A Novel DDoS Floods Detection and Testing Approaches for Network Traffic based on Linux Techniques

Muhammad Tahir*[1], Mingchu Li[1], Naeem Ayoub[2], Usman Shehzaib[3], Atif Wagan[4]

[1]School of Software Technology, Dalian University of Technology, (DUT), Dalian, Post (116621), P.R. China
[2]School of Computer Science & Application Technology, Dalian University of Technology, Dalian, P.R. China
[3]Dept. Of Computer Science, COMSATS Institute of Information Technology, Lahore, Pakistan
[4]School of Computer Science & Eng., Nanjing University of Science & Technology, Nanjing, P.R. China

***Abstract*—In Today's Digital World, the continuous interruption of users has affected Web Servers (WSVRs), through Distributed Denial-of-Service (DDoS) attacks. These attacks always remain a massive warning to the World Wide Web (WWW). These warnings can interrupt the accessibility of WSVRs, completely by disturbing each data processing before intercommunication properties over pure dimensions of Data-Driven Networks (DDN), management and cooperative communities on the Internet technology. The purpose of this research is to find, describe and test existing tools and features available in Linux-based solution lab design Availability Protection System (Linux-APS), for filtering malicious traffic flow of DDoS attacks. As source of malicious traffic flow takes most widely used DDoS attacks, targeting WSVRs. Synchronize (SYN), User Datagram Protocol (UDP) and Internet Control Message Protocol (ICMP) Flooding attacks are described and different variants of the mitigation techniques are explained. Available cooperative tools for manipulating with network traffic, *like;* Ebtables and Iptables tools are compared, based on each type of attacks. Specially created experimental network was used for testing purposes, configured filters servers and bridge. Inspected packets flow through Linux-kernel network stack along with tuning options serving for increasing filter server traffic throughput. In the part of contribution as an outcomes, Ebtables tool appears to be most productive, due to less resources it needed to process each packet (*frame*). It is pointed out that separate detecting system is needed for this tool, in order to provide further filtering methods with data. As main conclusion, Linux-APS, solutions provide full functionality for filtering malicious traffic flow of DDoS attacks either in stand-alone state or combined with detecting systems.**

Keywords*—DDoS attacks; floods detection; Linux-APS architecture; mitigation techniques; network traffic; Netfilter; testing approaches

I. Motivation and Introduction

Through the development of information and communication technology (ICT), our societies become global information societies with all-around smart computing environments, but unfortunately, the-security systems and policies that regulate this environment are not accelerated as needed. Attackers do not use the quarry in various vulnerabilities found in applications running on systems.

Among the various cyber-attacks, (such as: SQL Injection (SQLi), session recording, internal site scripting attack, Denial-of-Service (DoS) attack, have become the most threatening attacks so far, as very few methods have managed to mitigate these attacks problems). Distributed Denial-of-Service (DDoS) attacks, aggressors do not use a particular crowd on behalf of their attacks however a cluster of numerous loads or uniform hundreds of central processing units (CPUs), towards fixing an SYN (synchronized) attacks. After the creation of development solutions towards resolution the incidence of attacks stimulated the development of the attacks themselves. Currently, DoS and DDoS attacks have been outdated via DDoS attacks. The World Wide Web (WWW) Safety FAQs, scheduled DDoS declares that [1].

"A DDoS attack, usages numerous supercomputers towards promotion of a synchronized DoS attack beside one or further goals. By client and server machinery, the criminal is intelligent to increase the efficiency of DoS, pointedly by connecting the properties of numerous unknowing accessory supercomputers which help as attack stands. Classically a DDoS, main suite is connected the one (CPU, storage, memory, by a filched version). The main suite, on a selected time, before transfers to some digits of ***'proxy'*** plans, connected on CPUs, everywhere on the Internet. The proxies, once they obtain the -command, recruit the attack. By client and server know-how, the main suite can pledge hundreds or else level thousands of proxy series inside ages."

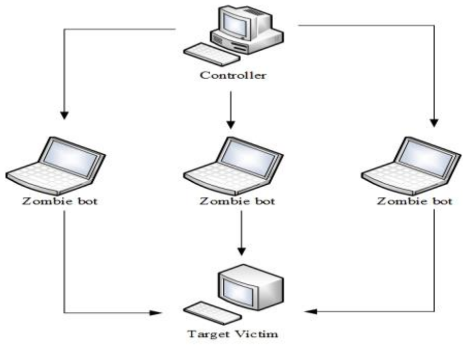


Fig. 1. Provides a clear idea about DDoS attacks.

Fig. 1 proves two main aims that make DDoS attacks, eye-catching to intruders. Firstly, there are effective automatic-tools to attack all victims [2]. That is, experience is not necessarily required. Secondly, that one is typically dreadful toward find an aggressor lacking general communication with a person or else lacking novel roles fashionable utmost the Internet routers [3].

According to *Akamai-state* of the Internet security reports that the frequency of DDoS attacks has been increased by **131%** Worldwide in **2017**. Along with recent event, involving the *WannaCry ransom-ware* attacker, (also known as *WanaCrypt0r & W-Cry Ransom ware*) malicious software spreading security and protection questions for Internet users are becoming more important than ever before.

The importance of this research work is to find out and testing most efficient and reliable tools existing in Linux-based systems for filtering and aggregation DDoS attacks. Linux-based solutions are considered as bit and easy to configured systems among available competitors.

Most common WSVRs, focused DDoS attacks, will be taken into consideration such as SYN, UDP and ICMP floods. By configuring filter servers and applying suitable setup, most efficient and reliable solution will be chosen. Aggregation of data traffic flow will be considered from a point of impact on filtering productivity.

- *This research work, is divided into following four parts:*
- ***Firstly,*** the overview of the DDoS attack is provided.
- ***Secondly,*** literature analysis & background gives a brief survey of research going in the area of most common types and methods of mitigations of DDoS attacks and available Linux-based solutions for data traffic filtering and aggregation. After going through the literature analysis and its background.
- ***Thirdly,*** the problem statement and related work has been identified and will describe experimental network components and kernel tuning.
- ***Finally,*** it focuses on the conclusion of the work that will include implementation of the selected solutions on data filter servers, differentiated by installed hardware.

II. Literature Analysis and Backgroud

DDoS attacks have become more common and fashionable in recent years. Large-scale systems of septic computers ***'zombie/bot'*** trust their processing plus processing capabilities to overload public service and deny it to authentic users.

The attacks on major *e-commerce locations* in February **2000** and attacks on origin domain name system (DNS), service **2003** and **2007** made community devotion towards the-problematic of DDoS attacks [4]. Nowadays, medium-volume Web pages are generally condemned through crooks toward getting defense from their venders. These are also deprived of appealing [5]. Additionally, Internet Service Providers (ISPs), must address the problem that DDoS traffic and increases the bandwidth of the communication channel.

The first tools such as Tribe Flood Network (TFN) [6], Stacheldraht, Trinoo or Mstream [7], have used communication structures without cryptography and were organized hierarchically. Best of these implements recycled TCP, SYN, UDP, and ICMP floods using potentially recognizable limits.

As several of these attacks have been positively relieved, an innovative formation of robots takes developed Spartan-dominion Robot (SDBot), Agobot, commonly used Phatbot and well-known representatives using Internet Relay Chat (IRC), as a secure link [8]-[10]. These tools also include distribution methods besides take additional refined attack algorithms that can stay updated concluded the Internet.

Sudden mitigating techniques of DDoS attacks, on the Internet sources or on kernel seems impossible owing towards the circulated and authoritative environment of the Internet protocol IP, based system network. Several methods were suggested to search for the original IP address, of the attacker using filtering mechanisms.

Internet Engineering Task Force (IETF) documented filter entry approach is defined in request for comments **(RFC: 2827)** [11]. And assistances mitigate DDoS attacks, through IP spoofing, which indirectly deal with different types of network misuse, causes Internet traffic to control source. Network filter-is a ***'good neighbor'***. This policy is based on mutual cooperation between ISPs, for their common benefits.

In order to avoid manipulation of IP addresses, Park et al, [12] Proposed packet filters distributed on standalone systems over the Internet are to be stopped packets of counterfeit IP addresses.

