CSE 473S Lab Assignment 1

Due date: TBA

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.**

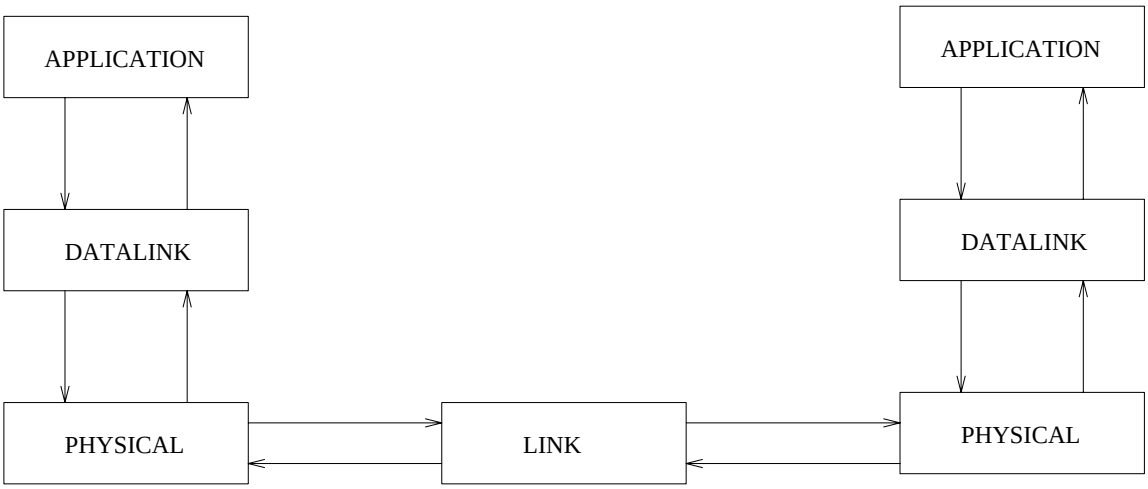

Figure 1: Layered Architecture

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 {                                 /* 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;
```

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, for example, 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 the file at **http://www.cse.wustl.edu/~cse473s/labs/lab1.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 layer.c`: File containing the outline for the lab.
   - `lab1.vcproj`: Visual Studio project file for the lab.
   - `test/3nodes_basic.config`: This file specifies the configuration of the network. In this lab you will only use a 3 node configuration with point-to-point links. This is a basic configuration with 3 point-to-point nodes. App1 sends to App2; App2 sends to App3; App3 sends to App1. The configuration is shown in figure 2 (next page).
   - `test/lab1.exe`: A sample executable file to familiarize you with the graphical user interface.

2) Unzip the `lab1.zip` file into your user directory at CEC. Once unzipped you should have the following folder structure. `H:/cse473s/lab1`. (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_LAB1
     - Variable Value: H:\cse473s\lab1\sim.src
   - Click OK for all open dialogs.
