

GPU Based Stateless Firewall

Student Name: Arnav Kumar
Roll Number: 2016017

BTP report submitted in partial fulfillment of the requirements
for the Degree of B.Tech. in Computer Science & Engineering
on 15th April, 2019

BTP Track: Research

BTP Advisor
Prof. Sambuddho Chakravarty

BTP Co-Advisor
Prof. Ojaswa Sharma

Indraprastha Institute of Information Technology
New Delhi

Student's Declaration

I hereby declare that the work presented in the report entitled “**GPU Based Stateless Firewall**” submitted by me for the partial fulfillment of the requirements for the degree of *Bachelor of Technology in Computer Science & Engineering* at Indraprastha Institute of Information Technology, Delhi, is an authentic record of my work carried out under guidance of **Prof. Sambuddho Chakravarty & Prof. Ojaswa Sharma**. Due acknowledgements have been given in the report to all material used. This work has not been submitted anywhere else for the reward of any other degree.

.....
Arnav Kumar

Place & Date: New Delhi, 15/04/19

Certificate

This is to certify that the above statement made by the candidate is correct to the best of my knowledge.

.....
Sambuddho Chakravarty

Place & Date: New Delhi, 15/04/19

Abstract

The use of Stateful Inspection in Firewalls has been in use for a long time. Also called Dynamic Packet Filtering, it monitors the state of connections and checks to see which packets to allow through the firewall. But the disadvantage of using a Stateful Firewall is the decrease in speed compared to Stateless Firewalls. There is a need to develop practical infrastructure for Stateless Firewall for improvements in speed while still retaining some of the security features used by Stateful Firewall. Due to the high number of threads and the capability of high parallelization of work loads, a Graphics Processing Unit will be used for the acceleration of the firewall. In this report, I explain my approach to do the same that have worked till now and how I tackled some of the problems that I faced.

Keywords: Network Security, Computer Networks, Firewalls, GPU, CUDA, Netmap, Packet Filter

Acknowledgments

I would like to thank all those who have guided me throughout this work. Firstly, I would like to thank my advisor, Prof. Sambuddho Chakravarty, for giving me this opportunity to work with him and always being patient with me and providing me with enough time for the development. He has been very helpful to me throughout my project. The knowledge that he has imparted to me throughout our project is invaluable to me. Secondly I would like to thank my co-advisor, Prof. Ojaswa Sharma for guiding me and giving me the insights about how a GPU works, the correct times to use the GPU and helping me in understanding the functioning of GPU programming (CUDA) as well as helping with debugging of CUDA code.

I'd also like to thank my father for always motivating me throughout the project.

Contents

1	Introduction	2
1.1	What Is A Firewall?	2
1.1.1	Stateful Firewall	2
1.1.2	Stateless Firewall	3
1.1.3	What is a State?	3
1.2	What is a GPU?	4
2	Motivation	5
3	Background	6
3.1	How Does a Network Work?	6
3.2	How Is Information Stored in a Network Packet?	7
3.2.1	Transmission Control Protocol(TCP)	7
3.2.2	User Datagram Protocol(UDP)	7
3.2.3	Internet Control Message Protocol(ICMP)	7
3.3	Netfilter	8
3.3.1	Netfilter Hooks	8
3.4	CUDA	9
3.4.1	Thread Hierarchy In Cuda	9
3.5	Netmap	9
3.5.1	How Does It Help Us?	10
3.6	Features of a Firwall	10
3.6.1	Packet Filtering	10
3.6.2	Network Address Translation (NAT)	10
3.6.3	Encryption	11
4	Our Approach	12
4.1	Description of The System	12
4.2	How Were The Packets Accessed?	13
4.3	How Were the Packets Modified?	13

4.4	Using The GPU	14
4.4.1	What processing is done on the GPU Currently?	15
5	Challenges Faced	16
6	Related Work	18
7	Future Plan	19
8	Conclusion	20

Chapter 1

Introduction

1.1 What Is A Firewall?

Although a lot of definitions out there exist for a firewall, the main gist in all of them is that - "A firewall is a software or a piece of hardware which is installed in a networked environment and acts upon the security policy defined by the network administrator so as to forbid or allow the communication with an external network".