Suspenseful Savage et al. [13] is recommended that IP Trace back find the basis of fake IP addresses using probabilistic labeling packages. Song et al. [14] provides an extended scheme for probabilistic packet selection to reduce the frequency of false positives to restore the attack path. Another improved scheme for probabilistic labeling of packages was proposed by Bellovin et al. [15] to reduce the cost of calculation. This ICMP is a tracking system similar to a probabilistic scheme for labeling packages.

In this system, the routers generate ICMP, packets at the Low-probability destination. For a significant data traffic flow, the recipient can gradually restore the route made by the packets in the leak.

Mahajan et al. [16] provide a system where routers learn fixed costs to provide good traffic from bad traffic. Siris et al. [17] represents the variance finding algorithms on behalf of identifying TCP and synchronized SYN attacks.

Modifying threshold algorithm and a specific request of the total amount of algorithm is for finding of a switching point. Modifying threshold algorithm associates with number of the SYN packets established over an estimated number of predetermined intervals, founded on the new dimensions. Towards promotion of increases, a panic inception should remain overdone in series. The, Cumulative Sum Control Chart (CUSUM), algorithm usages the change among the sum of SYN packet per time pause besides the number valued on behalf of the similar range such as ***'Gaussian'*** random variable. Then, flood attack SYN, is sensed by consuming the total amount built on the probability ***like;*** instantaneous significance due to the change in the average velocity of the circulation.

Several authors, including Wang et al., [18] has proposed a method for detecting SYN flood attacks. These are built and going on the construction of TCP, SYN, FIN flags and Rapid Spanning Tree Protocol (RSTP). Protocols on leaflets connect the final nodes to the Internet. There is a change in the sum of SYN, FIN packets perceived rest (RST) flag set and CUSUM, algorithm is used to detect the switch point. Towards decrease the influence of changed entree designs at diverse locations, the change among the amount of: (SYNs), (FINs) and (RSTs) remains controlled with the calculated typical by (FINs) and (RSTs).

Luo et al. and others [19] Also proposed a system for detecting Pulsing Distributed Denial-of-Service (PDoS) and DoS attacks on expressive target's WSVRs, with **(t\¥0)** variances produced by PDoS attacks, specifically jitter in inbound documents traffic flow and reduction of outward---bound TCP, and acknowledgement (ACK) data networks traffic flow.

Cabrera et al. [20]. Detecting DDoS using Management Information Base (MIB), traffic flow valuables on behalf of the aggressor and destination. Appropriate autographs were identified towards detect attacks from known attacks. On the side of the attacker, the DDoS attacks must be detected before it is launched by identifying precursors based on MIB.

In addition to previous work, intrusion detection systems and firewalls are one of the best security systems that work by pairing packets with predefined rules and filtering them accordingly. Linux-kernels had a **1.1** packet filter. At the end of **1994**, kernel-hacker Alan Cox carried the IP firewall from Berkeley Software Distribution (BSD), to Linux. In mid-**1998**, Rusty Russell et al. [21] and others revised most of the networks under the Linux kernel of the **2.1** development series and introduced the IP chains user space tool.

The Linux-kernel that preceded this had some very serious disadvantages with some main working functionality. The old Linux-firewall code does not apply to fragments. The **32-bit-** counter (*at least on Intel*), does not allow to enter other protocols than TCP, UDP and ICMP.

It cannot make major atomic changes that can not specify ***'Inverse Functions'***. It has little and can be tricky to handle (*which makes it prone to user failure*).

Russell's work redefined radically the Linux-network layer and enabled filtering of kernel-packets in user-space (*which simplified usage, configuration, and security*).

Finally, the next advanced generation of tools, ***"Ebtables-Iptables"*** and another core transcript took place in mid-**1999** for Linux **2.4**. The enhancement initiated in **2.2** continued, and the extensive Linux arsenal of networking tools. The core is rewritten as ***'Netfilter'*** [22].

In this research article, we proposed different Linux-based resolution approaches and testing simulations for network traffic, filtering data packets to protect against DDoS attacks using cooperative ***"Ebtables"*** and ***"Iptables"*** tools, mitigation techniques, and Linux-based resolution lab design firewall architecture.

III. Problem Statement and Related Work

- *Problem No.1.* The DDoS attacks, stands a rigid challenge towards mark and connected packages inaccessible near all users, frequently via briefly disturbing before interrupting sudden services from their domain server. Originally based on target service resources limitation, DDoS attacks can be done by either spoofing attacker IP address or using so called: ***'Botnet'***. Botnet is a network of hacked devices, connected to global network which control is gained by third party. Compromise devices can send data traffic to target services which makes DDoS mitigation complex. As it is hard to distinguish authentic malicious traffic flow from the DDoS attacks.
- *Problem No.2.* The DDoS attacks, stance has a main warning to the internet. This must be reduced the class from the internet service. This is significant because the internet has now become a critical resource whose violation has financial consequences or even terrible consequences for human security. More and more critical services use the internet for daily work. DDoS attacks do not just mean losing the latest game results of the environment. This could malicious traffic flow from DDoS attacks losing an effort on the item you want to buy or lose to your customers for a day or two under attack. So this one is significant in the direction of consume funds towards stop or else moderate the system.

Smart breaks support unbroken competition among the aggressors and the protectors. Passing of these remain a real ramparts beside a confident category of attack. The aggressors' revolution devices finding an approach toward avoid this defense.

In addition, since absolute protection is not feasible, developing effective defense framework involves an often complicated set of Trade-offs:

For this purpose, numerous solutions have been proposed as discussed in literature analysis background. ***'Firewall'*** is one of them. By using a dedicated firewall we can block all the IP addresses that are flood the server to consume its resources.

According to Radware's: **(2016-2017)** Global application and network security report [23]. The most common types of attacks were **2016** (e.g., UDP, SYN and ICMP Flood).

A. *Distributed Dinel-of-Service (DDoS) Flooding Detection - Tools & It's Mitigation Techniques*

1) *The Synchronized (SYN) Flood Detection:*

Attacks of this kind targeting three-way TCP, link mechanism that sends connection requests faster than the target computer that can handle them, causing network capacity [24].

In common situations, the client method is activated through distribution of SYN junk messages toward affecting server.

The affecting server formerly confirms effective SYN junk messages by distribution and SYN acknowledgement (ACK) messages toward sudden client.

Affecting client server formerly complements impressive configuration of the linking, responds by an ACK message.

This opens powerful link among the client and server then provides capacity information container to be swapped among sudden client plus server. **Fig. 2(a)** displays three-way appearances regarding junk message flow:

A potential weak point is that when the server system sends the confirmation of (SYN-ACK) towards the client, however it does not consume all data so far to acknowledge (ACK-Message). It is called a partial built-up linking.

The SYN server made a data structure (Queue-SYN) in system memory, relating all incomplete connections [25].

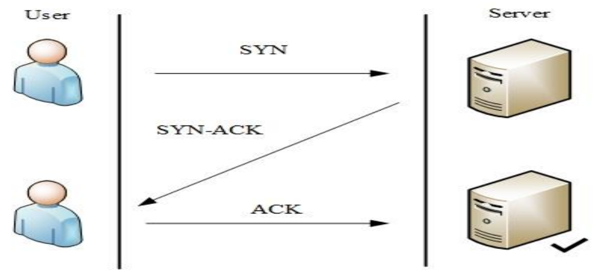


Fig. 2. (a) Three-way handshake mechanism.

This data structure has a problem dimension and it can be transmitted through purposefully generating as well numerous moderately sweeping networks. Generating sweeping networks remains calm by IP spoofing. Suspenseful attacker drives (SYN- Junk Messages) toward the quarry's server systems. They seem genuine, but they actually refer to a client system-that cannot reply to (SYN-ACK-Messages). These resources are used in order to terminate (ACK-Messages) spirit.

It is not once being directed near the quarry's server systems, as per displayed in **Fig. 2(b).** The figure displays norms of SYN flood attacks:

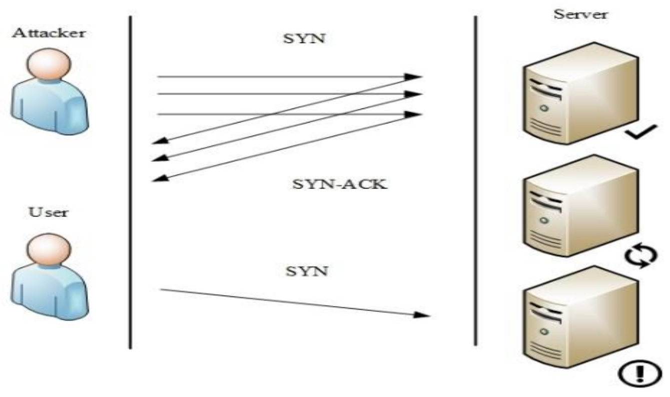


Fig. 2. (b) Norms of SYN flood attacks.

