Please type your homework. We will discuss each problem in class but you must do your own work for submission. The date on which each problem is discussed is indicated at the beginning of the problem.

Homework

1. (9/20, 20 points) Tracing DNS with Wireshark

   Procedures
   1. Use `ipconfig` to empty the DNS cache in your host (`ipconfig /flushdns`).
   2. Open your browser and empty your browser cache.
   3. Open Wireshark and enter “ip.addr == your_IP_address” into the filter, where you can obtain your_IP_address (the IP address for the computer on which you are running Wireshark) with `ipconfig /all`. This filter removes all packets that neither originate nor are destined to your host.
   4. Start packet capture in Wireshark.

   Answer the following questions:
   a) Locate the DNS query and response messages (provide screenshot). Are they sent over UDP or TCP?
   b) What is the destination port for the DNS query message? What is the source port of DNS response message?
   c) To what IP address is the DNS query message sent? Use `ipconfig` to determine the IP address of your local DNS server (provide screenshot). Are these two IP addresses the same?
   d) Examine the DNS query message. What “Type” of DNS query is it? Does the query message contain any “answers”?
   e) Examine the DNS response message. How many “answers” are provided? What does each of these answers contain?

2. (9/25, 10 points) P2P

   a) (5 points) Consider query flooding in a P2P file sharing system. Assume that each peer is connected to at most N neighbors in the overlay network and that the node-count field is initially set to K. Given an initial query, find an upper bound on the number of query messages that are sent into the overlay network.

   b) (5 points) Consider the following circular Distributed Hash Table. Describe the steps a new peer 6 needs to take in order to join this DHT. Assume peer 6 initially only knows peer 15’s IP address.
4. (9/23, 30 points) Web Server Program

In this lab, you will learn the basics of socket programming for TCP connections in Python: how to create a socket, bind it to a specific address and port, as well as send and receive a HTTP packet. You will also learn some basics of HTTP header format.

You will develop a web server that handles one HTTP request at a time. Your web server should accept and parse the HTTP request, get the requested file from the server’s file system, create an HTTP response message consisting of the requested file preceded by header lines, and then send the response directly to the client. If the requested file is not present in the server, the server should send an HTTP “404 Not Found” message back to the client.

**Code**

Below you will find the skeleton code for the Web server. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

**Running the Server**

Put an HTML file (e.g., HelloWorld.html) in the same directory that the server is in. Run the server program. Determine the IP address of the host that is running the server (e.g., 128.238.251.26). From another host, open a browser and provide the corresponding URL. For example:

http://128.238.251.26:6789/HelloWorld.html

‘HelloWorld.html’ is the name of the file you placed in the server directory. Note also the use of the port number after the colon. You need to replace this port number with whatever port you have used in the server code. In the above example, we have used the port number 6789. The browser should then display the contents of HelloWorld.html. If you omit ":6789\”, the browser will assume port 80 and you will get the web page from the server only if your server is listening at port 80.

Then try to get a file that is not present at the server. You should get a “404 Not Found” message.

**What to Hand in**

You will hand in the complete server code along with the screen shots of your client browser, verifying that you actually receive the contents of the HTML file from the server.

**Skeleton Python Code for the Web Server**

```python
# import socket module
from socket import *

serverSocket = socket(AF_INET, SOCK_STREAM)

# Prepare a server socket

#Fill in start

#Fill in end

while True:
    # Establish the connection
    print 'Ready to serve...

    connectionSocket, addr = #Fill in start  #Fill in end
```
try:
    message = #Fill in start #Fill in end
    filename = message.split()[1]
    f = open(filename[1:])
    outputdata = #Fill in start #Fill in end
    #Send one HTTP header line into socket
    #Fill in start #Fill in end
    #Send the content of the requested file to the client
    for i in range(0, len(outputdata)):
        connectionSocket.send(outputdata[i])
        connectionSocket.close()
except IOError:
    #Send response message for file not found
    #Fill in start #Fill in end
    #Close client socket
    #Fill in start #Fill in end
serverSocket.close()

5. (9/23, 30 points) Web Cache Program

In this lab, you will learn how web proxy servers work and one of their basic functionalities – caching.