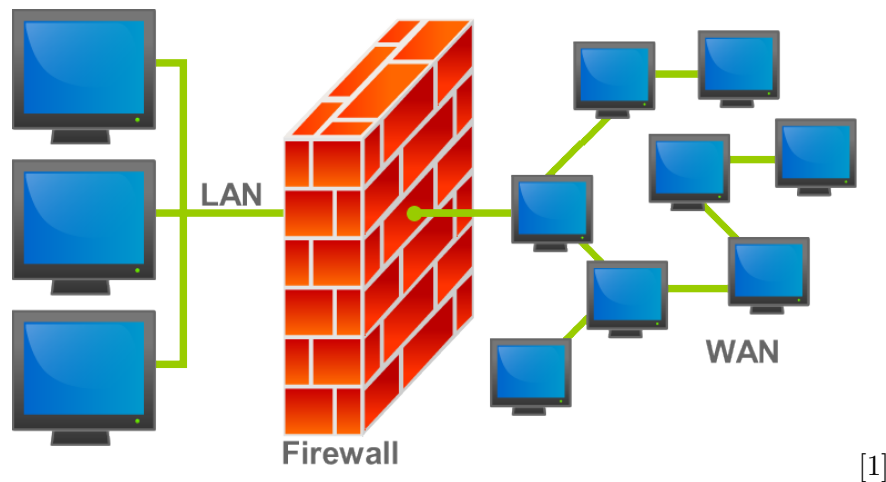


Figure 1.1: Representation of a Firewall

1.1.1 Stateful Firewall

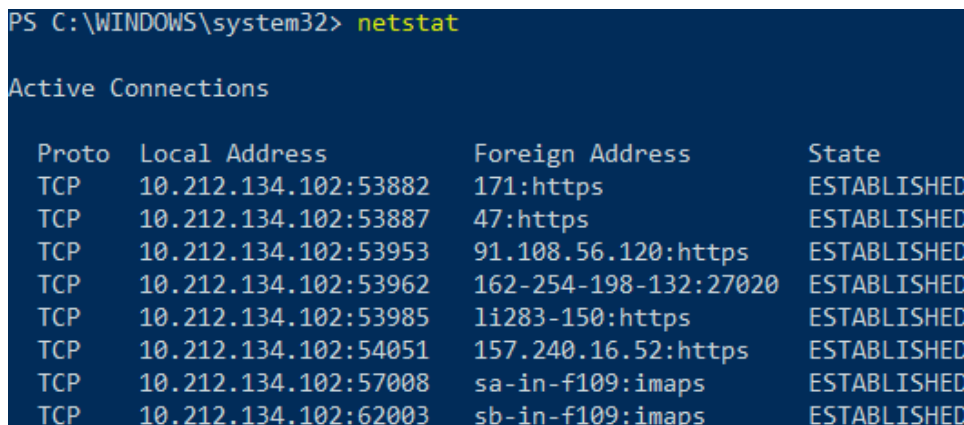
Stateful firewalls have been around for a long time and since have been become the staple in Linux and other open source operating systems (like Free-BSD). The technology was 1st introduced in 1994. Stateful firewalls also known as dynamic packet filts, keep track of the state of the network connections in memory (like TCP or UDP). It keeps track of the incoming and the outgoing packets and based on the network rules defined by the network administrator, allows the packets in the network connection to go through or get blocked by the firewall.

1.1.2 Stateless Firewall

Before stateful firewalls came into being, firewalls were stateless. In stateless firewalls also called static packet filters, each packet is treated individually, and it can't be determined whether a packet is part of an existing connection or is trying to establish a new one. The analysis of the packets are based on source and destination addresses or some other static value in the packet/frame. Stateless firewalls are more efficient in terms of speed because only the headers of the packet are looked upon and there is no additional overhead of looking up the state table for each packet(as there is in stateful firewalls).

1.1.3 What is a State?

State can be defined in multiple ways in computer networks. Basically, in a given communication session, the condition of 'being' is called the state. In a 'Stateful' firewall, the 'state' of the communication session is tracked in a table. This information of state is used to uniquely identify the communication session it represents. The sequence/acknowledgment numbers, source/destination IP address or any other flag etc. are all part of this information. So when the traffic passes, or returns the firewall uses the state table information to determine whether the packet is part of a current communication sessions or not. To give an example, typing `netstat` in Windows Command Prompt or Powershell will give you the state information of active connections to or from the host.



```
PS C:\WINDOWS\system32> netstat
```

Proto	Local Address	Foreign Address	State
TCP	10.212.134.102:53882	171:https	ESTABLISHED
TCP	10.212.134.102:53887	47:https	ESTABLISHED
TCP	10.212.134.102:53953	91.108.56.120:https	ESTABLISHED
TCP	10.212.134.102:53962	162-254-198-132:27020	ESTABLISHED
TCP	10.212.134.102:53985	1i283-150:https	ESTABLISHED
TCP	10.212.134.102:54051	157.240.16.52:https	ESTABLISHED
TCP	10.212.134.102:57008	sa-in-f109:imaps	ESTABLISHED
TCP	10.212.134.102:62003	sb-in-f109:imaps	ESTABLISHED

Figure 1.2: State of Active Connections of The Host

State & TCP

Now, TCP is a connection oriented protocol i.e. the state of the sessions following this protocol can be solidly defined. TCP is known as a stateful protocol because the beginning/end of the communication session is well defined as well as the state of the connection. There are 11 'states' in a TCP connection as defined in RFC 793[15]. Below listed are some of the better known states.

- **CLOSED** - Exists before a connections actually starts.

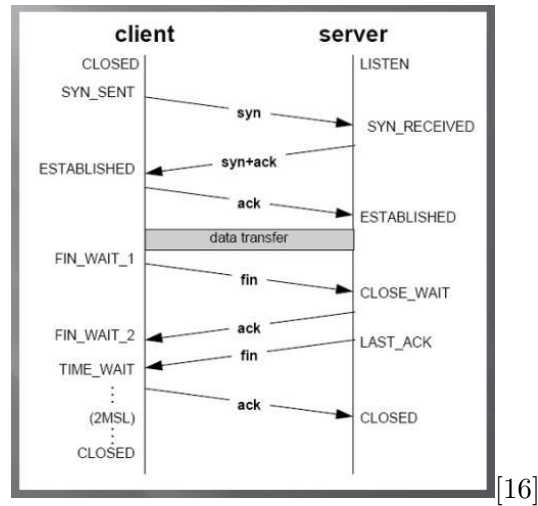


Figure 1.3: The States involved in a TCP connection.

- **LISTEN** - The host is in a listening mode for a request to start a connection. This is actually the starting state of a TCP connection.
- **SYN-SENT** - When the host is waiting for a SYN-ACK reply after sending out a SYN packet.
- **SYN-RCVD** - When the host has received a SYN packet and is replying with its SYN-ACK packet.
- **ESTABLISHED** - The connection goes into this state after its necessary ACK packet has been received. This is the state the initiating host goes after receiving a SYN-ACK packet while the responding host goes into this state after receiving the lone ACK packet.