Among possible solutions for mitigation, these kinds of attacks are using firewalls with (SYN-Cookie). This cookie feature enabled, filtering by limitation of possible (SYN-Packets) per second passing accepted by the server and blocking attacker source IP addresses.

The (SYN-Cookies) is the technique by which the original (TCP-Sequence) numbers will be selected by the (TCP-Servers). These are designed to mitigating the SYN flood attacks.

The main differences are between the original sequence number that created by the server and client are:

- ***Highest 05 bits:*** T mod 32, wherever T stands a 32 bit device hostage, increasing each 64 seconds;
- ***The following Subsequent 3-bits:*** Is a programming of a ministry of state security (MSS) designated via the client server cutting-edge reply near client MSS;
- ***Lowest 24 bits:*** Sudden server-designated classified purpose regarding sudden client IP- address also port no. of sudden server IP address and port no. and **T**.

Therefore, sudden client server using SYN cookies, files should no more delete networks after the situation SYN queue is full. This one will return SYN+ACK, by way of the SYN queue was higher.

After tense server obtains an ACK message, its authorizations whether the Top-secret function workings aimed at the last significance of **T**. Besides reconstructs of the SYN queue has been scheduled to encrypt Maximum Segment Size (MSS).

Restrictions can be done based on exact server statistics, including average traffic rate, connections per second during specific time.

2) The User Datagram Protocol (UDP) Flood Detection:

The UDP, is an offline network protocol that provides data integrity check numbers and port numbers for addressing functions [26].

If there is no first handshake, there is no guarantee of data transfer, sorting or duplication of protection to establish a valid connection.

Therefore, a lot of best-staked traffic can be sent over UDP channels to any host without built-in security to limit UDP, DDoS throughput.

This means that not only UDP, thread attacks are very effective, but they can also be done with relatively few resources.

Through UDP flood attack, attackers direct a huge amount of UDP packets towards the offer system, resulting in network saturation and bandwidth reduction available for authentic quote requests. Once ***'rib-tickling'*** offer structure obtains the UDP packets, it controls which request is to come for electrifying endpoint docks.

After the server decides so that the application is not spoofing to the application, it will generate an: ***"Unavailable"*** ICMP packets for the fake home address. Unknown sufficient UDP packets are brought toward the victim's docks. Sudden system resolve will be continued to reduce.

Another way to perform an attack is to send huge amount of UDP packets to certain the ***'Opened-Ports'***, leading to link bandwidth exhaustion.

Among known option of mitigating is to close all unused port on server and filter all incoming traffic that is destined to target server, except Domain Name Servers (DNS).

3) The Internet Control Message Protocol (ICMP) Flood Detection:
The ICMP, is recycled by devices, including the router, to send operating evidence messages to support networks for example, diagnostic or control drives [27].

An attack using ICMP Flood can be performed in the ***"ping of dead"*** mode and sends victims a large number of the ***"ICMP_ECHO_ REQUEST"*** messages that cause the target server to respond.

Thereby it leads to saturation of the network connection with the victim's behavior.

During the ICMP flood attacks, the foundation IP address might be manipulated. Possible solutions for mitigations are limit size of ping requests as well as the rate at which they can be accepted and denied all ***"Icmp_Echo_Requests"*** from all source IP addresses, except local network.

B. Linux-Based Resolution Availability Protection System (Linux-APS) Architecture

In this section, we present the Linux-based resolution lab design architecture for DDoS network control.

While deciding which best resolution is to choose for traffic filtering and DDoS attacks mitigations.

Linux-kernel based products can become a preferable for considering price tag and user friendly environment.

There is lot of tools and tuning options for working with traffic, on which we need to, have a closer look:

Linux distributions are Operating Systems (OS), created from a collection of software [28].

These are based on the Linux-kernel and packet management system.

The Linux kernel is a CPU suite and it is the essential for the OS, through full controller, especially utilized in our given method to be able to apply the suitable techniques for filtering traffic.

It is a crucial to know, how incoming and outgoing packets of Linux network stack are being handled by Linux Protection System (LPS).

Linux-based resolution lab design architecture provides clients and users with a chance to discover the LPS.

This lab design architecture enables clients and users to deliver demos to execute their own-due meticulousness in our given proposed testing methods of surroundings.

With our demonstration tool end-users will gain: additional information of the Linux-based availability protection system (Linux-APS) and it also help users to fast-track safe data on WWW.

In this connection, **Fig. 3** provides protection solution to assist sense and mitigation techniques against DDoS attacks and cutting-edge malicious threats.

Linux-based (APS) mitigates accessibility threats. *Such as:* web application-layer DDoS attacks previously they influence WSVRs, accessibility.

So we compromises a lab setting to simulate a tier **1** web application and DDoS attacks, which can be mitigated by the Linux-based (APS) system.

The lab enables web domain partners to fully understand the APS, system and provide informative demonstrations for their clients and users.

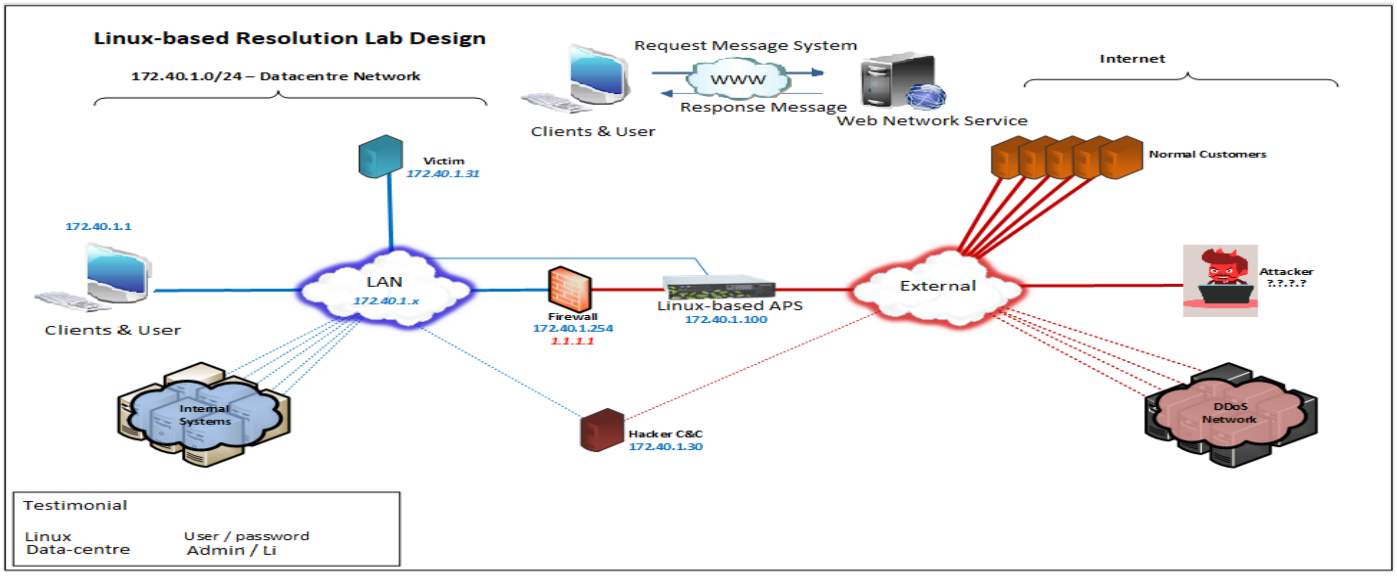


Fig. 3. Linux-based resolution lab design architecture.

