuffer(dlc_layer_entity,pdu,dci);

     /* update snd_una  */

     /* Send as many pdu's as allowed by window */
     /* Use window_open(dci) */
     /* and AttemptToSend(dlc_layer_entity, dci); */

     /* Free pdu */
     return 0;
}
/*******************************************************************/
static
DatalinkProcessNAK(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                 PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{
     /* Free up space in the retransmission buffer */
      /* because a NAK may ack a few PDUs */
    UpdatePDUBuffer(dlc_layer_entity,pdu,dci);

    /* set snd_una and snd_nxt */

     /* Send as many pdu's as allowed by window */
     /* Use window_open(dci) */
     /* and AttemptToSend(dlc_layer_entity, dci); */

  /* Free pdu */
      return 0;
}
/*******************************************************************/
static
DatalinkProcessInfo(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                 PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{
PDU_TYPE *pdu_to_application;

     /* OutOfSequence PDU => send Nak, Discard pdu and return 0 */

    /* use  out_of_sequence_pdu() */
    /* and  SendNak() */

     /* Expected PDU => increment rcv_nxt */
/* Use a maximum sequence number of one more than the window size */
/* Use IncrementSequenceNumber()    */
/* Reset nak_already_sent = 0. New Ack received => Naks may be sent */
/* Send an Ack                      */
/* Use SendAck()                    */
/* You can ignore piggybacked acks if you want */
/* if you choose to ignore them, then make sure you send non-
piggybacked acks */


     /* --- Send pdu to application : Same as Lab1 --- */
     pdu_to_application = pdu_alloc();
     bcopy((char *)&(pdu->u.d_pdu.a_pdu),
         (char *)&(pdu_to_application->u.a_pdu), A_PDU_SIZE);
     /* -- Send to app -- */

send_pdu_to_application_layer(dlc_layer_entity,pdu_to_application);

     pdu_free(pdu);
     return 0;
}
/*******************************************************************/
/* Do not change the name of the following function  */
/* This 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; /* Retransmit All Unacknowledged
D_PDUs */

     /* Send as many pdu's as allowed by window */
     /* Use window_open(dci) */
     /* and AttemptToSend(dlc_layer_entity, dci); */
     return 0;
}
/*******************************************************************/
static
AttemptToSend(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                 DLC_Conn_Info_TYPE *dci)
{
PDU_TYPE *pdu_to_send;
PDU_TYPE *pdu_to_physical = pdu_alloc();

     if(window_open(dci)){
        SetTimer1(dlc_layer_entity,dci);

        pdu_to_send = GetPDUFromBuffer(dci);

          /* Copy it to pdu_to_physical amd */
        /* fill the remaining fields of pdu_to_physical */
        send_pdu_to_physical_layer(dlc_layer_entity, pdu_to_physical);

          /* increment snd_nxt */
        IncrementSequenceNumber(dci->snd_nxt,(dci->window_size +1));
     }

     return 0;
}
/*******************************************************************/

static
```

```
SendAck(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{
PDU_TYPE *pdu_to_physical = pdu_alloc();

/* fill in the needed fields */

/* Send to Physical layer */
  send_pdu_to_physical_layer(dlc_layer_entity, pdu_to_physical);

   return 0;
}
/**********************************************************************/
static
SendNak(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity, PDU_TYPE *pdu,
                 DLC_Conn_Info_TYPE *dci)
{
PDU_TYPE *pdu_to_physical;

/* Don't send Nak if nak_already_sent is 1 */
/* But do send an Ack */

pdu_to_physical =  pdu_alloc();

/* fill in the needed fields */
/* Send to Physical layer */
  send_pdu_to_physical_layer(dlc_layer_entity, pdu_to_physical);

/* nak_already_sent is set to 1 */


   return 0;
}

/**********************************************************************/
```