4) Experiment with `lab1.exe`. Refer to section 6 for instructions on how to run your program.

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

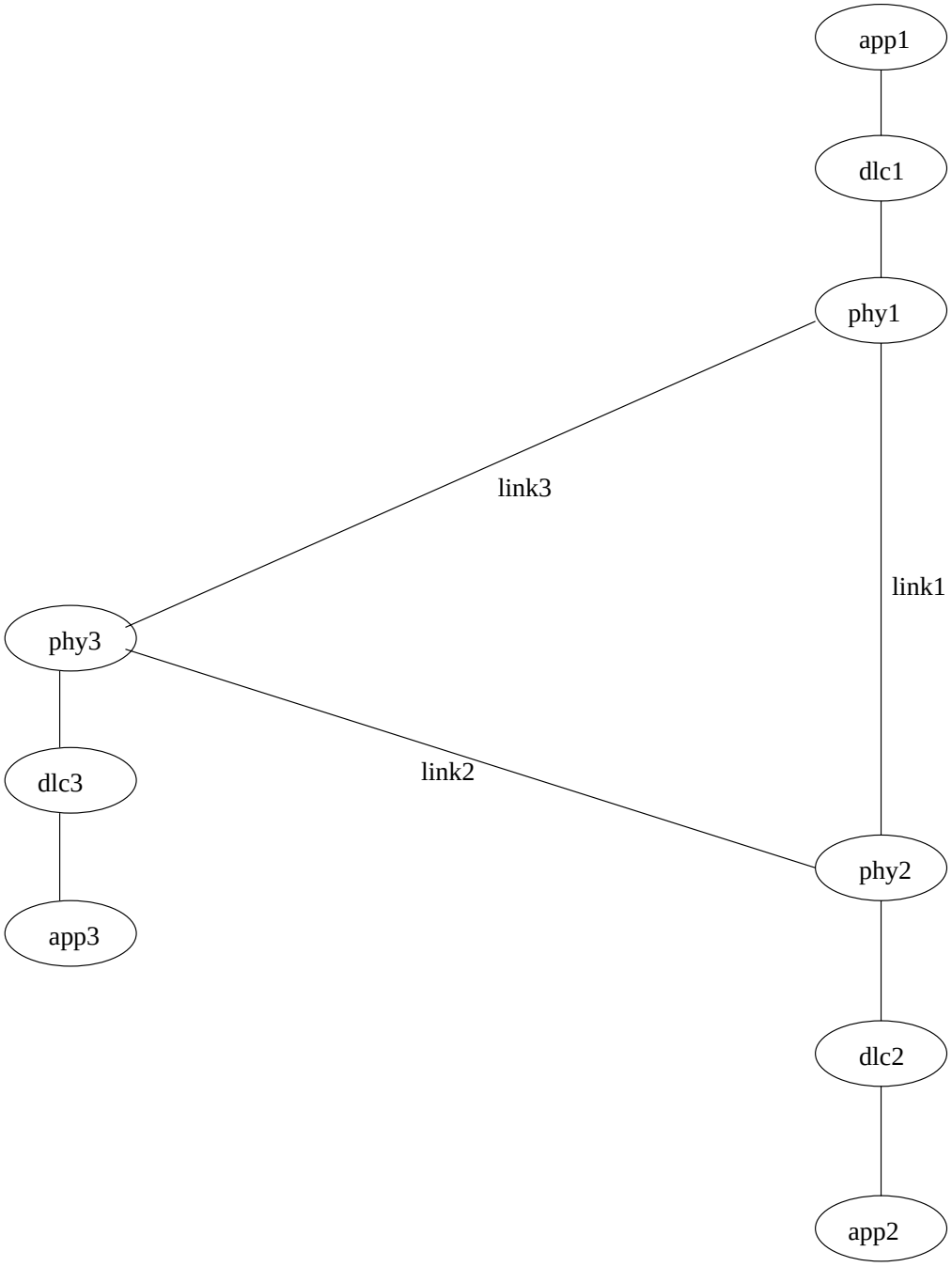

Figure 2: Three node configuration

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 for `DatalinkToPhysical()` and `DatalinkToApplication()` in `dlc_layer.c`.

7) To start, open `lab1.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. Running your program: The Graphical User Interface

When you execute the lab1.exe file (in the test directory) or use the *Debug/Start* menu option from Visual Studio .NET you will see a window with the simulator interface.

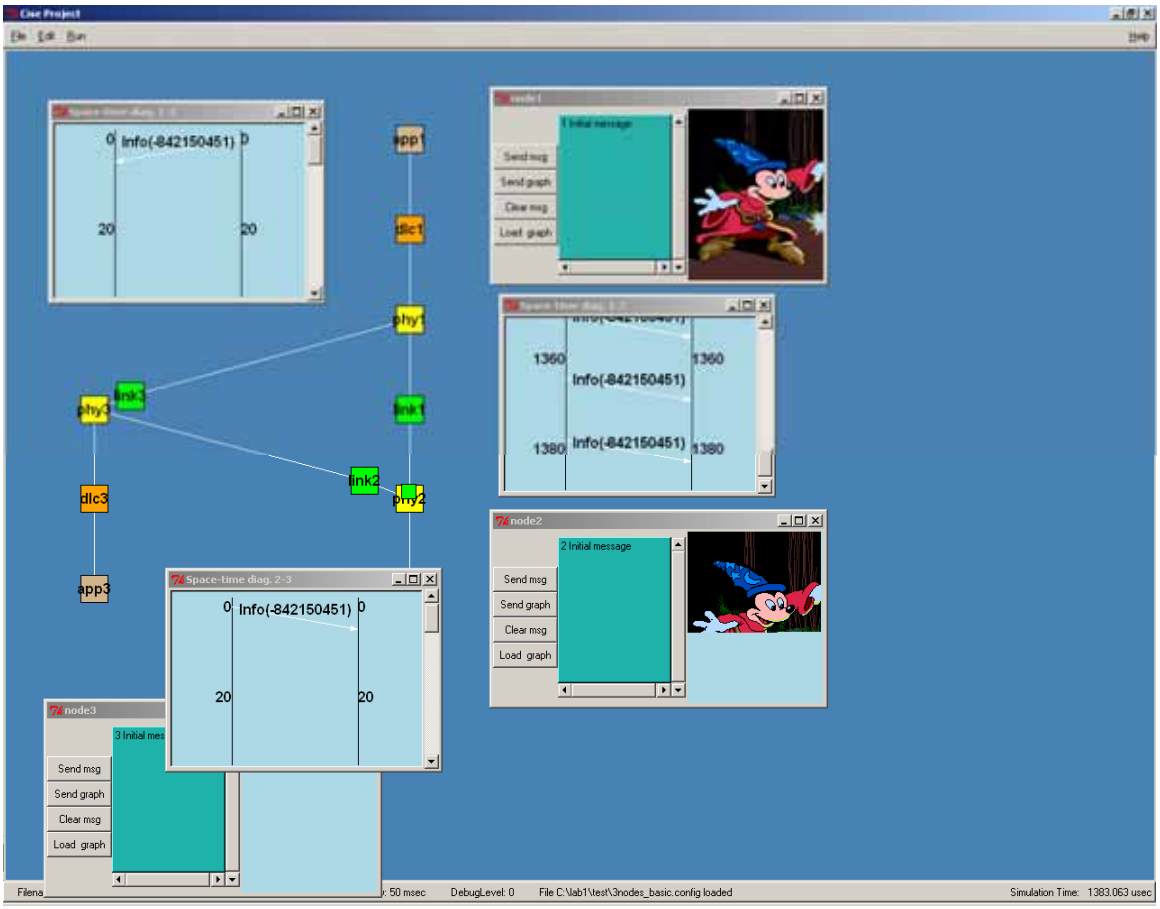

Figure 3: The simulator environment

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.

Menu	Sub Menu	Description
File	Load	Load configuration file (**Include full path**)
	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.
	Inc/Dec Debug Level	Increament/Decrement Debug Level, which is used in `dprintf()`.
	Stop Time	The simulation time to stop.
Help	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	Debug level used in `dprintf()`.
Simulation time	The clock in the simulator.

- First, load the configuration file. This can be done by using the *File/Load* menu option and typing *H:\cse473s\lab1\test\ 3nodes_basic.config* in the input dialog that comes up.
- To send one or more messages, type in the messages in the text window. Press *Send Msg* button.
- To send a graph, press *Load Graph* first and then press *Send Graph*.
- Change the delay to make it run faster/slower. Use *Run/Single step* menu option and the space bar to see it step by step.

- Change Debug Level (from 0 3) and use `dprintf(int debug level, "format", variables)` in your program to print out debug information.

7. 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 "lab1_lastname_firstname. Please also name the zip file as lab1_lastname_firstname.zip (e.g. lab1_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.

8. Miscellaneous

- 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

9. Appendix

```
/* skeleton dlc_layer.c for lab1 */


#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"

/*************************************************/
/* ----- 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; /* use pdu_alloc() to create this */

   /* 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() */
   /* use bcopy to copy the contents */

   /* 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;


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

    /* ------ DO YOUR CODING HERE ----- */
    /* Use the macro:

send_pdu_to_application_layer(dlc_layer_entity,pdu_to_application);
       */

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

    pdu_free(pdu_from_physical);
    return 0;
}
```