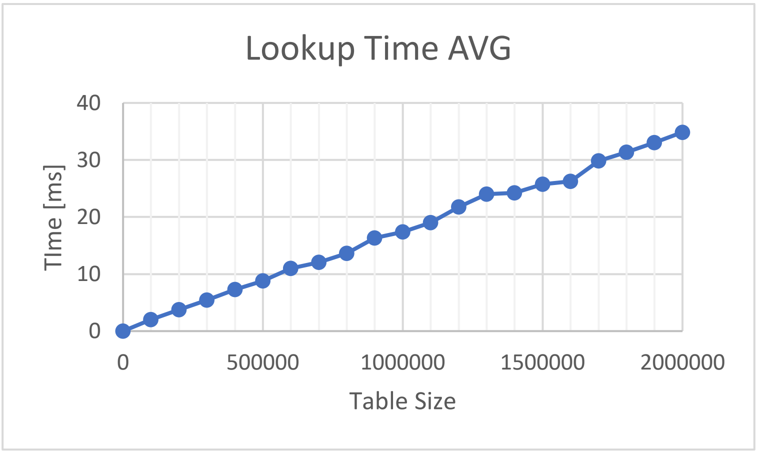
– Project 4 –
Packet Matching Algorithms
Advanced Network Security

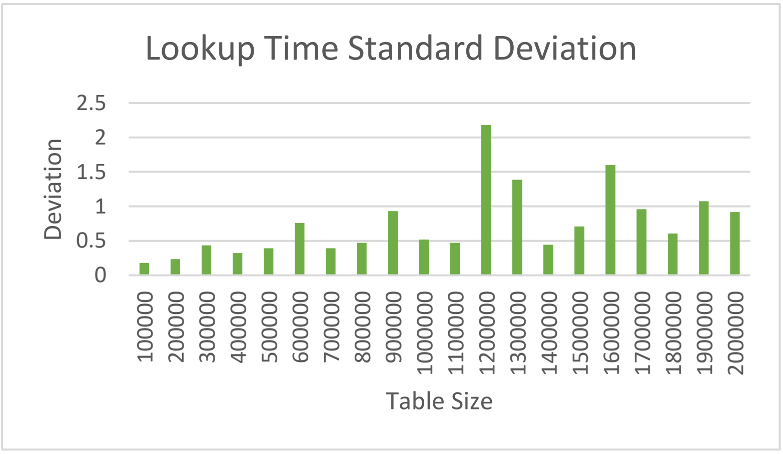
Mohammad Umeer – #4748549

Task 1:

I created an input blocking rule in the iptables using the command: "*sudo iptables -A INPUT -s 23.78.89.67 -j DROP*" and then I saved the iptables data in a file to replicate the rule multiple times. After modifying the file and uploading (restore) to the iptables and I run multiple times a packet loop request to localhost to analyse the delay variation associated to the blocked IP table size.

And this Is the result that I got: (more info in the attached excel file)





From the graph the lookup time increase linearly with the size of the table.

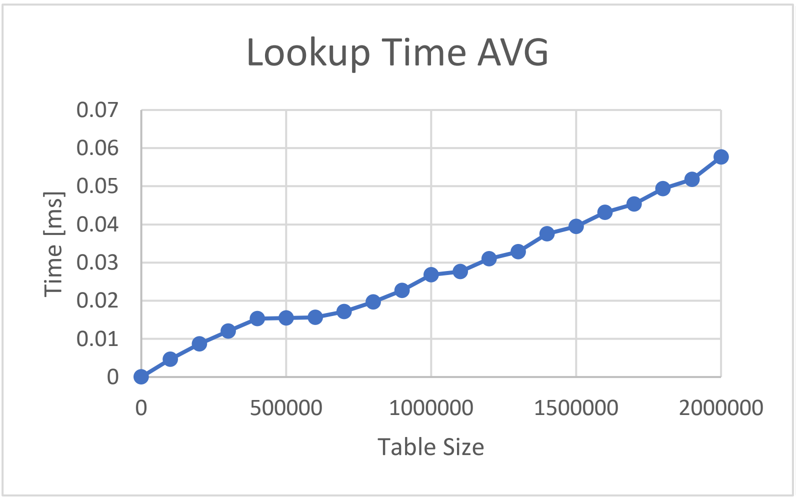
HOW TO RUN IT:

