Practical Exercises in Computer Networks

Study of basic PDU encapsulation and multiplexing with Wireshark (WIP)

In this lab we extend the capabilities of the Java WoI program that we developed in the previous practical by retrieving the parameters of the sending interface and by visualizing them by using Wireshark

Introduction

Java offers an API known as Java Network Interface that provides access to the system network interfaces, thus allowing the programmer to choose, in a programmatic way, the most convenient network interface for some applications, particularly those related to Muticasting (Recall Unicast/Broadcast/Multicast modes of communication in the Internet that we introduced in Lab 1). Also, some applications require the programmer to select an interface that fulfills some conditions such us MTU (Maximum Transfer Unit, please, skim Ch.3 of PD's textbook for a more detailed explanation of this important networking concept), whether the interface is loopback, real or virtual, etc.

In order for us to understand the basic concepts of the Java Network Interface API we will write a program that retrieves the full list of system network interfaces and then prints out the parameters of the Ethernets (IP addresses applied, the MAC address, the MTU, whether the interface is Up).

1. **Exercise.** Download the NifEnumerator_base.java program from http://paloalto.unileon.es/cn/NifEnumerator_base.java

    a. Compile and run the program from the program line (Make sure you install your program class into a directory named after its package name and then you invoke your class by using the fully qualified class name from its parent directory, or set the CLASSPATH variable accordingly)
    b. Contrast the results with those obtained by executing /sbin/ifconfig, is there something that calls up your attention?
    c. The program provided does not print out the MAC address of each interface, then, complete method named static void printOutMacAddressAndMTU(NetworkInterface netIf) so that it prints the MAC address exactly as /sbin/ifconfig does (Hexadecimal separated by colons, you can use java.io.printf() with the write format).

Extension of the Java WoI program

The Java WoI program that we wrote in the practical on Mac-related programming did not select a specific Network Interface to send the WoI packet over, now we want to be able to select the output interface based

on its IP address and calculate the corresponding network's Broadcast Address without having the user provide it (Recall that we had to provide the Broadcast address as a command-line argument to the Wol java program).

2. **Exercise.** Write a new version of the wol.JavaWol program that we tested on our previous practical, the new version will print out the list of network interfaces along with an identifying index and, then ask the user for the IP address interface's index he wants the Wol packet be sent over. Therefore, you will have to integrate the knowledge that you acquired from both programs, JavaWol.java and NifEnumerator_base.java. You may have to refactor both of them substantially, therefore, proceed carefully. Name the new wol program WolNifSelect.java and use the package name as before (wol):
    a. The Magic packet is sent to the broadcast address of the LAN to which the target host is connected to. Depending on the JVM version used and its default options, the DatagramSocket created will not accept broadcast traffic, in which case we will have to invoke DatagramSocket's setBroadcast() method. The original version of Wol created a DatagramSocket object but did not check whether broadcast traffic was allowed, in your new version of the program, make sure that the DatagramSocket object accepts broadcast traffic.
    b. Have WolNifSelect.java ask the user to enter the Nif's index over which to send the wol packet.
    c. Is it possible to calculate the Broadcast Address to which the wol is sent, or the user must necessarily provide it? Otherwise, can you obtain the subnet mask applied to a Network Interface?

3. **Exercise**. Now, test your WolNifSelect.java using Wireshark, use the following steps as a guide:
    a. Start Wireshark (See in Internet applications in your Linux desktop)
    b. Activate the Capture on the Ethernet interface (Don't specify any capture filter)
    c. Start the capture and specify a "wol" filter in the display filter textbox
    d. Send the wol from the command line:
    $ java wol.WolNifSelect
    Interact with your program and indicate it that it must use the Ethernet interface
    Check that the target computer gets powered up
    e. Select the Magic Packet on the Wireshark frame capture listing
        i. Unfold the Ethernet II frame and its most important fields: Source MAC, Destination MAC and "protocol"
        ii. Where is the error checking sequence of this frame (CRC , Cyclic Redundancy Check)? How many bits it takes in the frame?