1.2 What is a GPU?

A graphics processing unit is a single-chip processor which is used to boost and manage the performance of video and graphics. GPUs are used in embedded systems, personal computer, mobile phones, gaming consoles among many other machines. The highly parallel structure makes GPU more efficient than general purpose CPUs. Due to this highly parallel nature and the high number of cores present on a GPU and its multiple thread model, it has now become a trend to maximize efficiency and parallelization in areas like machine learning, encryption, crypto-currency mining etc. using GPUs.

Chapter 2

Motivation

The Need for a Stateless Firewall. In organizations around the world, Linux is preferred as the default operating system to implement a firewall in the network[5]. Most of the famous firewalls that are used such as IPTables, PFsense, Shorewall all do stateful packet filtering. We are attempting to build a Stateless Firewall that doesn't require the Linux Netfilter Framework and actually bypass it. Stateless firewalls are faster compared to stateful firewalls in packet filtering and perform better under heavier loads[6]. Also there does not exist any modern practical infrastructure for a stateless firewall right now (to the best of our knowledge) so we are trying to fill that gap as there are plenty of applications of a stateless firewall(including but not limited to packet filtering, NAT, encryption & VPN). Also in places where the possibility of a rogue packet is very low such as a data center or a server farm, stateless firewalls can provide huge improvements in speeds while still giving a respectable level of security in the networked environment. The firewall can also be deployed on a portable computer such as a Raspberry Pi which can act as a lightweight portable router.

The Need for a Graphics Processor Unit (GPU) on a Firewall GPUs provide a powerful platform for creating network related applications and parallelizing the network traffic for a boost in speed using low-cost hardware(if multiple functionalities such as NAT and Encryption are simultaneously required). GPUs provide a high number of cores which are ideal for high throughput jobs such as processing hundred thousands to millions of packets as compared to a CPU which comparatively provide very few number of cores to work with[7]. GPUs have quite a number of applications in technology such as in game rendering[8], deep learning and AI[8], cryptography[9] and encryptions[10] and people have realized its potential in the use of network traffic to process the high number of packets of the network[11]. What we are trying to achieve is to accelerate the speed of our stateless firewall for a high number of flows using a GPU for a substantial improvement in packet filtering, NAT and Encryption.

Chapter 3

Background

3.1 How Does a Network Work?

Computer systems across the world follow the OSI(Open Systems Interconnection) model which is a standardized model, for networking and all it's functionalities. It consists of seven layers named - Physical, Data link, Network, Transport, Session, Presentation & Application from layer 1 to layer 7 respectively.

For general networking purposes, the OSI model is reduced to protocol or network stack which combines the session, presentation and application layers (5,6,7) into one layer - application. Any sort of functionalities of session layer or presentation layer are implemented within the application layer independently by the respective application.

We are concerned with Layer 2 (Data link), Layer 3 (Network), Layer 4 (Transport) and Layer 5 (Application) for the purposes of the project.

Some more insight into the concerned layers-

- **Data Link** - This protocol layer is used to transfer frames using the physical link within hosts in the same local area network (LAN). Ethernet is the most widely used protocol in data link layer and has since become an industry standard.
- **Network** - This layer is responsible for the correct routing and forwarding of the packets between nodes(either networked hosts, switches or routers). IP(Internet Protocol) is the most widely used protocol in the network layer.
- **Transport** - This layer is called an end-to-end layer because instead of providing connection between hops, the transport layer provides point-to-point(host to host) connection between specific application processes running on the host machines either reliably or unreliably. TCP(Transmission Control Protocol) is the most widely used protocol in this layer and ensures reliable connectivity.
- **Application** - This layer is responsible for the actual data/information that a host transmits. It is the layer through which applications interact in a networked environment. It consists of plethora of services such as SSH(Secure Shell), HTTP(Hypertext Transfer Protocol), FTP(File Transfer Protocol) and a whole lot more.

3.2 How Is Information Stored in a Network Packet?

3.2.1 Transmission Control Protocol(TCP)

It is the most widely used protocol in the world for data delivery and communication[29]. TCP provides ordered, reliable and error-checked delivery of the packets to the receiver without the user having to worry about implementing these functionalities at the application level.

3.2.2 User Datagram Protocol(UDP)

UDP protocol is also one of the protocols used for data delivery and communication. UDP is a connectionless protocol which means there is no guarantee on reliable data delivery neither any guarantee on the ordering of the packets. It also has no handshaking mechanism for initiating the connection(like the 3-way handshake in TCP). That is why UDP is called a no frills best effort protocol.

3.2.3 Internet Control Message Protocol(ICMP)

The ICMP protocol is used by machines & routers to send messages/information about the operation or functioning of network devices(for eg. whether a particular node is active or inactive). The utility ping uses ICMP as its underlying protocol.