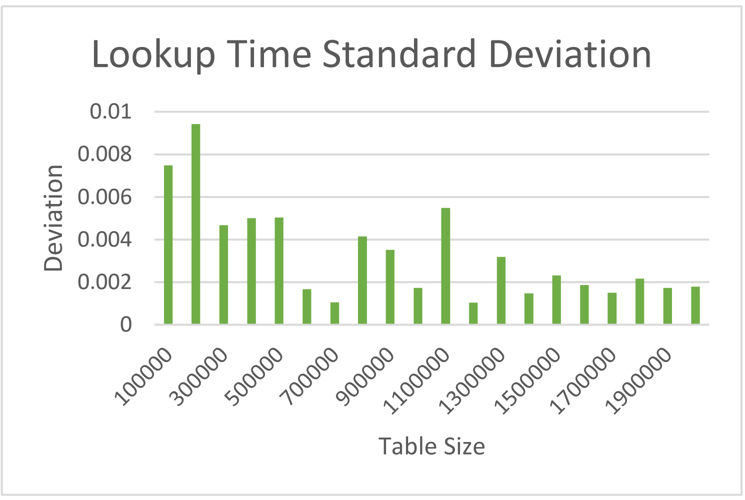
```
Sudo iptables -L

Sudo iptables -F

sudo iptables -A INPUT -s 23.78.89.67 -j DROP

sudo iptables-save > fileRuleIpset         //By doing this I downloaded and studied the structure to build a generator.

sudo ./runIPTable    //You may want to modify the SIZE inside "RandomIPTableGeneretor.c"

Sudo iptables-restore -c < fileRule2

//Here you need to analyse the lookup time, I used a ping to localhost to see the variation. (before filling the table run a
ping so you have a base line that you can remove from all the data)
```

Task 2:

In other to block a DDoS attack with iptables, the performance of the rules are extremely important because TCP-based DDoS attack types use a high peak number of packet that can take down the server.
Typically most iptables are configured with filter table and INPUT chain for anti-DDoS protection, the problem with this approach is that the INPUT chain is only processed after the PREROUTING and FORWARD chains and this cause a delay in the filtering of the packet which consumes resource; to optimize the defence we need to move the protection as far up the chains as possible. And this is possible using the mangle table and the PREROUTING chain. It is also possible to optimize the kernel to better mitigate the effects of DDoS attacks.
To reduce the load furthermore is it possible to: use the mangle table is it possible to block invalid packets, block new packet that are not SYS, drop all ICMP packets. Additionally, is also possible to limits the connections that a client can establish per seconds. If the source IP address are consecutive you can set a IP range filter.
Ipset is an extension to iptables that allows you to create firewall rules that match entire "sets" of addresses at once. Unlike normal iptables chains, which are stored and traversed linearly, IP sets are stored in indexed data structures, making lookups very efficient, even when dealing with large sets.

The result of the optimization test with Ipset is: (more info in the attached excel file)





From the table above is it possible to see that the whole graph is now lowered, because the lookup time was up to 500 times smaller.

