Project 4 Netfilter Kernel Module

Ziqi Yang && Xuanyang Ge

Section 2

Github repo: https://github.com/CS3281-2016/Netfilter-Kernel-Module

Project Report
Ziqi Yang
Xuangyang Ge
Github repo: https://github.com/xuanyangge/CS281FinalProject

Project 4 Kernel Module

General Description:

We first studied the concepts and examples of kernel module. Then after struggling for a while, we implemented the proc read and write functionality.

After we have everything prepared, we started to implement the network filter functionality. We use a big character array named allip to store all the ip address and blocking direction user puts in and separate each by the null terminator. Each time the user writes to the proc file, we will use copy_from_user to get the ip address. Depending on the blocking direction, we put only the ip address in the incoming ip list or the outgoing ip list. These two lists are array of character pointer that store the address of element of allip.

We also add the functionality to clear the whole list and to print out the current list.

Then we went on to the hook function that filter the network packet. We first use the ip_header structure to extract the ip address from the network packet. We use strcmp to compare the ip address with each we have in the list. If we found it in the list, we drop the packet, and if not, we accept the packet. In the case we drop the packet, we will print the message to the kernel using printk function.

Difficulties We Encountered:

We really spent a fair amount of time on /proc file system. As we all know, the example is not compiled in the latest version of Linux system and there are some errors in the original example. Luckily, the professor provided a corrected version and so we can move on.

Another difficulty that we encountered is to extract the ip address from the network packet and compare it with the user’s list. At the beginning, we always got the same source and destination addresses, 127.0.0.1 and 127.0.1.1. We managed to fix that by getting rid of part of the code from an external example. Then we found that even though the ip address is in the list, strcmp still failed to return the expected result.

After digging into the Linux documentation and printing out all the possible information we need, we finally noticed that we put the string terminator in a wrong position for each ip address in the list. Then after we fix that, our netfilter module works.

Answer Questions on Piazza:

- In your project report, describe how the netfilter hooks work, what kinds of hooks there are, and how you would implement more advanced functionality, such as a stateful firewall (https://en.wikipedia.org/wiki/Stateful_firewall).
- Describe, with netfilter specific terms, how you would enforce a "quota" on how much traffic a user can generate to/from a specific address.

Hooks in Netfilter are implemented such that they will invoke callback functions in kernel modules that are registered with the hooks once some kernel's networking stack is reached. For example, pre-routing hooks invoke callback function when the network stack before the routing stage is reached.

In our project, we register two hooks, one is NP_IP_LOCAL_IN hook, NP_IP_LOCAL_OUT hook, these hooks are called when a packet is passed into the routing stage and the packet is destined for a local process. If the packet is incoming, it is LOCAL_IN, otherwise it is LOCAL_OUT.

```
--->[1]--->[ROUTE]--->[3]--->[4]--->
              |            ^
              |            |
              |         [ROUTE]
              v            |
             [2]          [5]
              |            ^
              |            |
              v            |
```

There are 5 hooks in various points in the IPv4 protocol stack. There are NF_IP_PRE_ROUTING hook, NF_IP_LOCAL_IN hook, NF_IP_FORWARD hook, NF_IP_POST_ROUTING hook and NF_IP_LOCAL_OUT hook.

Pre_Routing invokes callback function before the routing stage starts.

NP_Forward invokes callback function during routing stage if the packet is destined for another process instead of a local process.

Stateful Firework:

Use array of pointer of character to store a state table. For every network packet on the flow, check whether the ip address, port number and sequence number match an entry in the state table. If none, as it necessarily happens at the beginning, then we perform a detailed check over the userlist proc file. If the packet did not match any target address, we add the ip address, port number and sequence number to the state table. If the packet is from or goes to one of the target ip address, we simply drop it.

With the state table data structure, we are able to rapidly process network packet after the state table is established. Since it’s a dynamic allocated structure, we need to kmalloc the space first and kfree the space when exit the module.