4. **Exercise**. In this exercise we want to use Wireshark to display the Multiplexing/Encapsulation process as an information sent by an application passes through the hierarchy levels top-to-bottom. We are going to connect to a web server and fetch its root document.

    a. Start Wireshark on the Ethernet interface of your client host and specify a display filter "http"
    b. Start Wireshark on the web server that we are going to access (Login into protocol.unileon.es as estudiante) and apply the same "http" filter as above
    c. Use the wget command to fetch protocol.unileon.es's root document:
    $ wget procotol.unileon.es/index.html

d. Stop the captures on both the client and on the server
e. Study the multiplexing/encapsulation process that takes place in the client: Start with the application layer whose protocol, in this case, is the http protocol; proceed downwards the hierarchy through the datalink layer, at each layer, observe the payload received from the upper layer and the header added by the present layer (Encapsulation). What does multiplexing mean in this context? Document this exercise so that you can later compare it with the results obtained at the server (Observe fig. 1 example)
f. Now, study the inverse process at the server: begin with the datalink layer and observe its header and payload and how that payload will be deencapsulated/demultiplexed and delivered to the upper layer protocol indicated in the demultiplexing key at the present layer's header. Document the results conveniently and compare them with those at the client; check that everything in consistent at the client and the server.

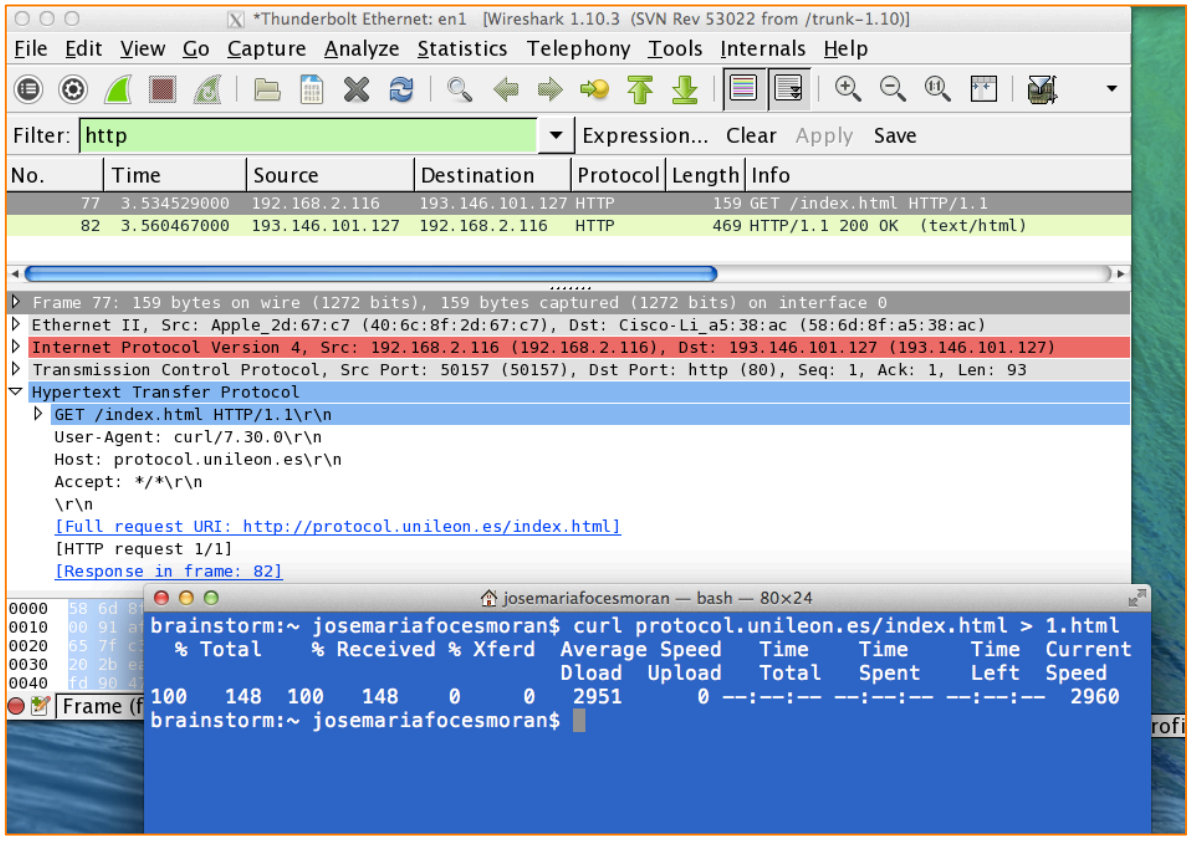

Fig. 1. Capture of a client http request from host 192.168.2.116 to web server host at IP 193.146.101.127

5. **Exercise.** In exercise no. 3 of the previous practical (Ethernet MAC programming in Java) we studied the significance of the integer argument constant 9 in the invocation of the constructor DatagramPacket(data, SIZE, address, 9), what is the UDP service it corresponds to? Consult your the internet well-known port in your system's services database (/etc/services).