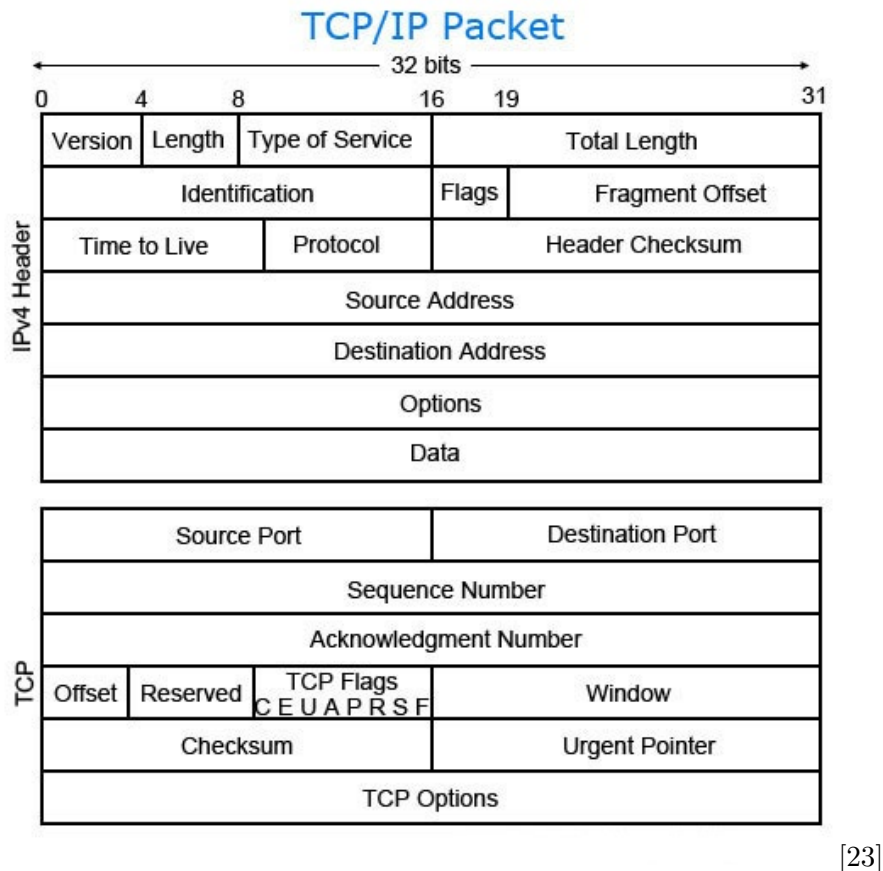


Figure 3.1: The fields in IP and TCP headers in a packet

The figure above displays the fields that are used to create the TCP and IP headers in a network packet. The key fields are source address, destination address, source port, destination port and protocol number. These 5 key fields together are known as the 5-tuple and are a crucial aspect for a majority of networked processing or telemetry.

The entire packet or datagram consists of ethernet/frame header(14 bytes), IP header(20 bytes), TCP header(20 bytes), application data(variable byte size), ethernet/frame trailer(4 bytes).

When the packet arrives at a node, each layer peels its respective header and then accordingly either sends the packet to the layer above or discards it(due to a bad checksum for example).

When the application layer wants to send data, each layer attaches its header to the packet and then sends it to the layer below until the packet is on the physical layer after which it is ready to be sent on the wire.

The minimum size of a TCP packet is 64 bytes and the maximum size is 1500 bytes which is also called the MTU(Maximum Transmission Unit).

3.3 Netfilter

Netfilter[11] is the packet filtering framework which has been present in Linux since the 2.4.x kernel series. The most common utility program associated with Netfilter is IPTables which is the default Linux firewall. Netfilter is the staple framework upon which most of the open source firewalls are built upon. By definition it is "a set of hooks inside the Linux kernel that allow other kernel modules to register callback functions with the network stack. A registered callback function is then called back for every packet that traverses the respective hook within the network stack"[11].

Netfilter has provisions for both stateless and stateful packet filtering (IPV4 & IPV6).

3.3.1 Netfilter Hooks

There are primarily 5 Netfilter Hooks and each serve a purpose in the protocol stack.[17]

- **NF_IP_PRE_ROUTING[1]** - After arriving to the Network Interface Card(NIC) and passed its checks (eg. Checksum OK), this is the first hook the packet arrives to. This is the most early access to a packet that a user can have with Netfilter.

After the 1st hook, the routing code then decides whether the packet is intended for the machine or is to be forwarded to the next machine.

- **NF_IP_LOCAL_IN[2]** - If the routing code determined that the packet is destined for the machine(host) then this is the 2nd hook the packet arrives to.
- **NF_IP_FORWARD[3]** - If the packet is intended for another machine/interface then this is the 2nd hook the packet arrives to.
- **NF_IP_LOCAL_OUT[5]** - This is the hook for the packets that originate from the local machine(host).
- **NF_IP_POST_ROUTING[4]** - This is the final netfilter hook the packet arrives to before going on "the wire". Packets from NF_IP_FORWARD or NF_IP_LOCAL_OUT hooks arrive here(because they are outbound).

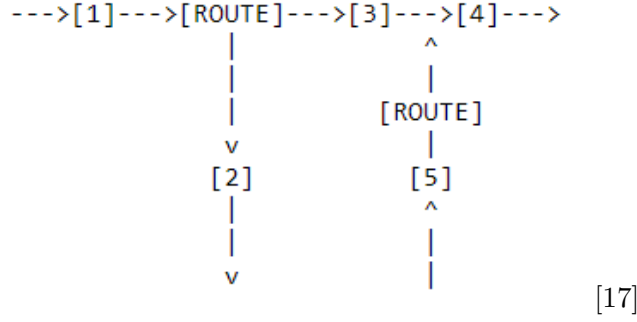


Figure 3.2: A Packet Traversing the Netfilter System

3.4 CUDA

CUDA is a parallel computing platform and programming model developed by NVIDIA for general computing on GPUs[12]. A similar platform called OpenCL(developed by Apple Inc.) exists which is open sourced while CUDA is proprietary but for the scope of this report, CUDA is going to be the framework which is used.

3.4.1 Thread Hierarchy In Cuda

There are 3 main keywords associated here - Thread, Block & Grid.

Threads are essentially the basic element of the data to be processed[19]. They are very lightweight and a context switch between two different threads is not a costly operation (unlike in a CPU).[19]

Threads in CUDA are divided into **blocks** (thread blocks to be specific). They are nothing but a programming abstraction for a group of threads that are to be run parallelly. Combination of blocks form a **grid**. It is necessary for all the blocks in a grid to have the same number of threads.

Any function or program which is to be run on a GPU is called a **kernel**.

On a GPU, threads run in **warp** step. A warp is a set of 32 threads which all execute the same instruction.

