

CIS 677 LAB ASSIGNMENT 1

Date Assigned: ~~07-09-2010~~; Due date: ~~Monday, 07-19-2010~~

IMPORTANT!

Make sure you have a Cse account and you are included in the lab list. In order to verify your presence in the lab, list run this command: `submit c677aa lab0 testfile`. Where “testfile” is an existing file of yours located at the same directory you are launching this command.

1 Goals

- To learn the basic infrastructure of layered architecture and service primitives in computer networks.
- To design a simplified datalink layer.
- To get familiar with the simulator environment used for this and the next lab. ‘

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 simple DLC layer entity.**

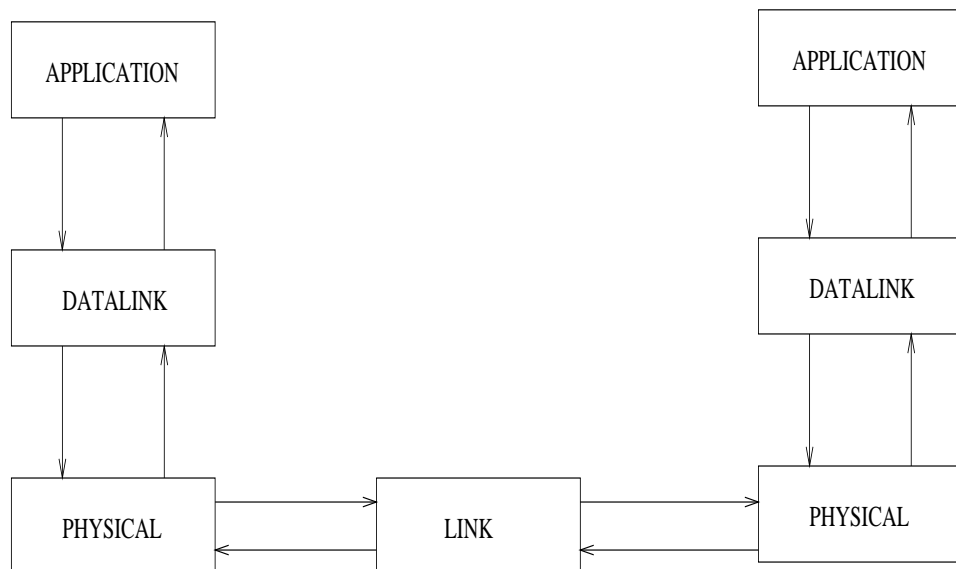


Figure 1: Layered Architecture

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;

typedef struct {
    int          curr_node; /* address of this node      */
    int          next_node; /* address of next node      */
    A_PDU_TYPE  a_pdu;     /* application pdu           */
    enum boolean error;
} D_PDU_TYPE;

typedef struct {
    int          type;
    D_PDU_TYPE  d_pdu;     /* dlc pdu                   */
} PH_PDU_TYPE;

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

Friday, 26 July.

The application layer sends an `a_pdu` to the DLC layer. The DLC layer receives this `a_pdu` and encapsulates it within a `d_pdu`. It then performs its functions on the `d_pdu` and sends the `d_pdu` to the physical layer. In the same manner, the physical layer receives the `d_pdu`, encapsulates it within a `ph_pdu` and sends it to the link entity. The link entity receives a `ph_pdu` from one physical layer and delivers it to the physical layer at the other end. When a physical layer receives a `ph_pdu` from the link, it extracts the `d_pdu` from it and sends it to the dlc layer. The dlc layer checks the `d_pdu` for errors, extracts the `a_pdu` and sends it to the application layer.

4 Service Primitives

Inter layer communication takes place by means of *service primitives*. At the physical-datalink layer interface, there are two service primitives: `PH_DATA_request` and `PH_DATA_indication`. At the datalink-application layer interface, there are two service primitives: `DLC_DATA_request` and `DLC_DATA_indication`.

A service primitive, say `DLC_DATA_request`, is implemented as two procedures: `ApplicationToDatalink()` and `DatalinkFromApplication()`. `ApplicationToDatalink()` puts `a_pdu`'s into the dlc entity, while `DatalinkFromApplication()` gets `a_pdu`'s from the dlc entity. Notice that `ApplicationToDatalink()` is called by the application layer to send `a_pdu`'s, and `DatalinkFromApplication()` is called by the datalink layer to receive these `a_pdu`'s.