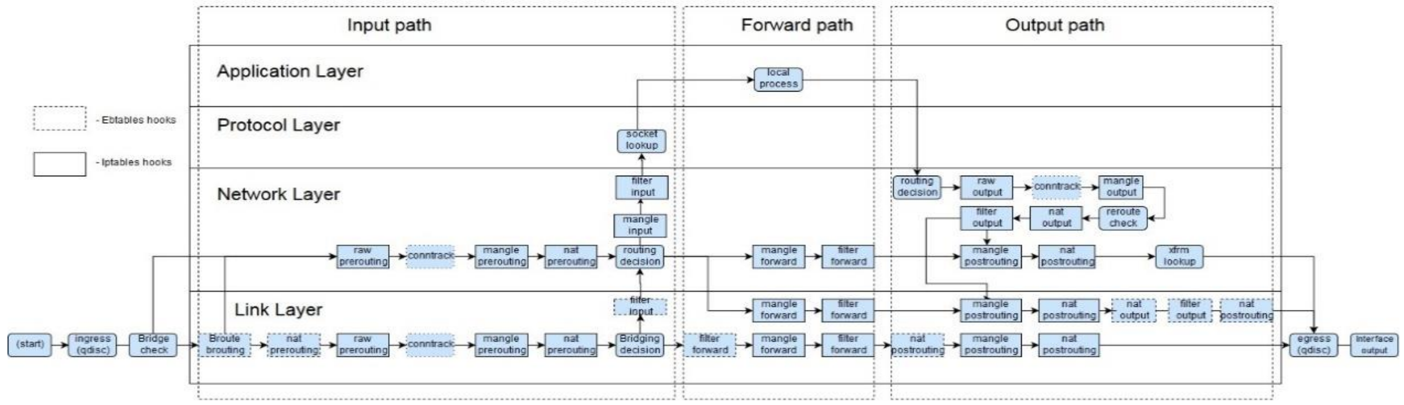


Fig. 4. Data packets flow over Netfilter.

C. The Netfilter Data Packet's Flow

Disreputable for all manipulating with network stack in Linux kernel is Netfilter. To be able to apply the suitable techniques for filtering traffic, it is crucial to know, how incoming and outgoing packets of Linux network stack are being handled by Netfilter. Netfilter is the infrastructure that is provided by the Linux-kernel and it is also a set of hooks inside the, enabling single core modules to record call-back features with the core network stack [29]. Which you can see in a general packets flow path through Netfilters hooks. It is displayed in **Fig. 4.**

Now, let's have closer look at the most essential functions of our testing purposes:

- ***Classless Queuing Disciplines (qdiscs):*** It is the scheduler and the major building block in Linux - traffic-control process. It queues all packets based on appropriate queuing discipline and transmits the packet as soon as it can. There is ingress the (*inbound traffic*) and egress the (*outbound traffic*) *qdiscs*. The default queuing discipline for all interfaces under Linux is ***'pfifo_fast'***. It is based on a conventional first in, first out (FIFO) qdisc and provides prioritization. There are three different bands for separating traffic. The highest priority traffic is placed into the band **0** and is always serviced first.

- ***Bridge Check:*** It is simply checked, if the interface, from which packet was received, that may belong to bridge or not. If so, frame will not be processed at this point and will be sent to bridging decision function.

- ***Conntrack Tools:*** It is the connection tracking subsystem. It supplies evidence around the formal of a-linking, including foundation and endpoint IP addresses, docks digit sets, procedure forms, public, and recreation in structured memory [30].

- ***Bridging Decision:*** At this point frame is being investigated whether its destination is local process or its endpoint media access control (MAC) address is-located on extra sideways of the bond. This can be done with four different frame things:

1. Bridge it.

2. Flooding it over, if the endpoint media access control (MAC) address is unidentified to the bond.

3. Permit this to developed procedure encryption (IP-code).

4. Disregard it, condition an endpoint media access control (MAC) address stands on the similar cross of the bond.

- ***Routing Decision:*** Based on IP address it decides if packet destination is local process or it should be forwarded. Packet will be sent through bridge interface at this point, if forwarded.

D. Proposed Linux-based Resolution Tools and Mitigation Techniques for Web Servers (WSVR's)

a) Ebtables Tool:

The Linux kernel is an integrated filtering tool, starting with core version 2.2 [31], [32] which allows you to configure and to maintain control tables that control Ethernet frames. This is similar to iptables tool, but with less functionality, because the Ethernet frame header is less complex than that of the IP packet header. Ebtables rules are working only with bridged frames and compare to iptables, the frame is processed earlier in the stack, consuming less resources.

There are three tables named ***filter***, ***nat*** and ***broute*** [33]. Syntax for managing with ebtables rules is the same as with iptables rules:

- The ***filter*** is the default table and contains three embedded strings: INPUT (for frames - intended for the bridge), OUTPUT (for locally generated frames or (b) and FORWARD (for frames sent by the source).

- The ***nat*** is usually used to change media access control MAC addresses and contains three - embedded strings: PREROUTING (to change frames as soon as they arrive), OUTPUT (to - change locally- generated or (b) routed to their bridge) and POSTROUTING they will leave.

- The ***broute*** table has a built-in chain: BROUTING. The goals for DROP and ACCEPT are of particular importance in the broute table. DROP means that the frame must be routed, while ACCEPT means that the frame must be locked. The BROUTING chain goes very early. However, it only passes through the frames entering the bridge port, which is in the forward-looking state.

b) Iptables Tool:

Netfilter iptables is a user-space command line utility to configure packet filtering rules [34].

It's the default firewall management utility on Linux-systems.

Iptables exists recycled to configure, preserve plus validate the IPv6 packet refine control tables cutting-edge the Linux-built kernel.

There are four different tables ***filter***, ***nat***, ***mangle*** and ***raw.*** All tables' covers a digit of integrated-chains and might similarly cover well user-defined-chains.

All new chains are lean of comments that may correspond to a fixed of packets. Also every instruction sets a goal: [35].Which means action with matching packets.

- ***Filter:*** Standard tables, contains built-in INPUT chains (for local-purpose plugs), FORWARD (on behalf of packets sent over sudden field), and the OUTPUT (On behalf of nearby produced- packets).
- ***Nat:*** The nat-table, while a packet is found so that makes an original linking. It involves like three diverse integrated modules: PRE-ROUTING (To modification packets when they will- enter), OUT-PUT (To change packets created locally before routing) and POSTROUTING (To change packets when they-expire).
- ***Mangle:*** The mangle-table remains recycled on behalf of particular data-packets changes. From the **02.04.19** kernel, ternary added embedded chains are moreover maintained: IN-PUT (On behalf of-incoming packets to powerful actual block), FORWARD (To change-packets that pass finished the-packet), and POST-ROUTING (To change packets once- drive-towards departure), PRE-ROUTING (Changes trendy for incoming pre-routing- packets) and OUTPUT (To change packets that are generated locally before routing).
- ***Raw:*** This table is primarily used to configure connection tracking exception along with the NOTRACK target. It is registered in higher priority Netfilter hooks, then stands so-called formerly an ***'ip_conntrack'***, before several further IP tables. Tendency delivers suspenseful next integrated new chains: PRE-ROUTING (On behalf of packets coming by some system edge) OUT-PUT (On behalf of packets produced via native methods).

c) ***The Link Aggregation:*** Link aggregation contains several methods for combining numerous system networks in similar to growth bandwidth.–Such as a linking can support then deliver termination cutting-edge situation of failure of one of the links [36]. Link Aggregation Group (LAG), syndicates a range of physical ports to create an only high-bandwidth-information channels near performance capacity distribution of traffic - between member ports and improve connection reliability. The channel aggregation principle is displayed in **Fig. 5.**

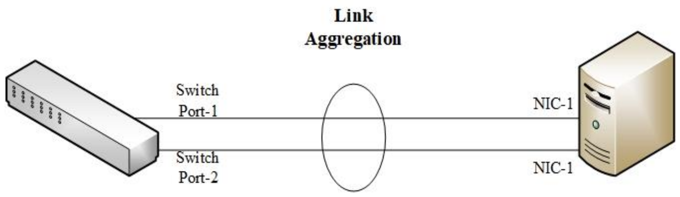


Fig. 5. The appearances of link aggregation.

Following are the most important benefits of link aggregation:

1) ***Multiple connections with little speed loss:*** Single file transfers or one-at-a-time transfers do not get much benefit from link aggregation, but multiple connections and file transfers do.

