CS 352 Assignment 4
Web Server Characterization using Wireshark Traces

In this project you will write a Python3 program that measures web-server performance. Your program will read saved packet traces in the pcapng (packet-capture - next generation) format. This format is used by many networking analysis programs, including the popular Wireshark, and its terminal-mode version, tshark, These programs can observe live network traffic, as well as record network traffic to files for further offline analysis.

Your program will filter HTTP request packets from a trace and match the corresponding HTTP response packets, and then record the round-trip latency of each HTTP request response pair. It will report the average latency, as well as the 25th, 50th, 75th, 95th and 99th percentiles of latency. Recall the percentile is the Nth latency value if we sorted the latencies. For example, if we had 1000 response observations sorted by their values, the 25th percentile is the 250th value in the sorted list, and the 99th percentile would be the 990th value. Note that the 50th percentile is the median value.

Web site performance is often characterized by "long tails"; that is, most web requests are fast, but a significant fraction are slow. When plotted on a graph of response time vs. frequency, a long "tail" becomes visible. Thus, the average, median, 95th and 99th percentiles are often included when describing a web server's response time performance.

The scapy Python library

The scapy Python library is used for processing packets, both ones captured live from a network interface, as well as ones read from stored trace files. The library represents a packet as a series of nested Python dictionaries. In general, a packet is a dictionary where a field name in the packet, for example, the destination address or port number, is represented by a named key and the packet's field value is the value of the key. The different encapsulation levels of the packet are represented by nesting dictionaries in Python. So, for example, a dictionary holding values for a TCP packet would be a child dictionary of the dictionary holding the Ethernet information.

You can see the dictionary structure using the show() method. For example, call `packet.show()` on a packet to see what keys are available.

Installing:

You can install scapy library by running pip. For example:

```
pip3 install scapy
```

The ilab machines support installing scapy using pip3 as above.

Example program showing how to use scapy:

Below is an example program to get you started using the library.

```
#!/usr/bin/python3

from scapy.all import *
import sys
import time
import math

# make sure to load the HTTP layer
load_layer("http")
pcap_filename = "pcap1.pcap"

# example counters
number_of_packets_total = 0
number_of_tcp_packets = 0
number_of_udp_packets = 0

processed_file = rdpcap(pcap_filename)# read in the pcap file
sessions = processed_file.sessions()  #get the list of sessions/TCP connections
for session in sessions:
    for packet in sessions[session]:  # for each packet in each session
        number_of_packets_total = number_of_packets_total + 1 #increment total
packet count
        if packet.haslayer(TCP):    # check is the packet is a TCP packet
            number_of_tcp_packets = number_of_tcp_packets + 1   # count TCP
packets
            source_ip = packet[IP].src # note that a packet is represented as a
python hash table with keys corresponding to
            dest_ip = packet[IP].dst    # layer field names and the values of
the hash table as the packet field values

            if (packet.haslayer(HTTP)): # test for an HTTP packet
                if HTTPRequest in packet:
                    arrival_time = packet.time  # get unix time of the packet
                    print ("Got a TCP packet part of an HTTP request at time:
%0.4f for server IP %s" % (arrival_time,dest_ip))

        else:
            if packet.haslayer(UDP):
                number_of_udp_packets = number_of_udp_packets + 1

print("Got %d packets total, %d TCP packets and %d UDP packets" %
(number_of_packets_total, number_of_tcp_packets,number_of_udp_packets))
```

Approach:
Your program needs to match the HTTP requests with the responses. One approach is to find an HTTP request, remember arrival time and the TCP ports and IP addresses. The code then would try

to find a match to the return HTTP request using the TCP ports and IP address (recall these are inverted on the response). When a match is found, the time difference is computed as the server's response time.

Running the program:

Usage:

Your program must be named: `measure-webserver.py`

Input Arguments: a pcapng file, a destination server IP address and a destination port, as:

```
measure-webserver.py <input-file> <server-ip> <server-port>
```

The code must produce 2 lines of output:

```
AVERAGE LATENCY:  <float>
PERCENTILES:  <float-25%> , <float-50%>, <float-75%>,  <float-95%>,  <float-99%>
```

Note: values are to be reported in seconds. Use at least 5 digits of accuracy to the left of the decimal point. The autograder will give a +/- 5% tolerance on the answer.

Included project files and scripts:

You are provided with the following goodies:

- `example-scapy.py`
  - The example code above.
- `pcap1.pcap`
- `pcap2.pcap`
- `pcap3.pcap`

  These are test input files, described below.

.