INFORMATION SOURCE:
Paper: Mitigating-DoS-DDoS-attacks-using-iptables.pdf

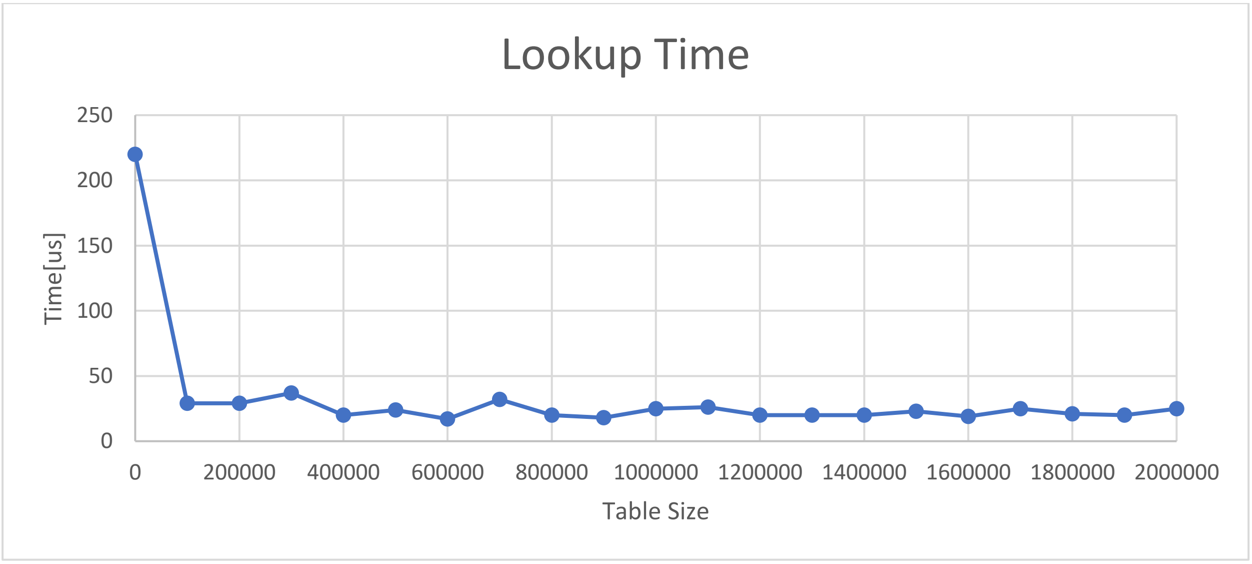
HOW TO RUN IT:

```
Sudo ipset -L

Sudo ipset -X

Sudo ipset -N myset iphash

ipset -A myset 1.1.1.1

sudo ipset save > fileRuleIpset

Sudo ipset -X        //At this point I analysed the saved file in order to automatize the creation process
```

```
sudo ./runIPSet      //You may want to modify the SIZEIpset  inside "RandomIPSetGeneretor.c"
sudo ipset restore < fileRule2
sudo iptables -A INPUT -m set --match-set myset1 src -j DROP
//Based on how many blocked IPS you need you can add more set to the iptables, each set contains 50000 IPS and the names of the set are consecutive (myset1, myset2, myset3 …) up to a total of SIZEIpset ipset.
```

Task 3:

To test the performance of the Boom Filter I created a small program with java able to simulate the filter. The result of the test where very interesting, due to the way of how the filter works there is no performance degradation associated with number IP address stored, hence the timing for all the tests were similar. The performance of filtration is amazing, just 25 microseconds (AVG) per IP lookup.
The result of the tests is: (more info in the attached excel file)



INFORMATION SOURCE:

xHash Algorithm: https://github.com/OpenHFT/Zero-Allocation-Hashing

HOW TO RUN IT:

- Load the entire folder (there are two package) in Netbeans and build & execute it, you will see the result appearing the output console after couple of seconds. Alternatively, you can compile with *javac filename.java* and run it with *java filename (*src folder*).*
- You need to modify the value of the variable NUMBER_OR_READING_TEST in "BloomFilter.java" to change the number of blocked IPS.

Task 4:

Using Java, I built up a tree generator based on dynamic class recursion.

The program firstly read the data from a file (same data as the given website) and then it creates a tree, lastly, the program tests the lookup of random IPs and the output is given on console and it consists of duration time and number of lookups

The program has multiple configuration parameters, all settable form TrieMatch.java:

- FILE_BLOCKED_IP = location of the blocked IP file
- TOTAL_BLOCKED_IP = number of IP among the latter file that the program has to actually store in the tree, if equal to -1 it builds the tree with all the IPS in the list.
- WIDTH_OF_TREE = this number indicates the structure of the tree, if equal to 1 it's a single but trie and if equal to 2 the program will build a 2-stride multi-bit trie, it is however possible to set any positive integer number.
- NUMBER_OR_READING_TEST = The output is based on an average on multiple random reading, this value indicates how many reading the program need to do.

With all the 32K IP of the file blocked and the lookup test run on 1000 different random IP, these are the average results for the tires.

Type of Trie	Time Consumed	Number of Lookup
Single Element	21us	17
2-Stride Multi Bit	15us	10

HOW TO RUN IT:

- Load the program in Netbeans and build & execute it, you will se the result appearing the output console after couple of seconds. Alternatively, you can compile with *javac filename.java* and run it with *java filename (*src folder*).*
- You need to chance the value of variable WIDTH_OF_TREE in “TrieMatch.java” to test different trie.