Some transfer rate increase might be apparent, but link aggregation perfectly works with multiple simultaneous transfers where several clients connect and download concurrently. Having more network lanes available allows all clients to encounter faster download speeds. Examples include a personal media server or network attached storage where multiple devices or users connect.

2) ***Redundancy:*** The physical links can be spread across multiple switches. If one switch fails or a cable is torn or disconnected, the transfer continues but at slower speed until the issue is resolved.

3) ***Load Balancing:*** This balances the network load across multiple network cards for more performance and better throughput. Rather than making one card do most of the work, let the other cards distribute the workload among them.

The Linux-built bonding connection delivers a technique on behalf of combining manifold system edges ***(Slaves)*** keen on a unique logically connected edge ***(Bond)***. Linux-built kernel provisions two bonding methods [37].

- The ***IEEE 802.3ad*** link-aggregation mode, and that tolerates single or else extra associates toward stay combined composed near method a link aggregation group (LAG), supposed that a MAC, client is able to delicacy the link-aggregation cluster by way of unknown draw stood a solitary relation.
- The ***balance-xor*** mode, somewhere the bonding of slave interfaces are static and fully slave interfaces are dynamic for load balancing and fault tolerance devotions.

IV. Discussion and Filter Testing

A. The Spirental Avalanche Commander Filter Testing Tool

Nowadays, it's essential that the performance of our ***'Network Infrastructure'***, ***'Security Systems'***, and ***'A-Web Applications'*** are carefully tested to ensure that performance goals are met. Heavy demands are placed on the- network by the emerging combination of voice, video, and data traffic, creating new challenges for client users and Digital Information Technology (DIT), staff and Web Networks.

The Avalanche can quickly identify potential points of failures by ***'stress-testing'*** the infrastructure. Large quantities of highly-realistic simulated user and network traffic can be generated, according to a wide range of real-world loading scenarios.

This proactive approach enables you to correct trouble spots and bottlenecks before network slowdowns or costly outages occurs [38].

All tests and measurements carried out on a specifically design and built network, equipped with network tester the ***"Spirent Avalanche 3100B"***, which allows generating traffic by **1Gbit/s** links. General view of network is displayed in **Fig. 6.**

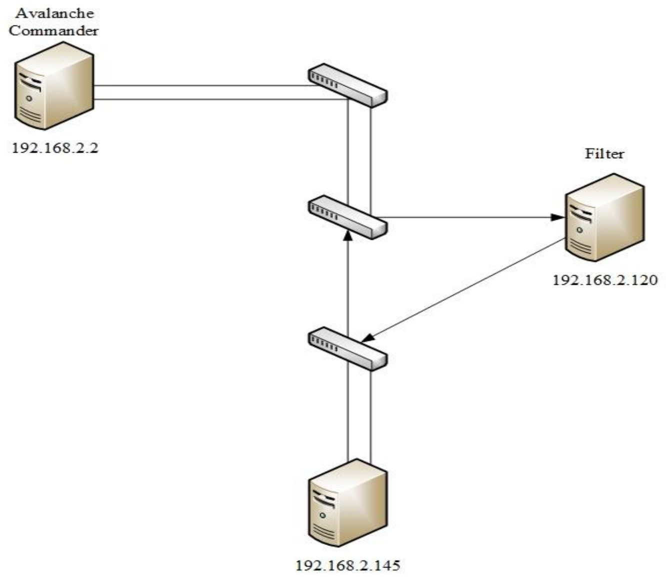


Fig. 6. Demonstrations of new testing network infrastructure.

The new testing filters **V1**, **V2** and **V3** are presented in the form of three separated server machines which are connected parallel. Same filtering tools will be used on all machines to compare hardware influence and link aggregation on filtering process. Hardware specifications for new testing filters are displayed in **Table I.**

TABLE I. THE NEW TESTING FILTERS SPECIFICATION

Requirements	**The New Testing Filter V1**	**The New Testing Filter V2**	**The New Testing Filter V3**
Number of Cores	4	16	2
Clock Speed(min/max)	2800/3600	1600/2200	3400
Network Interface Controller (NIC)	2x NC7782 Gigabit Server Adapter	2x 10-Gigabit X540-AT2	4x 82546EB Gigabit Ethernet
NIC Drivers	E1000	Ixgbe	E1000
Bus	PCI-X (66MHz)	PCIe(16)	PCI-X

- ***PCI-X:*** 64-bit parallel computer bus with theoretical maximum of the 1.06GB/s data exchange speed between computer processor and peripherals.
- ***PCIe:*** Is a serial point-to-point (P2P) connection bus with possible 4GB/s bandwidth in each direction.
- ***Ixgbe:*** Network interface card (NIC) driver with abilities to reduce the number of queues per interface-direction to the number of logical central process units (LCPUs).

The reasoning for this reduction is that each queue requires some over headed, and there is no advantage in maintaining for more queues between designed CPUs.

Because the filter software is used in one of the Linux-based distributions, ***'Debian (OS)'*** since it is entirely free software, most are licensed under the ***'GNU operating-system'*** and a (*Linux-distribution-General Public- License free software*).

In order to have traffic passing through filtering server, first bridge is needed to be configured. The configuration file for all interfaces is located at the */etc./network/interfaces*.

There are three network interfaces on current filtering server. One is serving for virtual private network (VPN) connection, remote control (RC), and other two services for carrying traffic.

B. The Link Aggregation and Interface Bonding

The bonding in place of link-aggregation essential exists maintained through in cooperation bottom line. Dual Linux-based machinery linked by edge chains can proceeds improvement of link-aggregation.

A lone device linked by dual physical chains near a switch whichever provisions port ***'trunking'*** know how to usage link-aggregation to the switch. First straight switch determination develops deeply disordered via a hardware-address looking scheduled numerous ports concurrently [39].

The bonding provision in Linux remains portion like a great accessibility resolution. Designed aimed at an entrance idea keen on the complexity of great convenience cutting-edge combination by Linux [40].

Affecting term concerning the edge is able to state in the user.

The situation remains usually bond **0** or else somewhat parallel. Equally a common-sense edge, that one is able to use in routing-tables together through ***'tcpdump'***. Link aggregation and bonding-interface setup, is displayed below in **Fig. 7.**

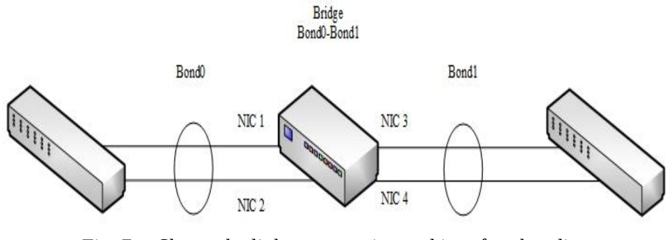


Fig. 7. Shows the links aggregation and interface bonding

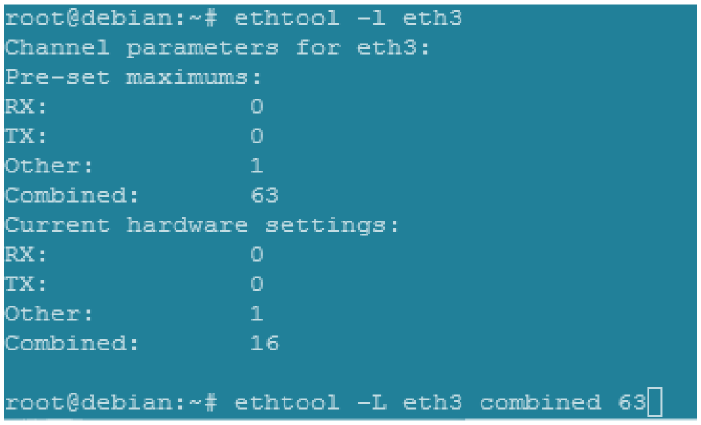
To be able to affect bridged frames, we needed to install ebtables tool and bridge-netfilter infrastructure. Also for tracking bandwidth and CPUs utilization, tools such as **'nload'** and **'htop'** will be installed on all servers [41].

That is it exist to console displays web network traffic flow and bandwidth procedure in physical time and **'htop'** powerful main excessive object. Nearby *htop* remains so to resolve display you practice apiece CPU, to boot expressive manuscript diagram of your memory and switch norm accurate on the maximum energy remain installed on all servers. Below code show the creation of headers for installation bridge-utilization, tools and run all servers.

