Introduction / Purpose

For this project, we will provide you with the framework for a virtual host – including the network stack. This framework provides all aspects of a network-enabled machine. Your job will be to extend this framework by completing the Ethernet layer of the protocol stack.

This project’s goals are twofold:
- to help you familiarize yourself with the project framework so you can best complete the development of subsequent project(s) in this course
- to help you develop a detailed understanding of Ethernet framing by completing the construction of the Ethernet layer of a network stack

Provided in the framework are the major components of a network stack for an emulated Ethernet network. The physical Ethernet connection is provided in the form of an Ethernet emulator – Ethernet over IP (EoIP). A simple Java-based application shell (EoIPShell) is also provided that allows you to exercise the environment via a simple command-line interface.

In this project, you will learn how to 1) implement protocol stack layers and 2) use a protocol stack in a network application. The core of this project is to extend the Ethernet layer which sits directly above EoIP’s physical layer. In later projects, you will extend this framework to include other network layers (e.g., IP, etc.).

Modular, layered programming is key to this project. A nicely-layered, object-oriented framework is provided to you. Take the time to understand it and use it.

This assignment is divided into 4 basic tasks in alignment with the above goals:
1. Familiarize yourself with the supplied framework and write up your observations in your report.
2. Fill in the details of the Ethernet layer.
3. Make changes in the Echo application as necessary.
4. Implement the PGrab application.
5. Use the Echo and PGrab applications to exercise the system.

Background

Framework Overview

The provided environment has two major components: 1) an application shell (called the “EoIPShell”) to translate, interpret and execute user commands and 2) a network stack (called the “NetworkStack”) that takes network requests and services them.

EoIP Shell

When you run your project on the Java interpreter, the application shell prompts you for input with a “eoip >” prompt. If you type “?” at the EoIP prompt, it will print out the available applications.

For this project, an Echo Application is also provided for you. Typing “echo 313337000001 This_is_my_message.” at the EoIP prompt will cause the shell to run the Echo Application, which will:
1. accept the first argument as the destination address (in hex) of the message and the second as the actual message to be sent (in this case we are sending to the Echo Server 313337000001)
2. send it to the network stack,
3. wait for the network stack to deliver a return message,
4. output the message (intended to be the same as the message sent) to the Java console, and then
5. exit.

The “eoip>” prompt will then be displayed again when Echo exits. Typing “exit” to the EoIP prompt causes
the EoIPSHELL to exit.