3.5 Netmap

One of the main obstacle we had was to access the network packets in userspace instead of in kernel (as CUDA works only in userspace not in kernel space). Previously we used the Netfilter library `libnetfilter_queue`[30] to add an IPTables rule to send all incoming packets to userspace where a program can access it with the necessary functions and then send it back to the kernel for processing but this was not desirable. Due to the packet going back and forth in the kernel and userspace, it made the process slow and added additional delay.

We came across `netmap`[20] which is a framework for fast packet I/O from userspace and we used it solve this obstacle of ours.

3.5.1 How Does It Help Us?

`netmap` supports I/O using a port which can be either connected to directly the NIC or to the host stack. These ports use preallocated circular queues of buffers (rings) which reside in a memory mapped region[21].

Inbound packets arrive on the RX(Receive) Ring and the outbound packets are sent through the TX(Transmit) Ring.

`netmap` provides a program `bridge.c` which is a simple bidirectional interconnect between two ports[22]. Our firewall is built upon this bridge program after heavily modifying it to implement the desired functionalities. The `bridge.c` provided the necessary setup to access and mangle(modify) the packets in userland however we want to and send it to its intended destination.

3.6 Features of a Firewall

There are a plethora of variations and features that are available on firewalls(either hardware or software). But below are the features we've chosen to implement on the firewall currently.

3.6.1 Packet Filtering

This is the most basic and still, one of the most important features of a firewall. By inspecting the 5-tuple, the firewall either allows/blocks a packet entering/exiting a network. Some firewalls use Deep Packet Inspection(DPI) to inspect the application data and determine the fate of the packet.

3.6.2 Network Address Translation (NAT)

As more and more devices capable of being part of a network emerged, the number of IPv4 addresses started diminishing. To overcome this IP address exhaustion problem, a mechanism called Network Address Translation was created which provides multiple machines in a network with different private IP addresses but a same public IP address for the world outside the network.

When a packet leaves the network, the source address of the packet is changed to the gateway's/firewall's IP address; this is called SNAT or Source Network Address Translation. When a packet intended for a host inside the network arrives on the gateway/firewall, the NAT table is looked up, and the destination address is changed to the private IP address of the host which initiated the connection; this is called DNAT or Destination Network Address Translation.

There are 3 types of NATs - one to one(1:1), many to one(m:1), many to many(m:n).

One-to-One NAT is done when there is only one machine inside a private network and wants to access the internet. The single private IP address is replaced by a single public IP address. This is also called a Static NAT.

Many-to-One NAT is done when there are multiple machines inside a network and all are assigned a private IP address which is replaced by a single public IP address when any of them wants to access the internet.

Many-to-Many NAT is done when there are multiple machines inside a network and there

are >1 gateways in the network with a public IP address. When the machines want to access the internet, their private IP address is replaced by one of the public IP addresses depending on the gateway the packet is going out through. The number of public IP addresses is still lower than the number of private IP addresses in the network.

The functionality of a NAT firewall is that it allows the internet traffic to pass through the gateway only if a machine in the private network requested it[24]. Any sort of unrequested packet or data is dropped/discarded. This prevents any communication with potentially dangerous devices on the internet.

3.6.3 Encryption

These days, there are multiple firewalls that exist that have the ability to act as a VPN gateway on top of providing all the firewalling features. A VPN or a Virtual Private Network is a mechanism to add security & privacy in public and private networks (Internet and public WiFi hotspots). As it has become incredibly easy to intercept and snoop on any data travelling through a network, VPNs provide an added level of security by encapsulating the data in an encrypted data packet and then sending it forward which helps to protect the data from any sort of MITM(Man-in-the-Middle) attacks.

There are plenty of cryptographic algorithms that are used to encrypt the data in a VPN but the most commonly used and secure is AES(Advanced Encryption Standard) which is also the algorithm adopted by the U.S government and is now used globally.

We plan to implement a VPN gateway in our firewall which will also use AES as the underlying cryptographic algorithm. We plan to speed up the process of AES encryption/decryption by harnessing the power of the GPU & by using the Galois/Counter(GCM) mode of AES which is parallelizable (unlike Cipher Block Chaining mode which can't be parallelized). We won't be using the Electronic Code Book(ECB) mode of AES(which is parallelizable) for encryption as it is proven to be semantically broken.

Chapter 4

Our Approach

4.1 Description of The System

For the sequential networking part, we had created a custom setup which was used to test all the features of the firewall. The setup consists of 3 hosts with one acting as the client, one as the server and the one in the middle acting as the default gateway for both of them(using two individual virtual network adapters).

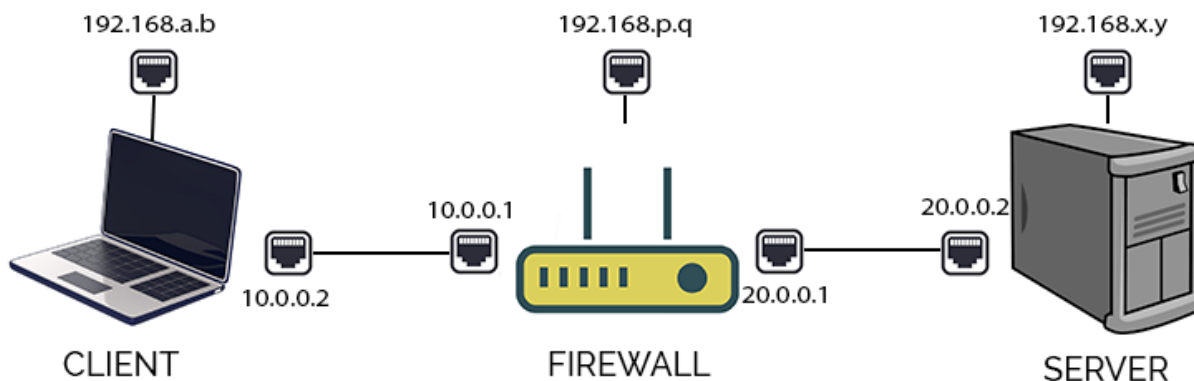


Figure 4.1: The Setup Used

Three Ubuntu 16.04 virtual machines were set up for the configuration. The client and server machines both have one 'NAT' virtual network adapter (to communicate with the internet) and one 'Host-Only' virtual network adapter (for intra-VM communication).