- #apt-get install bridge-utilization
- #apt-get install ebtables
- #apt-get install htop
- #apt-get install nload

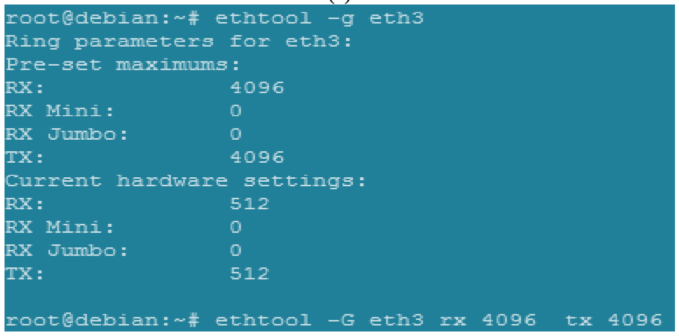
C. Data Packets Response Time Testing

Data packets response time testing, satisfaction increased traffic data of network interface controller (NIC) and Linux kernel. We need to improve packet reception process and all filters. **Fig. 8(a), (b)** shows the main check multi queue mode and ***rx_queues*** settings and buffers size enabling, if it's possible on data packets receiving network devices. **Fig. 8(b)** received frames on ***eth3*** will be processed in **63** queues, which increase possible amount of filtering the traffic of Cumulative NIC ***"ring buffers size"*** mentioned in above **Table I.**



```
root@debian:~# ethtool -l eth3
Channel parameters for eth3:
Pre-set maximums:
RX:             0
TX:             0
Other:          1
Combined:       63
Current hardware settings:
RX:             0
TX:             0
Other:          1
Combined:       16

root@debian:~# ethtool -L eth3 combined 63
```

(a)



```
root@debian:~# ethtool -g eth3
Ring parameters for eth3:
Pre-set maximums:
RX:             4096
RX Mini:        0
RX Jumbo:       0
TX:             4096
Current hardware settings:
RX:             512
RX Mini:        0
RX Jumbo:       0
TX:             512

root@debian:~# ethtool -G eth3 rx 4096  tx 4096
```

(b)

Fig. 8. (a) Shows the increasing number of *rx_queues* example. (b) Shows the ring buffers size.

- ***Supporting Receive Packet Routing:*** Therefore it will help prevent network data drops at the NIC during - periods, when large numbers of data frames are received we use this query such as: *(#echo 1 > /sys/class/net/eth3/queues/rx-0/rps_cpus)*.Checks the hardware queue of a single NIC, from becoming a bottleneck by creating the hash from the IP addresses and Port-numbers, which uses to determine the CPU near send the data packets.
- ***Hash Function Usage:*** The use of the hash function ensures that packets for the same stream of data are sent to the same CPU, which helps to increase performance.

V. Implementation and Experimental Testing Results

To make testing consistent, we will start tests from layer Two open systems interconnections (OSI) model. This means frames filtering with using only bridge code in Linux and then go up to network layer with filtering packets. For all DDoS types we will use: ***'ebtables rules approach'*** to filter Ethernet frames and ***'iptables rules approach'*** to filter IP packets as they are well-known and reliable.

Authentic traffic will be represented as **500** simulated users are sending ***'HTTP get'*** request to web server port **80** each second for **3** minutes. The output of such request and working preconfigured bridge is shown in **Table II and Fig. 9(a).**

First big spike corresponding is to unsuccessful transactions tab which is related to ***'Avalanche-Commander'*** specific functioning. Amount of traffic generated by authentic users on the '***Web server'*** is shown in **Fig. 9(b).**

A. Testing Result's of SYN (Synchronize) Flood Detection

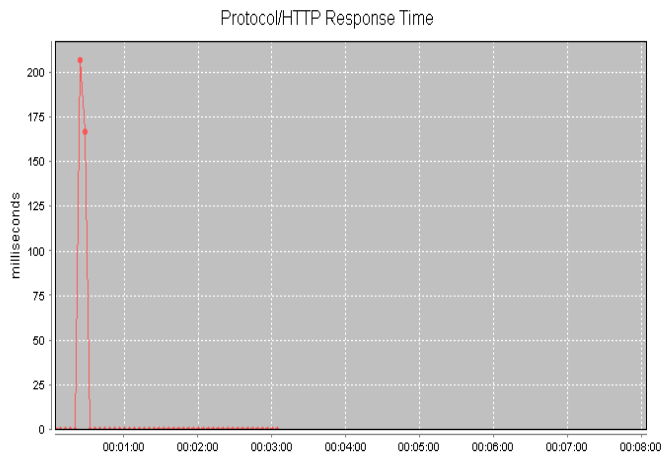
In this test ***'Avalanche Commander'*** sending packets with SYN flags set from IP addresses range **192.168.2.163** to **192.168.2.167**. By generating a big amount of SYN floods packets along with authentic traffic flow it is able to increased server response time besides execution script packet suite, which is shown in **Fig. 10(a).**

Mostly, there is no way to distinguish malicious packets traffic flow with SYN flag set on the Ethernet layer, so for filtering needed detections system which will provide us with malicious traffic flow from DDoS attacks using IP addresses:

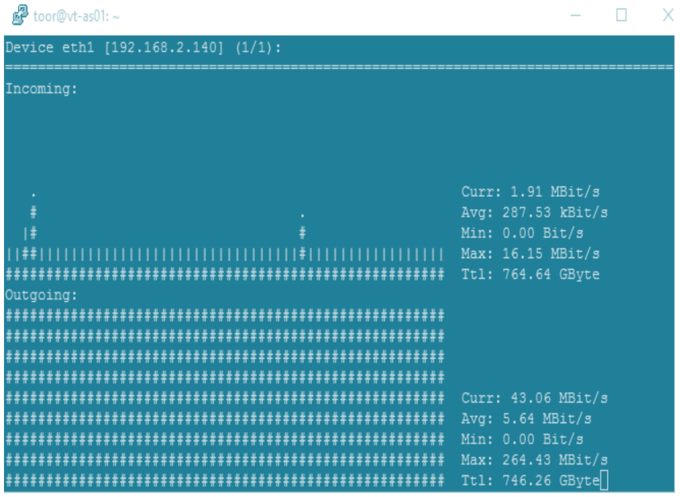
- #ebtables -f
- #ebtables -n syn_flooding
- #ebtables -a forward -p ipv6 --ip-proto tcp --ip-dport 80 -j syn_flooding
- #ebtables -a forward -p ipv6 --ip-proto tcp --ip-dport 443 -j syn_flooding
- #ebtables -a forward -p ipv6 --ip-proto tcp -j drop
- #ebtables -a syn_flooding -p ipv6 --among-src-file data -j drop
- #ebtables --atomic-file syn_flooding -t filter --atomic-save

TABLE II. USERS TRAFFIC FLOW ON WEB SERVERS (WSVRS)

Relations			Time (m s)						TCP Connections	
	Total	Rate Per Second		Page Response	URL Response	To TCP SYN/ACK	To First Data Byte	Est. Server Response		Total
Attempted	91879	491	**Minimum**	0	0	0.094	0.225	0	**Attempted**	91879
Successful	86031	460	**Maximum**	6017	6017	13999.418	6017.389	1996.095	**Established**	87494
Unsuccessful	5848	31	**Average**	14.329	14.329	210.003	14.594	3.928		
Aborted	0	0								



(a)

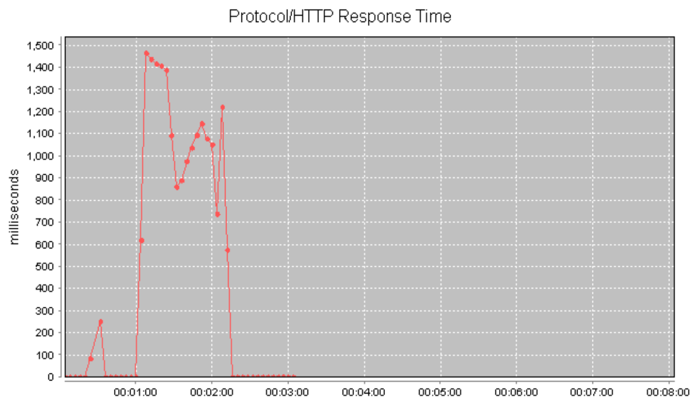


```
Device eth1 [192.168.2.140] (1/1):
==============================================================================
Incoming:





   .
   #                                  .
  |#                                  #
||##||||||||||||||||||||||||||||||||||#||||||||||||||||  Curr: 1.91 MBit/s
#######################################################  Avg: 287.53 kBit/s
                                                         Min: 0.00 Bit/s
                                                         Max: 16.15 MBit/s
                                                         Ttl: 764.64 GByte
Outgoing:
#######################################################
#######################################################
#######################################################
#######################################################
#######################################################  Curr: 43.06 MBit/s
#######################################################  Avg: 5.64 MBit/s
#######################################################  Min: 0.00 Bit/s
#######################################################  Max: 264.43 MBit/s
#######################################################  Ttl: 746.26 GByte
```