5 Methodology

In this lab, you will design the `DatalinkToPhysical` and `DatalinkToApplication` functions for the datalink layer. The outline of these functions is given in the Appendix, and provided in the file `dlc_layer.c`

1. In directory `/usr/class/cis677/SUN/Lab1/files/`, you will find these files:

- `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()`
- `dlc_layer.c`: file containing the outline for the lab.
- `Makefile`: makefile for the lab.
- Four configuration files: `3nodes*.config`. These files specify the configuration of the network. In this lab you will only use a 3 node configuration with point-to-point links. Each configuration file specifies a different error rate for the links. The configuration is shown in figure 2.

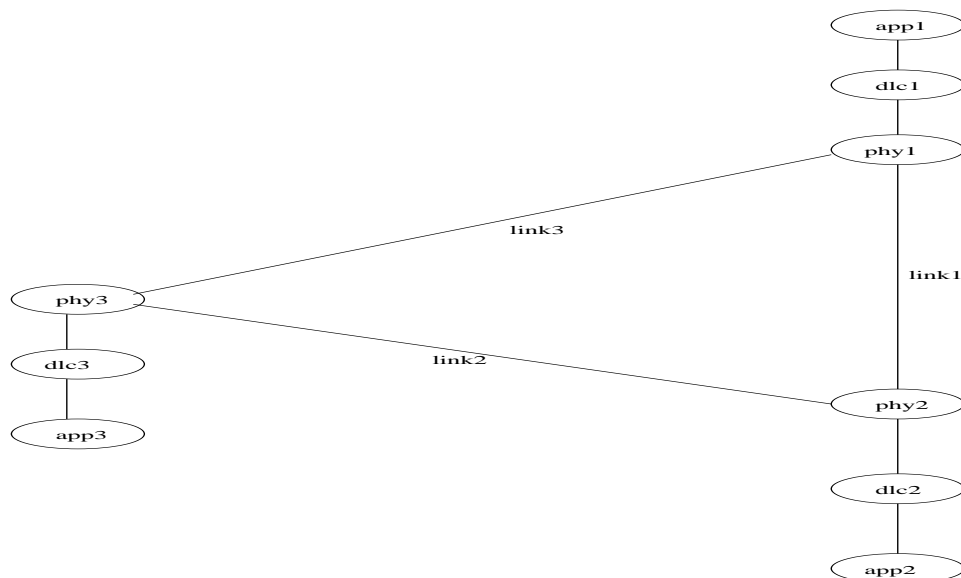


Figure 2: Three Node Configuration

- `3nodes_basic.config`: This is a basic configuration with 3 point-to-point nodes. App1 sends to app2. App2 sends to app3. App3 sends to app1.
 - `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, 0.3 respectively.
 - `3nodes_delay.config`: Link errors are 0, but link propagation delay vary as follows: link1 (app1 to app2) delay = 5 microsecs, link2 (app2 to app3) delay = 50 microsecs, and link3 (app3 to app1) delay = 200 microsecs.
 - `3nodes_speed.config`: Link errors are 0, but link capacities vary as follows: link1 (app1 to app2) Bit Rate = 155.52 Mbps, link2 (app2 to app3) Bit Rate = 77.76 Mbps, and link3 (app3 to app1) Bit Rate = 38.88 Mbps.
- `lab1_demo`: a sample executable file to familiarize you with the graphical user interface.

- `drawgraphs`: a script that will generate graphs that you will need to submit.
2. Copy the above files to your working directory (e.g., `/cis677/lab1/`)
 3. Experiment with `lab1_demo`. See section 6 for instructions on how to run your program.
 4. Study the source code files carefully. (don't worry about the config files).
 5. 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 for `DatalinkToPhysical()` and `DatalinkToApplication()` in `dlc_layer.c`.
 6. To compile your program, type **make**. This will produce an executable called `lab1_exec` in your working directory.
 7. Now execute your version of the code and use the configuration files to make sure it works.
 8. Run the script `drawgraphs`. The script will run your executable and produce several postscript files (one for each configuration). These files will contain graphs of the number of pdu's received by each dlc entity plotted against time. You must submit hard copies of these graphs. *Note*: This program may take a while to run, so please be patient with it.