The firewall machine has one 'NAT' and two 'Host-Only' virtual network adapters. The interface with the IP address 10.0.0.1 is used as a default gateway for the client for any outbound packet to 20.0.0.0/24 subnet(subnetwork) and the interface with the IP address 20.0.0.1 is used as a default gateway for the server for any outbound packets to 10.0.0.0/24 subnet.

A **Default Gateway** is a node which serves as the default forwarding host to other networks. In the routing table, if there are no other entries that match the destination address of the packet, then that packet is sent to the default gateway & the default gateway forwards it accordingly.

The bash command used to set a default gateway is `route add -net <subnet address>gw <gateway ip address>dev <interface name>`

For example - on the server virtual machine, the command was `route add -net 10.0.0.0/24 gw 20.0.0.1 dev ens39`.

One needs to have root access or be part of the sudoers group to execute the command as it modifies the routing table of the operating system.

4.2 How Were The Packets Accessed?

Section 3.5 describes `netmap` and its usage. As mentioned earlier, the sample `netmap` application-`bridge.c`[25] was used and modified accordingly to access and modify the packets. The code ran on the middle(firewall) machine.

A brief explanation of the code and the `netmap` API-

The `nm_open` `netmap` function is used to open the `/dev/netmap` device file for both the interfaces (10.0.0.1 and 20.0.0.1) and a `netmap descriptor` is assigned to both of the interfaces. The file descriptor present in the `netmap descriptor` are then assigned to a global structure `pollfd` and is then used to check for any incoming packets on the NIC ring (like a polling file descriptor normally does). The function `move` in `bridge.c` defines two `struct netmap_ring` - `txring` & `rxring` which are used to access the transmit and receive rings. The `struct netmap_slot` variables `rs` & `ts` are then used to access the individual slots in the `txring` or `rxring`.

To access the actual packet from the ring slot, a char pointer or a char array (of maximum size 1514) is used. The `NETMAP_BUF` `netmap` function is then used to access the packet in the slot in the ring. The function `NETMAP_BUF` takes two arguments - the tx/rxring & the index of the slot in the ring.

After we have the packet (either in the char pointer or the char array), we use the structures IP header(`iphdr`) & TCP header(`tcphdr`) defined in the header files `linux/ip.h` & `linux/tcp.h` respectively to typecast the char pointer/array and extract the necessary headers from the packet for further inspection and modification.

4.3 How Were the Packets Modified?

After extracting the headers from the packet, to perform NAT, the field `iphdr->saddr` (source IP address) or `iphdr->daddr` (destination IP address) is modified to the public ip address of the gateway (20.0.0.1 in this case) using the line `iphdr_var->saddr==inet_addr("20.0.0.1")`. The function `inet_addr` converts the IP address(given as argument) to the proper network order and returns it.

After modifying the IP address, the IP checksum needs to be modified as well or else the kernel on the receiving side will discard the packet. So using a generic IP checksum function[26] the IP checksum is recalculated and the `iphdr_var->check` field in the packet is replaced. Two more fields are also needed to be modified in the packet, the source MAC address and the destination MAC address. The approach we use to access the packets basically bridges the two interface, effectively bypassing the host stack(Netfilter and kernel checks) so every aspect of forwarding has to be handled by us individually(that's why the manual change of the MAC address).

For TCP packets, the TCP checksum is also needed to be recalculated if any field of `iphdr` is changed in the packet so a TCP checksum function[27] is called as well to calculate the new TCP checksum and then the field `tcphdr_var->check` is replaced by it.

For ICMP (ping) packets, as there is no TCP header, after changing either the source address or the destination address, only the IP checksum needs to be recalculated.

For UDP packets, the IP checksum is calculated after changing the source/destination address and then from `linux/udp.h` the `udphdr->check` is calculated using the UDP checksum function[28] or the value is set to be 0. A 0 UDP checksum informs the receiver that the UDP checksum of the packet is not used so no need to verify it.

4.4 Using The GPU

The diagram below describes the process of how the GPU was used in the firewall.