We will also use an array of integer to keep track of the how many times traffic has passed a certain connection. After a certain period of time or whenever the state table is filled up, based on this array, we clean the entries that hasn’t been matched to some extend. After we clean the state table, we will also reset this array to empty.

Quota enforcement:

To keep track of how many quota has been transmitted, we use the Total Length field in the ip header structure for each packet. After the packet passes the netfilter phase, we use an array to store the amount of quota for each ip address recorded. To be specific, we use an array of pointer of character array to store the ip address that has been accepted. And we use an array of integer to store the quota. The quota in integer array corresponds to the ip address in the same index in the character pointer array. Therefore, we need to perform a linear search each time a new packet comes in.

Every time some packet comes, we add it to the array that contains the info about the amount of data already received from that IP address. If the quota for a specific ip address exceeds a certain amount, we drop the packet, and print message to the kernel.

Network traffic kernel module

Xuanyang Ge, Ziqi Yang

Big picture

- Our project is a Linux kernel module designed to be able to block either incoming or outgoing traffic ip-address specified by a user.
- We implement our module under Linux version 4.4.
- Netfilter is a framework that uses a set of hooks to register callback functions with the kernel’s network stack. The functions usually filter and monitor on the network stack

Methodology/Key function calls

- Use proc to allow users to add new IP-address into the block list.
- Use NF_INET_LOCAL_IN and NF_INET_LOCAL_OUT for incoming and outgoing traffic packet separately.
- Use ip_header->sadd and ip_header->daddr to get source and destination addresses. Then compare the IP-addresses with the IP-addresses in the proc file.
- Use printk for logging info about the module.

Usage

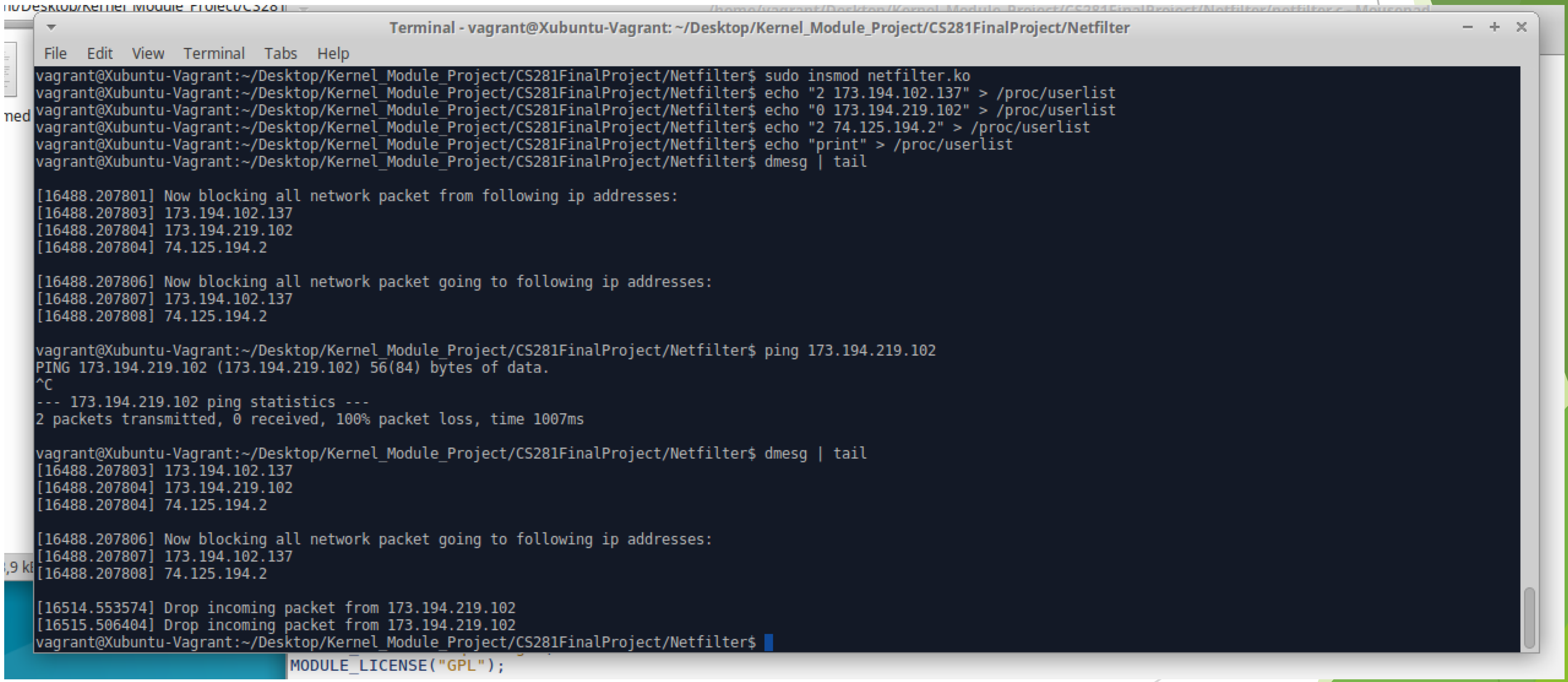
- In order to add new IP-Address to block, type echo “$i $IP_address” > /proc/userlist. Where i is 0,1,2 indicating the IP_Address should be blocked either incoming, outgoing, or both traffic. Type echo “r” > /proc/userlist will renew the list and erase previously stored addreses. Type echo “p” > /proc/userlist will see all the addresses that are currently blocking.