Test Input Files:

The following input files are provided to test your program. The output is also provided.

pcap1.pcap:
Website: http://example.com
Destination address: 93.184.216.34
Port number: 80
AVERAGE LATENCY: 0.00290
PERCENTILES: 0.00271 0.00283 0.00300 0.00355 0.00372

pcap2.pcap:

Websites 1: http://example.com
Destination address: 93.184.216.34
Port number: 80
AVERAGE LATENCY: 0.00292
PERCENTILES: 0.00267 0.00278 0.00293 0.00358 0.00570

Website 2: http://apple.com
Destination address: 17.253.144.10
Port number: 80
AVERAGE LATENCY: 0.00287
PERCENTILES: 0.00204 0.00218 0.00248 0.00806 0.01423

pcap3.pcap:
Website 1: http://info.cern.ch
Destination address: 188.184.100.182
Port number: 80
AVERAGE LATENCY: 0.08570
PERCENTILES: 0.08550 0.08564 0.08579 0.08603 0.08891

Website 2: http://neverssl.com
Destination address: 34.223.124.45
Port number: 80
AVERAGE LATENCY: 0.08284
PERCENTILES: 0.08193 0.08267 0.08340 0.08534 0.08583

Handing in Assignment:

You need to upload your code to Gradescope. Log into canvas and use the Gradescope tool on the left. The assignment is called “HTTP Server”. Upload a Python file called `measure-webserver.py`

How to add group members in gradescope:
https://help.gradescope.com/article/m5qz2xsnjy-student-add-group-members

Extra Credit (+50%): Computing the Kullback–Leibler divergence

We could model the time it takes an HTTP server to process a request with an exponential distributed service time, with a mean service rate of λ. The exponential distribution in this scenario models the probability of the time it takes to service the request. The rate parameter λ can be estimated from the observed average of a real web server.

However, the true distribution of server responses might not be well modeled by an exponential distribution. The Kullback-Leibler divergence is a method of comparing two distributions, so we can use the KL-divergence as a metric to evaluate how closely the exponential distribution matches the observed distribution of the time it takes to service web requests. If the distributions are “close”, the exponential distribution is a good approximation of real servers. If the KL-divergence is “far”, another distribution would be a preferable model.

Recall we can model a discrete probability distribution as a number of “buckets”, where each bucket is the probability of some outcome. In our cases, a bucket is the probability that the time of an HTTP request falls in some range. For example, the probability that a web request takes between 0.02 and 0.03 seconds might be 25%. The sum of all the bucket probabilities must add to 1.

Given two discrete probability distributions, the KL-divergence is easy to compute. One distribution will come from the measured responses, which we will call the “measured distribution”. We will get the second distribution from the exponential function, which we will call the “modeled distribution”.

In this assignment we will use 10 buckets. For the measured distribution, compute a range from zero seconds to the maximum observed latency and divide this time range into 10 equal pieces. Next, assign each observed latency sample a bucket. Dividing each bucket count by the total number of observations gives the measured distribution.

Use the mean response time to create an exponential model distribution. Note that the exponential parameter is rate, so you need to make the rate, λ = (1.0 / mean response time) . To get the probability mass of each bucket, compute the exponential distribution’s CDF at each bucket’s boundary, and subtract the greater one from the lesser one; this difference in the CDF values gives the probability mass over the time-range. Note that for the last bucket, you may have to make the time-range at the edge of the distribution very large to get most of the mass to add up close to 1.

Pcap1:
Website: http://example.com
Destination address: 93.184.216.34
Port number: 80
AVERAGE LATENCY: 0.00290
PERCENTILES: 0.00271 0.00283 0.00300 0.00355 0.00372
KL DIVERGENCE: 2.87489

Pcap2:
Websites 1: http://example.com
Destination address: 93.184.216.34
Port number: 80
AVERAGE LATENCY: 0.00292
PERCENTILES: 0.00267 0.00278 0.00293 0.00358 0.00570
KL DIVERGENCE: 2.87489

Website 2: http://apple.com
Destination address: 17.253.144.10

Port number: 80
AVERAGE LATENCY: 0.00287
PERCENTILES: 0.00204 0.00218 0.00248 0.00806 0.01423
KL DIVERGENCE: 1.45393

Pcap3:
Website 1: http://info.cern.ch
Destination address: 188.184.100.182
Port number: 80
AVERAGE LATENCY: 0.08570
PERCENTILES: 0.08550 0.08564 0.08579 0.08603 0.08891
KL DIVERGENCE: 1.34722

Website 2: http://neverssl.com
Destination address: 34.223.124.45
Port number: 80
AVERAGE LATENCY: 0.08284
PERCENTILES: 0.08193 0.08267 0.08340 0.08534 0.08583
KL DIVERGENCE: 1.3453

References:

- Scapy Main Site: https://scapy.net/
- Scapy documentation: https://scapy.readthedocs.io/en/latest/
- More example code using scapy that uses trace files: https://medium.com/a-bit-off/scapy-ways-of-reading-pcaps-1367a05e98a8
- KL-divergence definition: https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence
- Another KL-divergence description: https://towardsdatascience.com/understanding-kl-divergence-f3ddc8dff254
- Computing the KL-divergence in Python: https://machinelearningmastery.com/divergence-between-probability-distributions/
- Modeling service delay with the exponential distribution: https://courses.lumenlearning.com/introstats1/chapter/the-exponential-distribution