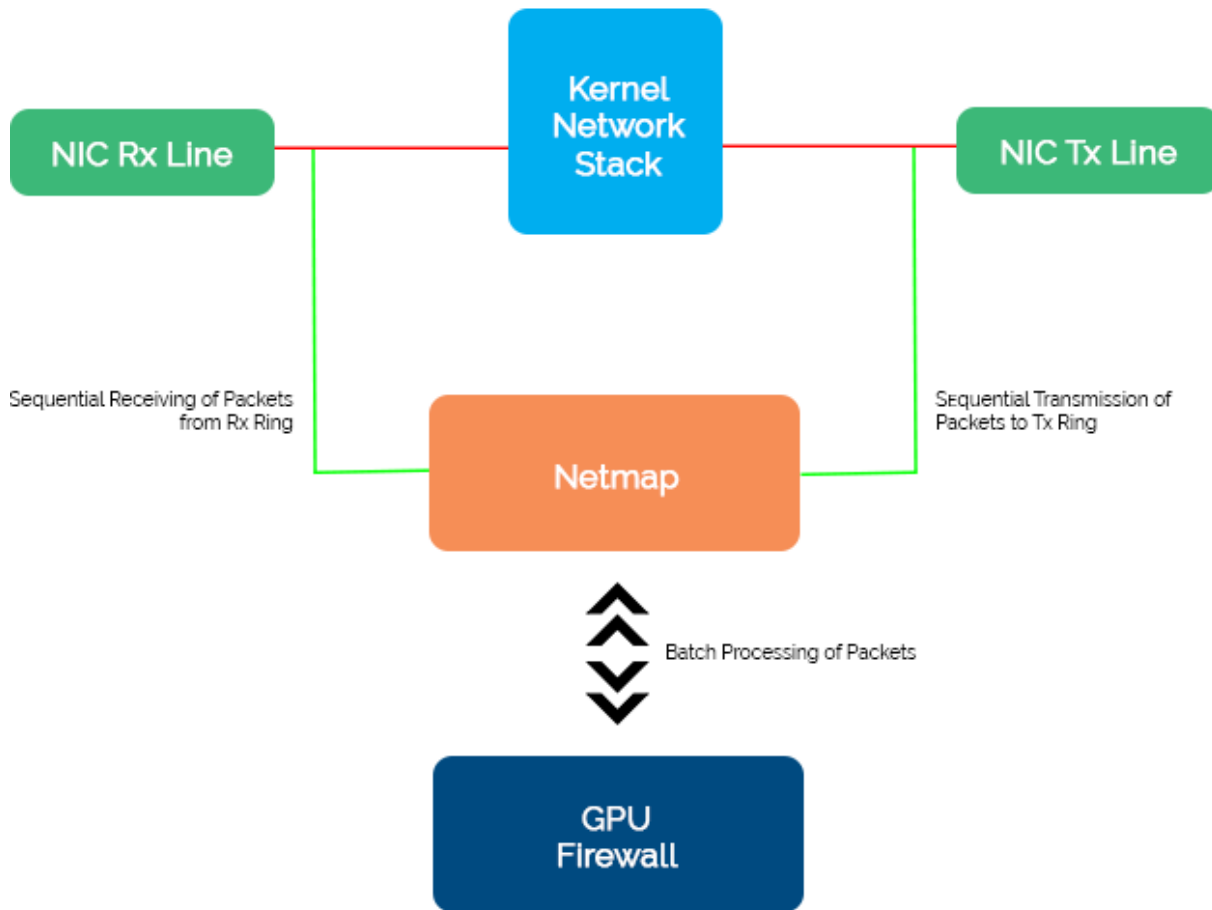


Figure 4.2: GPU Use in The Firewall

The packets are sequentially buffered from the receive ring using the sequential networking code and the packets are then sent to the CUDA code for batch processing(parallel processing); then the packets are sent back to the sequential code where they are then sent to the transmit ring.

4.4.1 What processing is done on the GPU Currently?

Currently the GPU handles the NAT(both SNAT & DNAT) of each packet and recalculating the IP & TCP checksum of each packet. All the three operations are done in a single thread and a single thread handles a single packet.

Further plan is to add the operation of encryption on top of NAT & checksum calculation in each thread of the GPU. As also observed in GASPP[4], if more operations are done on the GPU, the throughput will be higher, compared to the throughput when only 1 or 2 operations are performed on the GPU(as doing more operations on the GPU will compensate for the delay caused by memory copies from the host to the GPU and from the GPU to the host).

Chapter 5

Challenges Faced

There were multiple reasons why the packet after being processed by our firewall were getting dropped at the receiver.

- **Incorrect Checksum:** Initially we were sending out the packets without recalculating any checksum and therefore those packets were getting dropped at the receiver side due to incorrect checksum. This problem was recognized quite early and fixed by recalculating the respective checksum of the underlying protocol.
- **Incorrect Source MAC Address:** Initially our tests ran on ICMP packets which after the checksum error was resolved were correctly being received by the receiver. But as soon as we shifted to sending out TCP or UDP packets, even after calculating the IP and TCP/UDP checksum, the packets were getting dropped. We soon realized that the source MAC address is also needed to be changed to the default gateway's (firewall's) public one. If the source MAC address is not changed, the combination of source IP address & source MAC address the receiver expects and the source IP address & the source MAC address the receiver gets have a mismatch and hence the packets gets dropped. This problem was also fixed soon after switching to testing with TCP & UDP connections. Surprisingly, if any MAC address is modified, there is no need to recalculate any checksum (IP or TCP/UDP).
- **Connection Handshake Handling:** For TCP connections, the packets involved in the three-way handshake (SYN, SYN-ACK & ACK) needed to be handled sequentially rather than using the GPU. So any handshaking packets are processed sequentially rather than on the GPU.
- **Rewriting of External Library Functions:** Functions like `inet_addr` and `ntohs`, `htons` were needed to be coded again as part of a custom header file because CUDA uses a compiler called `nvcc` (Nvidia CUDA compiler) and `nvcc` can't compile code which is using any functions which are defined in an external library. The functions were coded again and stored in a separate header file and was linked with `nvcc` by creating a separate object file of it and then linking all the object files together while calling `nvcc`.
- **Handling Miscellaneous Packets:** As netmap blocks any packet going to the host stack, there were certain packets that weren't required to be handled by our firewall and were required to go to the host stack such as ARP (Address Resolution Protocol) or LLNMR (Link Layer Multicast Name Resolution) packets. For packets that need to go to the host stack, netmap has a provision to add a flag in the packet's ring's slot+ as `NS_FORWARD` which sends the packet to the host stack for processing. Before finding out

about this, the firewall used to stop after processing for a certain amount of time(due to the ARP packets not being processed and hence the sender not finding a suitable IP address to send the packets to). After solving this problem, the firewall doesn't stop working in the middle.

