ECE435: Network Engineering – Homework 4
DNS, UDP

Due: Friday, 26 February 2021, 5:00pm

This homework has two parts.

For the first part, short answers, create a document with your answers (text, pdf, libreoffice, MS Office if you must) and e-mail them to *vincent.weaver@maine.edu* by the homework deadline. Title your e-mail "ECE435 Homework 4" and be sure your name is included in the document.

For the second part, the coding, finish as described and then run `make submit` to create the submission tarfile and attach that to the same e-mail that includes your short answers document.

1. **DNS**

   (a) Look up the domain registration info for the **maine.edu** domain. There are various ways to do this; on Linux you can use the `whois` utility: `whois maine.edu`
   (you might need to install it first, `apt-get install whois`)

      i. When was the maine.edu domain *first* created?

      ii. What is the name of the domain registrar that maine.edu uses? This is the top-level registry that holds the info for the top-level `.edu` domain.

   (b) Use DNS requests to look up some information on various domains. On Linux you can use a utility named `dig` to do this easily. You might need to install the dnsutils package first `apt-get install dnsutils`. In the examples replace HOSTNAME with the name of the system you are asking about.

      i. What is the IP address of weaver.eece.maine.edu?
      `dig HOSTNAME A`, look for answer in the ANSWER section.

      ii. What is the IPv6 address of google.com?
      `dig HOSTNAME AAAA`

      iii. What is the name of the UMaine nameservers?
      `dig HOSTNAME NS`

      iv. What is the name of the UMaine mailservers?
      `dig HOSTNAME MX`

2. **UDP**

(a) The `tcpdump` program can record network packets. The following packet was gathered using the command `sudo tcpdump udp -XX -i eth0`.

The first lines show a summary of the packet. The rest is a hexdump of the packet. The leftmost column is the offset in hex. The next 8 columns are the hex representation of the bytes. The far right is the contents of the packet in ASCII (unprintable characters are shown as '.').

```
22:20:59.106555 IP macbook-air.43424 >
google-public-dns-a.google.com.domain: 57673+ A? www.adafruit.com. (34)
0x0000:  0013 3b10 667f 0050 b647 1cde 0800 4500  ..;.f..P.G....E.
0x0010:  003e e1ea 4000 4011 7fe6 c0a8 0826 0808  .>..@.@......&..
0x0020:  0808 a9a0 0035 002a 9299 e149 0100 0001  .....5.*...I....
0x0030:  0000 0000 0000 0377 7777 0861 6461 6672  .......www.adafr
0x0040:  7569 7403 636f 6d00 0001 0001            uit.com.....
```

The first part of the packet includes Ethernet and IPv4 headers that we don't know about yet. The UDP fields start at offset 0x22:

```
0x0020:       a9a0 0035 002a 9299 e149 0100 0001  .....5.*...I....
0x0030:  0000 0000 0000 0377 7777 0861 6461 6672  .......www.adafr
0x0040:  7569 7403 636f 6d00 0001 0001            uit.com.....
```

i. What is the source port (in decimal)?
ii. What is the destination port (in decimal)?
iii. What is the size of the UDP packet (in decimal)?
iv. Are checksums enabled? How can you tell?
v. What type of protocol is this / what is the packet doing? (note, the answer to this question is *not* UDP)

(b) What is one reason to use UDP over TCP?

3. **UDP Client/Server Coding**

(a) Download the code from:
`http://web.eece.maine.edu/~vweaver/classes/ece435/ece435_hw4_code.tar.gz`

(b) Unpack the files:
`tar -xzvf ece435_hw4_code.tar.gz`

(c) Build the C files:
```
cd ece435_hw4_code
make
```

(d) Try running the `udp_client` and `udp_server`.

- The client sends a UDP message you type to the server over UDP (DGRAM socket).
- The client then waits for a response from the server, and if it gets none within 5 seconds it gives up and prompts for another message.
  It uses the `select()` system call to wait for data with timeout.
- The server receives the incoming UDP packet from the client using the `read()` system call and prints it to the screen. Since UDP is connectionless, it cannot reply using `write`!

(e) Modify the `udp_server` code to use the `recvfrom()` system call instead of `read()`.

i. A `recvfrom()` call looks something like below. The call receives the IP address and port number as part of the sockaddr structure which can be used to send a reply back to the client.

```
n = recvfrom(socket_fd,buffer,(BUFFER_SIZE-1),0,
                        (struct sockaddr *) &client_addr,
                        &client_len);

// structure definition of struct sockaddr * for reference
//struct sockaddr_in {
//    short           sin_family;   // e.g. AF_INET, AF_INET6
//    unsigned short  sin_port;     // port (remember in network order)
//    struct in_addr  sin_addr;     // struct holding the address
//    char            sin_zero[8];  // zero padding
//};
```

ii. Have the server print out the host and port of the incoming connection. (You can use the following code to get a string version of the hostname and address).

```
hostp = gethostbyaddr(
                (const char *)&client_addr.sin_addr.s_addr,
                sizeof(client_addr.sin_addr.s_addr), AF_INET);
hostaddrp = inet_ntoa(client_addr.sin_addr);
```

iii. As with HW#2, uppercase the message before sending it back.

iv. You can use `sendto()` to send a response. `client_addr` will already be set from the incoming `recvfrom()` call.

```
n = sendto(socket_fd,buffer,strlen(buffer),0,
                (struct sockaddr *)&client_addr, client_len);
```

(f) When done, use `make submit` to create the submission tarball and attach it when you submit the overall homework assignment.