**Illumio Coding Assignment 2018:**

We were given to implement the Firewall class which has a method accept\_packet which takes the input of particular properties of a packet and decide whether to accept or reject the packet based on rules provided to us.

The main problems in this implementation are dealing with space and time complexities of the problem. In any implementation there is always tradeoff between time and space. I have few approaches which can be used to solve the problem and I have implemented one of the approach.

**Approach 1 (**Implemented**)**:

In this approach I have used the concept of Most frequently used rules which are used for making decision of accepting or rejecting the packet based on given properties. In this approach we maintain a cache which helps in storing the most frequent rules of the file. So, whenever accept\_packet method is invoked the method first checks for the rules in the cache instead of performing file I/O operations which are expensive. Every rule in the cache has a priority and its priority increases if packet is accepted based on that rule. So the size of cache is always maintained constant and rules present in the cache are updated based on their usage. If any rule present in the cache doesn’t accept the packet with given properties, then file is opened and given packet is checked against every rule present in the file. If packet is accepted based on the rule present in the file, then it is replaced with least recently used rules in the cache and its priority of this rule is set to 1.

**Advantages:**

1. Most important rules are placed in cache and number of file I/O operations are decreased and it decreases bottle- necks of file opening and closing.
2. If most of packets come from few rules, this approach helps in decreasing the time complexity and optimizes the space utilization the of program.

**Disadvantages:**

1. This approach makes an assumption that most of the times packets come from few hosts and we can save the rules from those hosts in cache and process the packets which may not be true all the times.
2. If packets come different hosts then cache cannot be used effectively and this results in file, I/O operations and it increases the time complexity of the problem.

**Approach 2:**

In this approach file is opened in start of the program and pointer points to the start of the program. If accept\_packet method is called to check the packet then, Pointer move through each the rule in the file and accepts the packet if certain rule satisfies in the conditions given and then pointer points back to the start of the file. If the pointer reaches the end of file, then packet is rejected and pointer is pointed back to the start of the file.

The advantage of this approach is that no extra space is required to load the rules into memory. The main disadvantage of this approach is that it leads to memory leaks as the file is never closed and time complexity of the problem depends upon the size of the file to be read.

**Approach 3:**

In this approach entire data present in the file is read into the main memory and these rules are divided into many parts and stored in different memory units. Partition of rules can be done based particular rules such as port numbers may be used to partition rules present in the file. When accept\_packet method is invoked based on port number given in parameters, rules in corresponding memory location can be used to check whether to accept or reject the packet.

Advantage of this approach is that time required to process each packet is very less because only few number of rules will be processed to check the acceptance of packet because the rules are partitioned based on port numbers.

Disadvantage of this approach is space complexity of the problem is proportional to the size of input file.

**Approach 4:**

In this approach before accepting the user input, I will preprocess my input file and divide the data into small files based on some partition rules i.e. based on direction and based port numbers and IP address. I will maintain the information of my files in the main memory and whenever I get a user request I will the load corresponding small file with the help of information present in the main memory. This will help in reading small files and less number of rules to be processed. This approach can be used along with other approaches like cache maintenance and increase productivity of the problem.