- To check the IP-Address is correctly block, use ping or other related commands to try connecting the IP-Address. Then check system logging file using dmesg command. If the IP is correctly blocked it will show something like “Drop package from/to IP_address”

- A detailed instruction can be found in README file in our github repo.

Result

- We successfully realize the functionality, here is an example of a small test scripts we write and it successfully drops the package from the address we wish to block.


Terminal - vagrant@Xubuntu-Vagrant: ~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter
File Edit View Terminal Tabs Help

```
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ sudo insmod netfilter.ko
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ echo "2 173.194.102.137" > /proc/userlist
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ echo "0 173.194.219.102" > /proc/userlist
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ echo "2 74.125.194.2" > /proc/userlist
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ echo "print" > /proc/userlist
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ dmesg | tail

[16488.207801] Now blocking all network packet from following ip addresses:
[16488.207803] 173.194.102.137
[16488.207804] 173.194.219.102
[16488.207804] 74.125.194.2

[16488.207806] Now blocking all network packet going to following ip addresses:
[16488.207807] 173.194.102.137
[16488.207808] 74.125.194.2

vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ ping 173.194.219.102
PING 173.194.219.102 (173.194.219.102) 56(84) bytes of data.
^C
--- 173.194.219.102 ping statistics ---
2 packets transmitted, 0 received, 100% packet loss, time 1007ms

vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$ dmesg | tail
[16488.207803] 173.194.102.137
[16488.207804] 173.194.219.102
[16488.207804] 74.125.194.2

[16488.207806] Now blocking all network packet going to following ip addresses:
[16488.207807] 173.194.102.137
[16488.207808] 74.125.194.2

[16514.553574] Drop incoming packet from 173.194.219.102
[16515.506404] Drop incoming packet from 173.194.219.102
vagrant@Xubuntu-Vagrant:~/Desktop/Kernel_Module_Project/CS281FinalProject/Netfilter$
```

```
MODULE_LICENSE("GPL");
```

Collaboration Doc

Project Report
Ziqi Yang & Xuangyang Ge
Github repo: https://github.com/xuanyangge/CS281FinalProject

We did most part of the work together. Xuanyang contributes a lot in understanding and implementing the proc functions. He fixed the bugs in the example himself before the instructor post the answer on piazza. Moreover, he wrote the skeleton of our module even before thanksgiving. He also debugged the code carefully after we finished the hook function.

Ziqi Yang did most part of the user input interface. He chose the structure to store the ip address and designed the way user interact with the module using the proc file.

We wrote the hook function and the rest of the code together. Other than the code, Ziqi Yang wrote the README in github repo and the document describing the project, while Xuanyang Ge made the slide and combine all the doc into the final report.