Practical Exercises in Computer Networks

A linkloop application written in C against libpcap (WIP)

This practical illustrates how to send in libpcap and how to filter received traffic. The application we aim to build consists of a client that sends a control string "Hello world\n" and receives echo from the server. This application serves for checking whether a host has local, link-layer connectivity with another host belonging to the same network.

Sending in libpcap

We finished the introductory practical on libpcap with a question about the sending capability of libpcap that we want develop now. Libpcap versions of today offer a sending function whose name is pcap_inject(). Follow the instructions below to download, compile and test a simple C/S application that illustrates the basic sending capabilities of libpcap:

1. Download client: `$ wget paloalto.unileon.es/cn/labs/datalink/pcapEcho.zip`
2. Unzip the compressed file which will yield the following three C-source files:
   `pcapSendHelloWorld.c`
   `receiveHello.c`
   `receiveHelloPcapFilter.c`

3. Study the sender program, attentively review the code that calls pcap_inject() and the structure of the sent Ethernet frame. Edit the sent frame so that the destination and source MAC addresses are correct (The destination MAC must be the MAC address of the receiving host's NIC and the source MAC must be your selected NIC's MAC address). Also, observe that the selected Ethertype value used in the program is 0x07ff which, according to the IEEE 802.3 numbers hasn't been allocated to any protocol.

4. The receiver program clearly resembles the libpcap test programs that we wrote on our past practical, however, a noticeable change is that receiveHello.c actually selects the frames it prints out, considering only those frames whose Ethertype value is equal to 0x07ff. Observe that the callback function (getNewFrame()) actually receives all the incoming traffic, *i.e.,* the NIC driver calls the callback function every time it receives a new frame, which consumes a lot of CPU and I/O resources. Later, in this practical we'll introduce a more sophisticated technique to have our program select traffic without incurring such a costly resource consumption.

5. Compile the first two programs:

   ```
   $ gcc –o pcapSendHelloWorld pcapSendHelloWorld.c -lpcap

   $ gcc –o receiveHello receiveHello.c -lpcap
   ```

6. Request an additional terminal (Shell) from your system

7. Run the receiver first in one terminal window
   ```
   $ ./receiveHello <net device> <n frames to capture>
   ```

8. Run sender in the other terminal window
   ```
   $ ./pcapSendHelloWorld <net device>
   ```

9. Observe the behavior of both programs and whether the receiving program received all the Hello World messages from the sender.

10. This time, you ran the sender and the receiver on the same host, now we ask you to run the receiver on a different host (Recall to edit the receiver's NIC MAC address in the Ethernet frame you are sending). Check that the sent messages are successfully received by the receiver.