6 Running your program: The Graphical User Interface

Login to a Unix machine. At your unix prompt, type `lab1_demo`. A window with the simulator interface will open on your terminal. Figure 3 shows the simulator window with the 3-nodes configuration. Each node has 3 layers denoted by three squares. Each link component is denoted by a square between the link connections. In addition to the simulator window, there are 5 smaller windows visible in the figure. Two of the windows are the space time diagrams for the data going between the links (the third space-time window is not visible in the figure but you will see it on the simulator). The other three windows are called the node graphs for the simulation. They allow you to send text data and/or a preloaded graphical image file over the network. The node-graph windows have the following functions:

Function Name	Description
Send msg	Send a message in the text window
Send graph	Send the graph in the graph window
Clear msg	Clear the text window
Clear graph	Clear the graph window
Load graph	Load a pre-defined graph in the graph window

The top of the main simulator window has a menu bar that has the following selections.

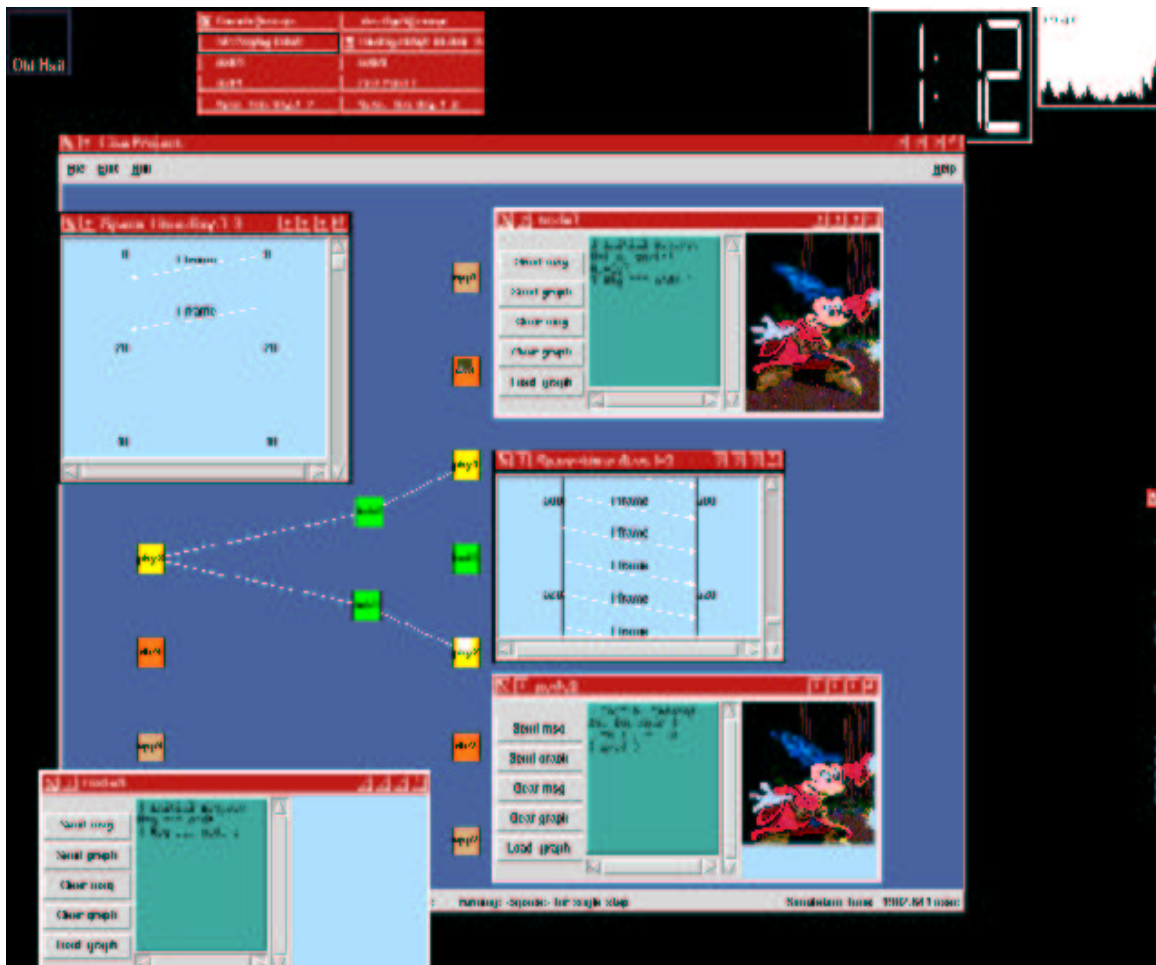


Figure 3: Simulator Interface

Menu	Submenu	Description
File	Load	Load config file
	Exit	Stop the program and exit
Edit	Raise	Raise the node graphs and space-time diagrams to the front of main window. Useful when you cannot see the graphs.
Run	Run	Start the simulator. You need to send msg/graph to see further animation.
	Single Step	Enters single step mode. Use the space bar for a step.
	Pause	Pause the simulation.
	Resume	Resume the simulation.
	Delay	Set delay between events. The default is 50. 5 is pretty fast.
	Stop Time	The simulation time to stop.
Help	Inc/Dec Debug Level	Increment/Decrement Debug Level, which is used in <code>dprintf()</code> .
	About	About the CISE Project.
	How to use	Help text.

The bottom of the main simulator window is the status bar with the following information.

Field	Description
Filename	The configuration file name.
Stop Time	The end time for the simulation
Delay	Delay between events.
DebugLevel	DebugLevel used in dprintf().
Simulation time	The clock in the simulator.

- First, load a config file. This can be done by File/Load or by putting the config file at the command line of this program. Use “Run/Run” to start running.
- To send one or more messages, type in the messages in the text window. Press “Send Msg” button.
- To send a graph, “Load Graph” first and then “Send Graph.” or just press “Send Graph” and the simulator will load the graph and start running.