6. The 48-bit Ethernet addresses are specified by IEEE (Institute of Electrical and Electronics Engineers), search the Internet and find the IEEE standard that specifies MAC addresses.

7. Is it possible to access the Service Interface of layer 3 and layer 2 in a Linux or BSD system? Search the internet to see if you can find how to programmatically access the Ethernet and the IP Service Interfaces; provide a brief explanation of your conclusions. Is that possible in Java, or one msut switch to a language like C or C++?

Appendix 1: Source Code

```
/**
 * ********************************************************************************
 * Universidad de León, EIII Grado en Ingeniería Informática
 * Course on Computer Networks (C) 2013-14 José María Foces Morán, lecturer
 *
 * CN LAB on encapsulation/multiplexing: Enumerate Network Interfaces, print
 * IP addresses, the MAC address and the interface's MTU
 *
 * NifENumerator_base.java
 *
 * ********************************************************************************
 */

package nif;

import java.net.*;

import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.*;

public class NIfEnumerator_base {

    static private Enumeration<NetworkInterface> allIfs;

    ArrayList<NetworkInterface> getEthernetIfList() throws SocketException {

        ArrayList<NetworkInterface> al = new ArrayList<NetworkInterface>();

        for (NetworkInterface netIf : Collections.list(allIfs)) {

            System.out.println(netIf.getDisplayName());
            if (isEthernet(netIf)) {
                al.add(netIf);
            }

        }

        return al;
    }

    /*
     * Assess whether an interface name identifies an ethernet (true)
     * or not (false)
     */
    private boolean isEthernet(NetworkInterface netIf) {

        /* Compile a regular expression for ethernet such as:
         * eth0, e12, en3
         */
        Pattern pt = Pattern.compile("e\\d|en\\d|eth\\d");
        /*
```

```
     * See whether the string that represents the name of an interface
     * matches the previous regular expression
     */
    Matcher mc = pt.matcher(netIf.getDisplayName());

    return mc.matches(); // true or false
}

/*
 * Print all the ip addresses assigned to a network interface
 */
static void printOutIpAddresses(NetworkInterface netIf) {

    Enumeration<InetAddress> inetAddresses = netIf.getInetAddresses();

    while (inetAddresses.hasMoreElements()) {
        InetAddress ip = inetAddresses.nextElement();
        System.out.println("\t" + ip);
    }

}

/*
 * Print the MAC addresses assigned to network interface netIf
 */
static void printOutMacAddressAndMTU(NetworkInterface netIf) {

    System.out.printf("MAC Address in hex base (Colon separated): ");

}

/*
 * Constructor for the class: Collects a list of all the network
 * interfaces installed in this system
 */
NIfEnumerator_base() {

    System.out.println("All system network interfaces:");

    try {
        allIfs = NetworkInterface.getNetworkInterfaces();
    } catch (SocketException ex) {
        Logger.getLogger(NIfEnumerator_base.class.getName()).log(Level.SEVERE, null, ex);
        System.exit(-1);
    }

    System.out.println();
}

/*
 * Test program: Lists the ethernets installed alongside with their
 * state (up/down) and their assigned ip addresses, these addresses will be
 * at least two per interface, one IPv4 and the other one IPv6.
 */
public static void main(String args[]) throws SocketException {
```

```
        NIfEnumerator_base nie = new NIfEnumerator_base();

        ArrayList<NetworkInterface> ethers = nie.getEthernetIfList();

        System.out.println("Ethernets installed in this system(" + ethers.size() + "):");

        Iterator it = ethers.iterator();

        while (it.hasNext()) {
            NetworkInterface eth = (NetworkInterface) it.next();
            System.out.print(eth.getDisplayName() + ": Interface is ");
            if (eth.isUp()) {
                System.out.println("up" + ", IP addresses assigned:");
                printOutIpAddresses(eth);
                printOutMacAddressAndMTU(eth);
            } else {
                System.out.println("down");
            }


        }

    }
}
```