Consider the following hypothetical exchange. Typed input is **em bodened**.

```
$ java MyProject
EoIP - Ethernet over IP [Version 0.2]
(C) Copyright 2003 University of Tulsa

Authentication : [OK]
MAC Address    : 31-33-37-A2-B3-11

EoIP (? for help) > ?
echo sends a string to the echo server; usage: echo <destination> <message>
grab grabs n number of incoming packets; usage: grab <number of packets to capture>

eoip> echo 313337000001 my_message
Echo Request --> my_message
Echo Reply   --> my_message

eoip> grab 1
/*** Packet [1] Received ***/
  0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5
+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--+--
 0 : AA AA AA AA AA AA 10 10 10 10 10 10 16 44 48 65 ......DHe
16 : 6C 6C 6F 57 6F 72 6C 64 2D 2D 48 65 6C 6C 6F 57 lloWorld--HelloW
32 : 6F 72 6C 64 2D 2D 48 65 6C 6C 6F 57 6F 72 6C 64 orld--HelloWorld
48 : 2D 2D 48 65 6C 6C 6F 57 6F 72 6C 64 2D 2D 48 65 --HelloWorld--He
64 : 6C 6C 6F 57 6F 72 6C 64 2D 2D 48 65 6C 6C 6F 57 lloWorld--HelloW
80 : 6F 72 6C 64 82 AF 04 05 orld....

eoip> exit
$```

**EoIP's NetworkStack**

Also provided is a partial network stack with a complete Physical Interface layer and a shell for the Ethernet
(framing) Interface.

Access to the rest of your computer (i.e., disk, keyboard, console) will be performed through standard interfaces
provided by your Java VM.

Note that all program output – **including error messages** – must be delivered to the user via the console (e.g., via
System.out.println()).

**Framework Details**

**EoIP Description**

A simulated Ethernet network is provided for you in the form of some Java software called EoIP, some of which
runs inside of the Java VM in your computer and some of which runs on a departmental server within the UTulsa
computing environment.

**Echo Server communication**

For “echo” type frames, the Echo Server receives the frame from EoIP, checks the CRC (discards if incorrect),
swaps the source and destination addresses in the frame and sends the new frame back to EoIP, which then
broadcasts it to all other interfaces currently connected to it.
EoIP Shell

The Application Shell is currently composed of MyProject (which extends the super class EoIPShell), EchoApp (a sample application) and EchoSocket (an interface between EchoApp and EchoLayer - the corresponding layer in the network stack). EchoPacket.java is the data object used in the interface between EchoLayer. You will be building the PGrab application using EchoApp as a model.

An additional diagnostic tool is included in the EoIP Machine for your convenience. It is the “Grab” application – GrabApp and GrabSocket. GrabSocket provides an interface between GrabApp and the Physical Interface layer without passing through the Ethernet Layer. The purpose of GrabApp is to provide you with a fully-implemented application that works so that you can ensure that your environment works properly before implementing and testing your code. GrabApp is invoked within EoIP Machine by typing “grab” at the “eoip>” prompt. GrabApp will instruct the Physical Interface to take \( n \) number of frames received in promiscuous mode and delivers them to back to GrabApp. GrabApp will then print diagnostic output on the Java console.

You will need to implement PGrabApp, which is very similar to GrabApp, but has a few differences. More details are provided below.

EoIP Network Stack

The EoIP Network Stack is currently comprised of EchoLayer (the Echo Protocol layer) and EthernetLayer (a partially-completed Ethernet layer). Note that the EoIP Network Stack provides the Physical Layer and the entire emulated Ethernet network, including a hub and various servers as needed. These external classes (as opposed to the EthernetLayer) are supplied only as a Java classes in a JAR file, and not as Java source code. Full JavaDoc API documentation (i.e., in HTML) for these external classes are included in the project startup ZIP archive file.

Send()

For this project, you will send a message from the application through a socket which is used to define a network stack. The Echo Layer will pass the message to the Ethernet Layer. Your Ethernet Layer must provide the proper source, destination, type/length, data and CRC and then deliver it to the Physical Layer.

Deliver()

The Physical Layer will also be continuously receiving frames from the Ethernet segment to which your host is connected (the emulated EoIP network). Since the network includes a simulated hub, the frames appearing at your stack’s Physical Interface will include many types of Ethernet frames:

- frames returned from the Echo Server – echoing what you just sent to it,
- other frames being sent by (and returned to) other students, and
- randomly-generated traffic – some of which have proper framing parameters and some of which have improper framing parameters (e.g., a bad CRC).

It is your job to receive this traffic from the Physical Layer and deliver messages from properly chosen frames to the appropriate layer by implementing the algorithms described in class.

When so instructed by PGrabApp, your EthernetLayer will configure itself into promiscuous mode and deliver an entire Ethernet frame to the PGrabLayer as the delivered message.

Normal Mode

Your Ethernet Layer should normally be in “normal” mode, and only be concerned with frames that have a destination address matching its own MAC address. All other frames are to be discarded.
Promiscuous Mode

When so instructed by PGrabApp, your Ethernet Layer will need to configure itself into “promiscuous” mode and deliver an entire Ethernet frame to the PGrabLayer as the delivered message. (In this case, we want to see the entire frame, not just the message.)

When an Ethernet layer is in promiscuous mode, it will take any and all frames it receives and deliver them up the stack — regardless of the destination MAC address or the validity of the frame. This is what a “sniffer” does, and what you are to do here.

Deliverables:

Tasks 1 and 2 are to be completed by ________________________________.

The deliverables (which are described in the next section) include:

- A hard-copy project report, including
  - Deliverables 1.1 and 1.2 from Task 1.
  - A traditional project report for Task 2, including:
    - description of your solution
    - description of your implementation,
    - challenges encountered and
    - feedback.

You may also include an electronic version of your report if you wish, but the hard copy of the report is still mandatory.

- All Java source files in a ZIP archive sent in email to mauricio-papa@utulsa.edu
  - Subject line – CS4333 Project 2–Fall 2003
  - The application program to be invoked must be named Project2.class

Task Descriptions

Task 1: Framework Familiarization

Familiarize yourself with the Java code handed out in class. The source files are named in a consistent and meaningful manner. This code will also be explained in class. Trace through the code line by line as it invokes module after module.

- Document the entire supplied code base both in written and graphical formats. Written format might be along the lines of “FooSocket passes the PacketObject to BarLayer by calling method foobarSend()”. We suggest using a graphical format similar to the one shown below:

  ![Diagram of network stack layers](Figure 3)

  **Figure 3**

  **Deliverable 1.1:** Document the “send” operation as invoked by EchoApp.
  **Deliverable 1.2:** Document the “deliver” operation as invoked by EchoApp.

Task 2: Complete the Ethernet Layer

Develop the code needed to implement both send and deliver functions for the Ethernet Layer, supporting the higher-level Echo – and later, other – network stack layers.
General Functionality:

EchoApp

To provide EchoApp functionality, your project needs to:

• accept a small text message (from 1 to 255 characters) from args.get(2).
• send the message down the stack to the Ethernet layer
• pad the message if necessary to meet minimum frame size requirements
• send the frame to the “echo server”
• receive the frame returned from the echo server
• verify the destination MAC address, type/length and CRC32
• deliver the message (if valid) back up the stack to EchoApp
• print the message to the console in ASCII

PGrabApp

To provide PGrabApp functionality, your project needs to:

• accept the number of frames to capture from args.get(1):
• configure the Ethernet Layer into promiscuous mode
• receive the captured frame(s) from the Physical interface
• deliver the captured frame(s) as messages back up the stack to PGrabApp
• print the frame contents to the console in hexadecimal. E.g., for n=2 frames,

\[00:f1:33:c9:7f:ea#6e:43:db:52:01:9e#00:f1#33:c9:...:4a:d0#e7:a2:34:55\]


• The “;” separates hex digit pairs. The “#” separates fields – Destination, Source, Type/Length, Data, CRC32. The ellipses “...” are used here represent a long string of data too big to fit on the page.

PGrabApp has a few very specific differences from GrabApp:

1. PGrabApp is invoked with a single parameter – the number of frames to sniff – up to 255. (e.g., “PGrab 255”). It should deliver a printout of that many frames to the Java console.
2. PGrabApp will need a PGrabSocket – much like EchoAppSocket.
3. PGrabSocket will need to instruct EthernetLayer to go into promiscuous mode.

Technical Requirements:

• The Echo Application (and all other applications used in the EoIP Machine) must implement the messagePump() method to handle all necessary invocation commands. Also note that the constructor for each application will need at least two static strings: one for the application name and one for the data. Look at the supplied Java Docs.
• Your Ethernet Layer needs to implement both the send() and deliver() methods.
Your Ethernet Layer must accept the message and send() a frame to the Physical Layer.

The frames sent to the Physical Layer must be IEEE 802.3 (Ethernet) compliant.

Your program must create valid Ethernet frames in all cases. Messages of as few as 1 byte or as much as 1500 bytes may be sent to your program. Note that the Ethernet frame sent to the Physical layer must be padded as necessary to meet standards for minimum frame length.

**Note:**
1. The Echo Protocol's Ethernet type is 5700 (decimal).
2. The echo server's MAC address is 0x313337000001 (31-33-37-00-00-01) (hexadecimal)
3. The CRC32 is typically handled by hardware in the Ethernet interface. For this assignment, you must provide and check the CRC32 in all (i.e., send() and deliver()) communications. You are not allowed to use java.util.zip.CRC32! You will implement this on your own according to the algorithm in the book.
4. Several frames certified to be valid will be provided to you in the ZIP file to manually verify your CRC implementation.
5. In future projects, you will need to send and deliver other protocols with their proper type/length.

The Echo application specifies normal mode frame reception, while the PGrab application specifies promiscuous mode frame reception. The deliver function of the Ethernet layer must be able to function properly in both of these modes.

Normal mode must process all received frames, but only deliver messages included in frames addressed to this interface (indicated by the received frame's destination address) to the next higher level in the network protocol stack.

Promiscuous mode processes and delivers all received frames up the network protocol stack – regardless of destination address.

The nature of the usage of promiscuous mode dictates that the entire frame of data must be delivered – instead of stripping the frame header and trailer as is done for normal mode.
• EthernetLayer’s deliver function must pass the proper Ethernet type for the Echo Protocol (the “demultiplexing key”, also known as the “demux key”) up the stack to the Echo Layer.

• EthernetLayer’s deliver function must check to ensure that all frames to be passed to the higher level layer have valid Ethernet CRCs.
  • Frames with an invalid CRC are considered to be corrupt and shall be dropped quietly (with no special notification, console output, etc.). This applies to all packets, regardless of the destination address contained in the frame.
  • You do not have to worry about handling dropped frames or frame collisions.

• Your stack must follow the general requirements for a protocol stack. Chapter 1 of the text provides a number of additional general guidelines and requirements for protocol stacks.

Testability Requirements:
Design your program with the following guidelines in mind:
• Your Java source code will be compiled and tested on Linux under Java 1.4
• All testing is done via automated script. Your program must work properly in a scripted environment.

Rules of Conduct
Some very important rules governing use of EoIP:
• You need to hard-code your University of Tulsa student ID number in your EthernetLayer class and pass the value via the EoIPShell extension (MyProject). See the example code for how this is done. EoIP will only respond to frames associated with valid student ID numbers (from the class list). Frames without a valid student ID will end up in EoIP’s “bit bucket”.
• Your student ID will be used only as a seed to an encryption algorithm and will not be visible to anyone in any transmissions.
• Do not send any vulgar, obscene or inappropriate messages, as these messages may be seen by other students doing their own EoIP development. The EoIP server logs all frames sent from EoIP clients, and since they are all identified by your (encrypted) student ID number, we know who you are, when you send messages and what those messages are. Just be polite.
• Do not “attack” the EoIP Hub or Echo Server with a large number of packets. Don’t even consider doing this. This kind of behavior – which would be disruptive to the efforts of other students and violate academic policy) – will not be tolerated.

EoIP Development Environment Requirements:
• EoIP was developed under and requires you to use Java2 v1.4 for your development.
• The EoIP package is contained in a Java Archive (jar) file, which contains the Java classes you need. Leave the jar file alone. Do not unzip it. Specify this package in your java code with an “import” declaration statement just like you would import any other package:

```java
import edu.utulsa.eoip.*;
```

• The EoIP package requires you to specify the jar file in your CLASSPATH environment variable when you compile your Java code and run your Java class. Put the jar file in the same directory as your source and class files. Do not extract the classes themselves, as this would only complicate things.

1 A demultiplexing key is only needed on delivery from the Ethernet Layer to the next higher layer. The other layer interfaces (Physical Layer > Ethernet Layer, Echo Layer > Echo Socket and PGrabLayer > PGrabSocket) are 1:1 interfaces and do not need any such identifying information.

2 A jar file is actually a regular “ZIP” archive of Java class byte code.
On Windows, your Java SDK (Software Development Kit) and JRE (Java Runtime Environment) will take care of this all for you automatically.

If you are compiling from a DOS (cmd) prompt, use these commands:

```
javac -classpath .;EoIP.jar MyJavaProgram.java

and then

java -classpath .;EoIP.jar MyJavaProgram
```

If you are compiling from the Linux or UNIX command line, you have two choices of how to add the jar file to your CLASSPATH:

- One time change (e.g., in your `.profile`, `.bash_profile` or whatever file):
  ```
  export CLASSPATH=${CLASSPATH}:.:EoIP.jar
  ```
  or

- One that you must do explicitly each and every time you compile or run:
  ```
  javac -classpath .:EoIP.jar MyProject.java
  and then
  java -classpath .:EoIP.jar MyProject
  ```

**External Resources:**

- This Document along with the latest version of EoIP can always be found at: [http://www.mcs.utulsa.edu/~papama/cs4333/eoip](http://www.mcs.utulsa.edu/~papama/cs4333/eoip)
- Please report all bugs in programs or documentation to: eoip@utulsa.edu