- Change the delay to make it run faster/slower. Use “Run/Single step” and space bar to see it step by step.
- Change Debug Level (from 0 3) and use `dprintf(inFriday, 26 July.t debug_level, ``format``, variables)` in your program to print out debug information.

7 Submissions

You must submit the following electronically by the command: `submit c677aa lab1 filename`

- Your source code file `dlc_layer.c`.

You must submit hard copies of the following:

- A *short* summary of the lab, including an interpretation of the graphs.
- Your source code.
- The graphs produced by `drawgraphs`.

8 Miscellaneous Notes

- If you have any questions, please first `lab1.faq` for answers to frequently asked questions on the course's web page.
- Also, you may visit the instructor or the grader

–

9 Appendix: `dlc_layer.c`

```
#include "dlc_layer.h"

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

static dlc_layer_receive(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                        GENERIC_LAYER_ENTITY_TYPE *generic_layer_entity,
                        PDU_TYPE *pdu) {
    if (DatalinkFromApplication(generic_layer_entity)) {
        DatalinkToPhysical(dlc_layer_entity, pdu);
    }
}
```

```

    } else if (DatalinkFromPhysical(generic_layer_entity)) {
        DatalinkToApplication(dlc_layer_entity, pdu);
    }
    return 0;
}
/*****

DatalinkToPhysical(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                  PDU_TYPE *pdu_from_application) {

    PDU_TYPE *pdu_to_physical;

    /* just a sanity check */
    if (pdu_from_application->type != TYPE_IS_A_PDU) panic("Empty a_pdu\n");

    /* ----- DO YOUR CODING HERE ----- */

    /* create d_pdu: use pdu_alloc() */
    /* Fill the fields of d_pdu */

    /* send to phy */
    /* Use the macro:
        send_pdu_to_physical_layer(dlc_layer_entity,pdu_to_physical);
    */

    pdu_free(pdu_from_application);
    return 0;
}

/* ----- */
DatalinkToApplication(DLC_LAYER_ENTITY_TYPE *dlc_layer_entity,
                    PDU_TYPE *pdu_from_physical) {
    PDU_TYPE * pdu_to_application; /* use pdu_alloc() to create this */

    /* just a sanity check */
    if (pdu_from_physical->type != TYPE_IS_D_PDU) panic("Empty d_pdu\n");

    /* ----- DO YOUR CODING HERE ----- */

    /* extract a_pdu, check for error and send to app app if error free*/

    /* Use the macro:
        send_pdu_to_application_layer(dlc_layer_entity, pdu_to_application);
    */

    pdu_free(pdu_from_physical);

    return 0;
}

```