Chapter 6

Related Work

GASPP [4] is a GPU-accelerated stateful packet processing framework for network traffic processing applications such as flow tracking, TCP stream reconstruction and stateful processing of network packets in buffer. The design, implementation and evaluation of GASPP was also discussed in it. It demonstrates the need and possibility of using GPUs for use in network traffic processing. But GASPP [4] does not talk about the stateless processing of packets using GPU which is what we are trying to achieve.

PacketShader:GPU accelerated software router [3] is a high-performance software router framework for general packet processing. The paper talks about the effectiveness of GPU for packet processing with router applications such as IPV4/6 forwarding, OpenFlow switch and IPsec tunneling. It also does not talk about stateless packet processing.

GPUstore: Harnessing GPU computing for Storage Systems in the OS Kernel [2] provides a framework for integrating GPU computing with storage systems in the kernel.

Firewall Engine based on Graphics Processing Unit [14] focuses on creating parallel algorithms written in CUDA for parallel packet filtering and threat detection. Although our project does not solely focus on algorithms, it is good to know that there exists scope for it.

Chapter 7

Future Plan

There remains some major functionalities that are needed to be added to the firewall.

1. Currently the firewall is being built in a Ubuntu machine. We wish to expand the functionality of our firewall to a cheap low-powered device like the Raspberry Pi which also has an integrated GPU. This is important, as if we are able to deploy the firewall on a Raspberry Pi then we will have a cheap & lightweight portable router on our hands which would be secure as well as easy to deploy. For this the OpenCL framework is needed to be used to harness the parallel capabilities of the onboard GPU on the Raspberry Pi and the CUDA code rewritten to follow the specifications of OpenCL.

2. Our future work also involves creating two VPN gateways for the firewall (one for encryption & one for decryption), parallelizing the AES encryption/decryption (Galois/Counter mode) using the GPU and then comparing it with the solutions that are currently available in commercial & open source firewalls.

4. Long term goal is to enable parallelized stateless firewalling in SDN switches.

I hope to achieve concrete results in some of them by the end of my Bachelors.

Chapter 8

Conclusion

In this thesis report we discussed about the functioning of a firewall, the differences between a stateful & a stateless firewall & the need for a GPU accelerated firewall. Some background was given on the OSI model & the different layers in the Internet Protocol Stack, on how networking happens in an operating system & how a packet is created and handled by the operating system & the different types of protocols that are used to send data.

Information about Netfilter, CUDA, netmap and the features of a firewall was also given.

Thereafter our approach to the solution of the problem was described and the various challenges we faced while developing the firewall and how we solved them. Some information was given on related existing work that has been done on the problem & what is the future plan and features we plan to implement in the firewall.

Bibliography

- [1] [https://en.wikipedia.org/wiki/Firewall_\(computing\)#/media/File:Firewall.png](https://en.wikipedia.org/wiki/Firewall_(computing)#/media/File:Firewall.png)
- [2] *GPUstore: Harnessing GPU Computing for Storage Systems in the OS Kernel*
Weibin Sun, Robert Ricci, Matthew L. Curry - *SYSTOR 2012*
- [3] *PacketShader: a GPU-Accelerated Software Router*
Sangjin Han, Keon Jang, KyoungSoo Park, Sue Moon - *SIGCOMM 2010*
- [4] *GASPP: A GPU-Accelerated Stateful Packet Processing Framework*
Giorgos Vasiliadis, Lazaros Koromilas, Michalis Polychronakis, Sotiris Ioannidis - *USENIX ATC 2014*
- [5] https://www.pcworld.com/article/202452/why_linux_is_more_secure_than_windows.html
- [6] http://www.inetdaemon.com/tutorials/information_security/devices/firewalls/stateful_vs_stateless_firewalls.shtml
- [7] https://ark.intel.com/products/126699/Intel-Core-i9-7980XE-Extreme-Edition-Processor-24_75M-Cache-up-to-4_20-GHz
- [8] <https://www.nvidia.com/en-us/about-nvidia/ai-computing/>
- [9] *On the performance of GPU public-key cryptography*
Samuel Neves, Filipe Araujo - *ASAP 2011*
- [10] https://link.springer.com/chapter/10.1007/0-387-34189-7_4
- [11] <https://www.netfilter.org/>
- [12] <https://developer.nvidia.com/cuda-zone>
- [13] <https://code.google.com/archive/p/kgpu/>
- [14] *Firewall Engine based on Graphics Processing Unit*
Abhaya Kumar Sahoo, Amardeep Das, Mayank Tiwary - *ICACCT 2014*
- [15]:<http://www.informit.com/articles/article.aspx?p=373120&seqNum=2>
- [16]:<http://blog.aeguana.com/2015/07/12/tcp-socket-states/>
- [17]:<https://www.netfilter.org/documentation/HOWTO/netfilter-hacking-HOWTO-3.html>
- [18]:<https://nyu-cds.github.io/python-gpu/02-cuda/>
- [19]:<https://www.tomshardware.com/reviews/nvidia-cuda-gpu,1954-7.html>
- [20]:info.iet.unipi.it/~luigi/netmap/
- [21]:<https://www.freebsd.org/cgi/man.cgi?query=netmap&sektion=4>
- [22]:<https://github.com/luigirizzo/netmap>
- [23]:<https://www.computerhope.com/jargon/p/packet.htm>

- [24]:<https://www.comparitech.com/blog/vpn-privacy/nat-firewall/>
- [25]:<https://github.com/luigirizzo/netmap/blob/master/apps/bridge/bridge.c>
- [26]:http://minirighi.sourceforge.net/html/ip_8c.html
- [27]:http://minirighi.sourceforge.net/html/tcp_8c-source.html
- [28]:http://minirighi.sourceforge.net/html/udp_8c.html
- [29]:<https://www.caida.org/research/traffic-analysis/tcpudpratio/>
- [30]:https://netfilter.org/projects/libnetfilter_queue/