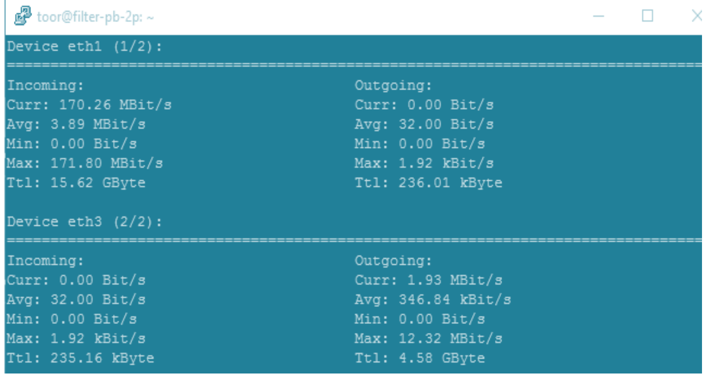
(b)

Fig. 9. (a) Shows specified milliseconds response time for HTTP/Protocol request. (b) Shows authentic user's incoming and outgoing network traffic flow generation on the web server.

In the file ***'data'*** we specified the range of MAC/IP-addresses. To test applied rule we will generate maximum available SYN flood speed and see how much of it is coming from filter server and how much is dropped. Using above IPv6 testing headers, mitigation techniques and ***'ebtables-tool'*** we get the incoming and outgoing SYN flood filter speed which is display in **Fig. 10(b).**



(a)



```
Device eth1 (1/2):
================================================================================
Incoming:                               Outgoing:
Curr: 170.26 MBit/s                     Curr: 0.00 Bit/s
Avg: 3.89 MBit/s                        Avg: 32.00 Bit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 171.80 MBit/s                      Max: 1.92 kBit/s
Ttl: 15.62 GByte                        Ttl: 236.01 kByte

Device eth3 (2/2):
================================================================================
Incoming:                               Outgoing:
Curr: 0.00 Bit/s                        Curr: 1.93 MBit/s
Avg: 32.00 Bit/s                        Avg: 346.84 kBit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 1.92 kBit/s                        Max: 12.32 MBit/s
Ttl: 235.16 kByte                       Ttl: 4.58 GByte
```

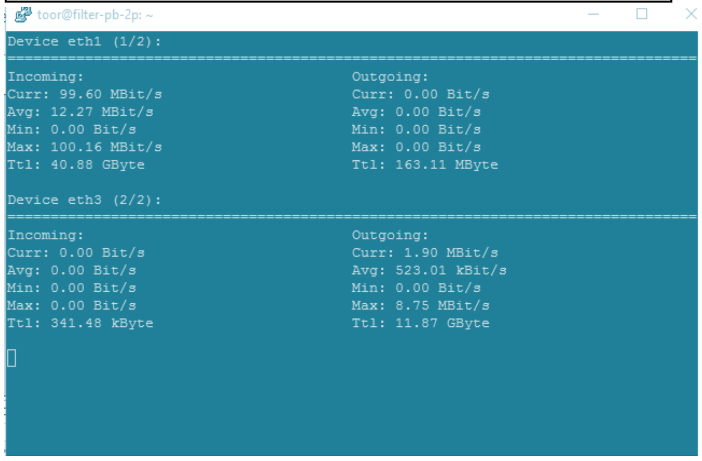
(b)

Fig. 10. (a) Shows authentic network traffic flow in SYN flood attacks. (b) Shows SYN flood incoming and outgoing filtering speed via ebtables.

As we see only user's traffic is passed through the server. There was no big additional CPU usage, corresponding to frame blocking. The maximum speed of incoming frames which kernel was able to filter is **170-180Mbit/s (~400000pps)**, including users and malicious traffic flow from DDoS - attacks, which corresponds to maximum throughput of NIC. Another way to filter SYN flood is limiting passing traffic flow which is based on ***'packets/s'***, to decrease some load from target servers. This resolution is affecting user traffic flow and also it is not preferable.

By default, only ebtables code is able to process bridged frames, so to let ***iptables rules*** receive traffic flow from bridged ports. So we need to enable ***bridgen-of-call-iptables*** feature. With the help of below IPv6 addresses, testing headers, mitigation techniques and ***'iptables-tool'***, we compare filtering flow with iptables rules utilizing following headers and result is display in **Fig. 10(c).**

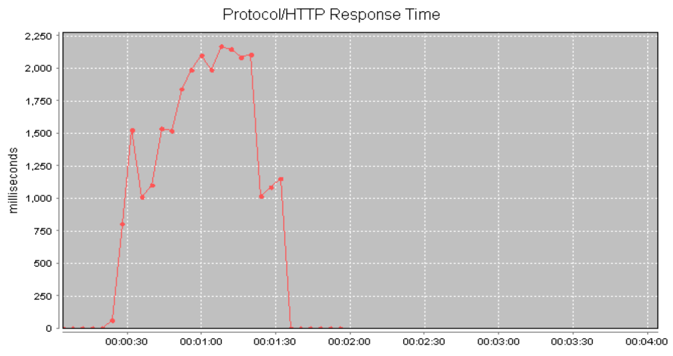
- #iptables –f
- #iptables –p forward accept
- #iptables -n syn_flooding
- #iptables -a forward -p tcp –m tcp --syn -j syn_flooding
- #iptables -a syn_flooding -m iprange --src-range
- 192.168.2.163- 162.168.2.167 -j drop



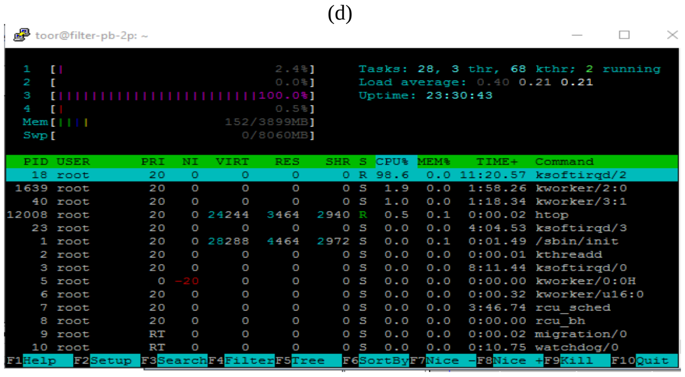
```
Device eth1 (1/2):
================================================================================
Incoming:                               Outgoing:
Curr: 99.60 MBit/s                      Curr: 0.00 Bit/s
Avg: 12.27 MBit/s                       Avg: 0.00 Bit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 100.16 MBit/s                      Max: 0.00 Bit/s
Ttl: 40.88 GByte                        Ttl: 163.11 MByte

Device eth3 (2/2):
================================================================================
Incoming:                               Outgoing:
Curr: 0.00 Bit/s                        Curr: 1.90 MBit/s
Avg: 0.00 Bit/s                         Avg: 523.01 kBit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 0.00 Bit/s                         Max: 8.75 MBit/s
Ttl: 341.48 kByte                       Ttl: 11.87 GByte
```

Fig. 10. (c) Shows SYN flood incoming and outgoing attacks via iptables filtering.