Your task is to develop a small web proxy server which is able to cache web pages. It is a very simple proxy server which only understands simple GET-requests, but is able to handle all kinds of objects - not just HTML pages, but also images.

Generally, when the client makes a request, the request is sent to the web server. The web server then processes the request and sends back a response message to the requesting client. In order to improve the performance we create a proxy server between the client and the web server. Now, both the request message sent by the client and the response message delivered by the web server pass through the proxy server. In other words, the client requests the objects via the proxy server. The proxy server will forward the client’s request to the web server. The web server will then generate a response message and deliver it to the proxy server, which in turn sends it to the client.
Code
Below you will find the skeleton code for the client. You are to complete the skeleton code. The places where you need to fill in code are marked with `#Fill in start` and `#Fill in end`. Each place may require one or more lines of code.

Running the Proxy Server
Run the proxy server program using your command prompt and then request a web page from your browser. Direct the requests to the proxy server using your IP address and port number.

For e.g. `http://localhost:8888/www.google.com`

To use the proxy server with browser and proxy on separate computers, you will need the IP address on which your proxy server is running. In this case, while running the proxy, you will have to replace the “localhost” with the IP address of the computer where the proxy server is running. Also note the port number used. You will replace the port number used here “8888” with the port number you have used in your server code at which your proxy server is listening.

Configuring your Browser
You can also directly configure your web browser to use your proxy. This depends on your browser. In Internet Explorer, you can set the proxy in Tools > Internet Options > Connections tab > LAN Settings. In Netscape (and derived browsers such as Mozilla), you can set the proxy in Tools > Options > Advanced tab > Network tab > Connection Settings. In both cases you need to give the address of the proxy and the port number that you gave when you ran the proxy server. You should be able to run the proxy and the browser on the same computer without any problem. With this approach, to get a web page using the proxy server, you simply provide the URL of the page you want.

For e.g. `http://www.google.com`

What to Hand in
You will hand in the complete proxy server code and screenshots at the client side verifying that you indeed get the web page via the proxy server.
Skeleton Python Code for the Proxy Server

```python
from socket import *
import sys

if len(sys.argv) <= 1:
    print 'Usage: "python ProxyServer.py server_ip"\n[server_ip : It is the IP Address Of Proxy Server]
sys.exit(2)

# Create a server socket, bind it to a port and start listening
tcpSerSock = socket(AF_INET, SOCK_STREAM)
# Fill in start.
# Fill in end.
while 1:
    # Start receiving data from the client
    # Start receiving data from the client
    print 'Ready to serve...'
tcpCliSock, addr = tcpSerSock.accept()
print 'Received a connection from:', addr
message = # Fill in start. # Fill in end.
print message
# Extract the filename from the given message
print message.split()[1]
filename = message.split()[1].partition("/")[2]
print filename
fileExist = "false"
filetouse = "/" + filename
print filetouse
try:
    # Check wether the file exist in the cache
    f = open(filetouse[1:], "r")
    outputdata = f.readlines()
    fileExist = "true"
# ProxyServer finds a cache hit and generates a response message
    tcpCliSock.send("HTTP/1.0 200 OK\n")
tcpCliSock.send("Content-Type:text/html\n")
    # Fill in start.
    # Fill in end.
    # print 'Read from cache'
# Error handling for file not found in cache
except IOError:
    if fileExist == "false":
        # Create a socket on the proxyserver
        c = # Fill in start. # Fill in end.
        hostn = filename.replace("www.", ",", 1)
```
print hostn

try:
    # Connect to the socket to port 80
    # Fill in start.
    # Fill in end.
    # Create a temporary file on this socket and ask port 80
    # for the file requested by the client
    fileobj = c.makesfile('r', 0)
    fileobj.write("GET " + filename + "
        HTTP/1.0\n\n")
    # Read the response into buffer
    # Fill in start.
    # Fill in end.
    # Create a new file in the cache for the requested file.
    # Also send the response in the buffer to client socket
    # and the corresponding file in the cache
    tmpFile = open("./" + filename,"wb")
    # Fill in start.
    # Fill in end.

except:
    print "Illegal request"

else:
    # HTTP response message for file not found
    # Fill in start.
    # Fill in end.
    # Close the client and the server sockets
    tcpCliSock.close()
    # Fill in start.
    # Fill in end.