receiveHello.c

```
/*****
 Receive HELLO WORLD from Ethertype 0x07ff, filtering traffic in the
 * callback function (i.e., not using a pcap filter)

 *****/

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <pcap.h>

unsigned int frameCount = 0;

/*
 * EtherType
 * Frame bytes ( 12 and 13 )
 */
void printEthertype(u_char *frame) {

    printf("Ethertype: HEX: %02x%02x, ", frame[12], frame[13]);
    printf("Decimal: %hu\n", ntohs(*(uint16_t *) (&(frame[12]))));

    fflush(stdout);

}

/*
 * Print the MAC addresses
 */
void printMac(u_char *frame) {

    int i;

    printf("Destination MAC:\t");

    /*
     * First six bytes of frame ( 0 - 5 )
     * DESTINATION MAC ADDRESS
     */
    for (i = 0; i < 6; i++) {
        printf("%02x:", frame[i]);
    }

    printf("\nSource MAC:\t");

    /*
     * Ensuing six bytes of frame ( 6 - 11 )
     * SOURCE MAC ADDRESS
     */
    for (; i < 12; i++) {
```

```
        if (i != 11)
            printf("%02x:", frame[i]);
        else
            printf("%02x", frame[i]);
    }

    printf("\n");
    fflush(stdout);
}

/*
 * pcap_pkthdr: Generic per-packet information, as supplied by libpcap:
 *      packet lengths and the packet timestamp
 *
 * u_char* frame: Bytes of data from the frame itself with all
 *                the wire bits stripped
 */
void printFrameHeader(u_char* frame, unsigned frameLength) {

    printf("\n\n");
    printf("---------------------------------------\n");
    printf("FRAME# %u\n", frameCount);
    printf("Off-wire frame length: %u\n", frameLength);
    printMac(frame);
    printEthertype(frame);
    printf("---------------------------------------\n");

}

void printPayload(char *payload, int frameLength) {

    //                                  DST MAC(6) + SRC MAC(6) + Ethertype(2)
    unsigned payLength = frameLength - (6 + 6 + 2);

    int i;

    printf("Payload:\n");

    for(i = 0; i < payLength; i++){
        printf("%c", payload[i]);
    }

    printf("\n\n");
    fflush(stdout);
}

/*
 * Callback function specified into  pcap_loop(...)
 * This callback will capture 1 frame whose header is available in frameHeader
 * The frame itself is stored into frame
 */
void getNewFrame(u_char *dummy, const struct pcap_pkthdr *frameHeader, u_char
*frame) {

    /****
     From pcap.h:

     struct pcap_pkthdr {
```

```
        struct timeval ts;    // time stamp
        bpf_u_int32 caplen;     // length of portion present
        bpf_u_int32 len;      // length this packet (off wire)
     };

     ****/

    unsigned int frameLength = frameHeader->len; //Off-wire length of this
frame
    unsigned int caplen = frameHeader->caplen;

    //Ethertype sought for is: 0x07ff
    if (frame[12] == 0x07 && frame[13] == 0xff) {

        frameCount++; //Counts only frames of Ethertype = 0x07ff

        printf("---> frameHeader->caplen = %u\n", frameHeader->caplen);
        /*
         * Print the frame just captured
         */
        printFrameHeader(frame, frameLength);
        printPayload(&frame[14], frameLength);

    }

    fflush(stdout);

}

/*
 * printIPandMask(char *defaultDev)
 *
 * Prints the IP address and the Network mask configured into the network
 * device whose p_cap name is into defatultDevice
 *
 */
void printIPandMask(char *defaultDev) {
    bpf_u_int32 netAddress;
    bpf_u_int32 netMask;
    struct in_addr inAddress;
    char errbuf[PCAP_ERRBUF_SIZE];

    printf("Network device name = %s\n", defaultDev);

    /*
     * pcap_lookupnet() returns the IP and the netmask of the passed device
     * Actual parameters netAddress and netMask are passed by reference since
     * we want them to hold the IP and the netmask, they are therefore output
     * parameters
     */
    if (pcap_lookupnet(defaultDev, &netAddress, &netMask, errbuf) == -1) {
        printf("%s\n", errbuf);
        exit(3);
    }

    /*
     * inet_ntoa() turns a "binary network address into an ascii string"
     */
```

```
    inAddress.s_addr = netAddress;
    char *ip;

    if ((ip = inet_ntoa(inAddress)) == NULL) {
        perror("inet_ntoa");
        exit(4);
    }

    printf("IP address = %s\n", ip);

    inAddress.s_addr = netMask;
    char *mask = inet_ntoa(inAddress);

    if (mask == NULL) {
        perror("inet_ntoa");
        exit(5);
    }

    printf("Network mask = %s\n", mask);
    fflush(stdout);
}

unsigned int performCapture(char* netDevice, unsigned int nFramesToCapture) {

    char errbuf[PCAP_ERRBUF_SIZE]; //The pcap error string buffer

    /*
     * Printout of IP address + Net mask
     */
    printIPandMask(netDevice);

    /*
     * Open network device for capturing frames not-in-promiscuous mode:
     *
     * pcap_t *pcap_open_live(
     * const char *device,
     * int snaplen,
     * int promisc,
     * int timeout_ms,
     * char *errbuf);
     *
     * On OS-X timeout_ms must be > 0 ms
     *
     */
    pcap_t* pcapStatus;
    pcapStatus = pcap_open_live(netDevice, BUFSIZ, 0, 1, errbuf);


    if (pcapStatus == (pcap_t *) NULL) {
        printf("Call to pcap_open_live() returned error: %s\n", errbuf);
        exit(4);
    }

    printf("\n\nCapturing %u frames:\n", nFramesToCapture);
    fflush(stdout);

    /*
     * int pcap_loop(
```

```
     * pcap_t *status,
     * int number_of_frames_to_capture,
     * pcap_handler callback_function,
     * u_char *user
     * )
     *
     */
    pcap_loop(pcapStatus, nFramesToCapture, (pcap_handler) getNewFrame,
(u_char *) NULL);

    return nFramesToCapture;
}

int main(int argc, char *args[]) {
    /*
     * Process command line arguments:
     * get the number of frames to capture
     */
    if (argc != 3) {
        printf("%s <net device> <n_frames_to_capture> \n", args[0]);
        exit(-1);
    }

    performCapture(args[1], atoi(args[2]));

    printf("\n\nFinished. %u frames captured from %s with concrete
Ethertype.\n", frameCount, args[1]);
```

pcapSendHelloWorld.c

```
/*
 * pcapSendHelloWorld.c
 * Send a Hello World message to an Ethernet host
 *
 * Courses on Computer Networks and Distributed Systems
 * (C) 2017 José María Foces Morán
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <errno.h>

#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>

#include <pcap.h>

/*
 * printIPandMask(char *defaultDev)
 *
 * Prints the IP address and the Network mask configured into the network
 * device whose p_cap name is into defatultDevice
 *
 */
void printIPandMask(char *defaultDev) {
    bpf_u_int32 netAddress;
    bpf_u_int32 netMask;
    struct in_addr inAddress;
    char errbuf[PCAP_ERRBUF_SIZE];

    printf("Network device name = %s\n", defaultDev);

    /*
     * pcap_lookupnet() returns the IP and the netmask of the passed device
     * Actual parameters netAddress and netMask are passed by reference since
     * we want them to hold the IP and the netmask, they are therefore output
     * parameters
     */
    if (pcap_lookupnet(defaultDev, &netAddress, &netMask, errbuf) == -1) {
        printf("%s\n", errbuf);
        exit(3);
    }

    /*
     * inet_ntoa() turns a "binary network address into an ascii string"
     */
    inAddress.s_addr = netAddress;
    char *ip;

    if ((ip = inet_ntoa(inAddress)) == NULL) {
        perror("inet_ntoa");
        exit(4);
    }
```

```
    printf("IP address = %s\n", ip);

    inAddress.s_addr = netMask;
    char *mask = inet_ntoa(inAddress);

    if (mask == NULL) {
        perror("inet_ntoa");
        exit(5);
    }

    printf("Network mask = %s\n", mask);
    fflush(stdout);
}

void performInjection(char* netDevice) {

    //The pcap error string buffer
    char errbuf[PCAP_ERRBUF_SIZE];

    /*
     * Open network device for capturing frames not-in-promiscuous mode:
     *
     * pcap_t *pcap_open_live(
     * const char *device,
     * int snaplen,
     * int promisc,
     * int timeout_ms,
     * char *errbuf);
     *
     */
    pcap_t  *pcapStatus;
    pcapStatus = pcap_open_live(netDevice, BUFSIZ, 0, 10, errbuf);

    if (pcapStatus == (pcap_t *) NULL) {
        printf("Call to pcap_open_live() returned error: %s\n", errbuf);
        exit(4);
    }

    unsigned char Ethernet_Hello[] ={
        // MACS
        // iMac home en0:      0x00, 0x1b, 0x63, 0xb5, 0xc1, 0x30,
        // MacBook Air en2:    0x40, 0x6c, 0x8f, 0x2d, 0x67, 0xc7,


        // Dest MAC field
        0x00, 0x1b, 0x63, 0xb5, 0xc1, 0x30,

        // Src MAC field
        0x40, 0x6c, 0x8f, 0x2d, 0x67, 0xc7,

        // EtherType field
        0x07,0xff,

        // Payload starts here:
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
```

```
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',
        'H', 'e', 'l', 'l', 'o', ' ', 'w', 'o', 'r', 'l', 'd', '\n',

    };

    printf("\n\nSending frame:\n");
    fflush(stdout);

    if (pcap_inject(pcapStatus, Ethernet_Hello, sizeof(Ethernet_Hello)) == -1)
{
        pcap_perror(pcapStatus, 0);
        pcap_close(pcapStatus);
        exit(1);
    }

}

int main(int argc, char *args[]) {

    if (argc != 2) {
        printf("%s <net device>\n", args[0]);
        exit(-1);
    }

    /*
     * Printout IP address + Net mask
     */
    printIPandMask(args[1]);

    /*
     * Send frame onto net device name in argument
     */
    performInjection(args[1]);

}
```