Although we are able to filter traffic on speed of **90-100Mbit/s**, the web server response time is still greatly increased due to CPU overloading, which is made by *Ksoftirqd-Per-CPUs* built kernel cord in order that turns while the device is below full Soft-Interrupt load [42]. Increased response time and CPUs utilization are shown in **Fig. 10(d) and (e).**



(d)



```
  1  [|                          2.4%]    Tasks: 28, 3 thr, 68 kthr; 2 running
  2  [                           0.0%]    Load average: 0.40 0.21 0.21
  3  [||||||||||||||||||||||||100.0%]    Uptime: 23:30:43
  4  [|                          0.5%]
  Mem[||||                 152/3899MB]
  Swp[                      0/8060MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   18 root       20   0     0     0     0 R 98.6  0.0 11:20.57 ksoftirqd/2
 1639 root       20   0     0     0     0 S  1.9  0.0  1:58.26 kworker/2:0
   40 root       20   0     0     0     0 S  1.0  0.0  1:18.34 kworker/3:1
12008 root       20   0 24244  3464  2940 R  0.5  0.1  0:00.02 htop
   23 root       20   0     0     0     0 S  0.0  0.0  4:04.53 ksoftirqd/3
    1 root       20   0 28288  4464  2972 S  0.0  0.1  0:01.49 /sbin/init
    2 root       20   0     0     0     0 S  0.0  0.0  0:00.01 kthreadd
    3 root       20   0     0     0     0 S  0.0  0.0  8:11.44 ksoftirqd/0
    5 root        0 -20     0     0     0 S  0.0  0.0  0:00.00 kworker/0:0H
    6 root       20   0     0     0     0 S  0.0  0.0  0:00.32 kworker/u16:0
    7 root       20   0     0     0     0 S  0.0  0.0  3:46.74 rcu_sched
    8 root       20   0     0     0     0 S  0.0  0.0  0:00.00 rcu_bh
    9 root       RT   0     0     0     0 S  0.0  0.0  0:00.02 migration/0
   10 root       RT   0     0     0     0 S  0.0  0.0  0:10.75 watchdog/0
F1Help  F2Setup F3SearchF4FilterF5Tree  F6SortByF7Nice -F8Nice +F9Kill  F10Quit
```

(e)

Fig. 10. (d) Shows milliseconds response time in SYN flood attacks. (e) Shows results of CPUs utilization.

Previously we disabled ***'rpc_cpus'*** feature to test CPUs utilization. Now we enable it back a run same test again. The result is shown in **Fig. 10(f).**

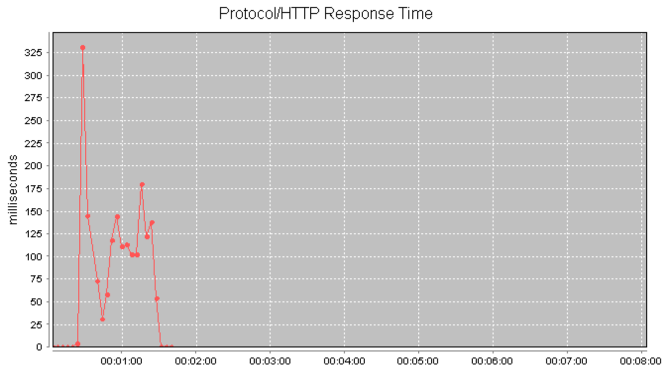


Fig. 10. (f) Shows milliseconds SYN flood attacks response time in *'smp_affinnity'*.

Only **3** out of **4** cores are loaded with processing packets, which decreasing response time to the acceptable range. However maximum amount of possible filtered traffic is increased from the **90Mbit/s** to **130Mbit/s**. Assuming using more complex iptables rules with same user's data traffic will lead to DDoS attacks. So we utilize more complex rules using headers and ***'Conntrack-tool'*** Linux kernel as follows:

These rules set the limit on amount connections per second coming from one IP address, assuming all **500** users will start simultaneously sending connection requests. The data traffic filtering speed, in this case, is around: **100Mbit/s** with enabled load spread. Now we compare same new tests to those which were conducted on all filter servers and compared results are shown in **Table III**.

- #iptables -f
- #iptables -n syn_flooding
- #iptables -a forward -p tcp -m state –state-new -j syn_flooding
- #iptables -a syn_flooding -m connlimit --connlimit-above 500 -j drop

TABLE III. SYN (SYNCHRONIZE) FLOOD ATTACKS FILTERING

SYN Flood Mbit/s	The New Testing Filter V1	The New Testing Filter V2	The New Testing Filter V3
Ebtables	182	178	174
Iptables V1	130	177	102
Iptables V2	100	184	83

As we can see, ebtables successfully allows filtering traffic on desired load. However, using ebtables can only be related with existence of any kind of detection system, which supplies filter with needed data. Using iptables rules we are able to filter in more stand-alone way but it requires more hardware resources to use in order to filter the same amount of data – traffic per second. Filter with aggregated links can benefit from having 1Gbps links instead of one, since the *'bottleneck'* is not in filtering. More of that due to less computing resources it show lower filtering throughput.

B. *Testing Result's of User Datagram Protocol (UDP) Flood Detection*

The User datagram protocol (UDP) flood attacks testing in our network based on sending as many packets as possible on web server port No.**80** with spoofed source IP addresses.

The main goal is to utilize all data filter servers for CPUs usage. Effect from generating UDP flood attacks can be seen in **Fig. 11(a) and (b).**

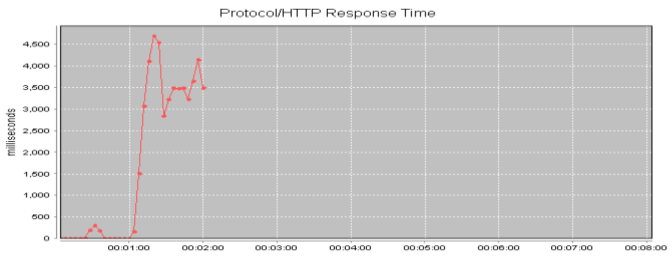
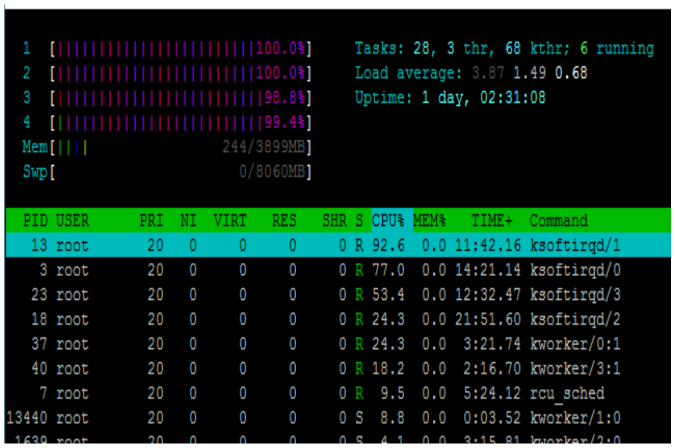


Fig. 11. (a) Shows Web Servers milliseconds response time in UDP flood attacks.



```
  1  [||||||||||||||||||||||||100.0%]     Tasks: 28, 3 thr, 68 kthr; 6 running
  2  [||||||||||||||||||||||||100.0%]     Load average: 3.87 1.49 0.68
  3  [||||||||||||||||||||||||98.8%]      Uptime: 1 day, 02:31:08
  4  [||||||||||||||||||||||||99.4%]
  Mem[||||               244/3899MB]
  Swp[                     0/8060MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   13 root       20   0     0     0     0 R 92.6  0.0 11:42.16 ksoftirqd/1
    3 root       20   0     0     0     0 R 77.0  0.0 14:21.14 ksoftirqd/0
   23 root       20   0     0     0     0 R 53.4  0.0 12:32.47 ksoftirqd/3
   18 root       20   0     0     0     0 R 24.3  0.0 21:51.60 ksoftirqd/2
   37 root       20   0     0     0     0 R 24.3  0.0  3:21.74 kworker/0:1
   40 root       20   0     0     0     0 R 18.2  0.0  2:16.70 kworker/3:1
    7 root       20   0     0     0     0 R  9.5  0.0  5:24.12 rcu_sched
13440 root       20   0     0     0     0 S  8.8  0.0  0:03.52 kworker/1:0
```

Fig. 11. (b) Shows results of CPUs utilization by UDP flood attacks.

On link layer to filter UDP flood attacks we able to set following headers and mitigation technique rules: UDP data packets which are destined to domain name servers (DNS) server on port 53 will pass, all others UDP data packets should be dropped. Results are shown in **Fig. 11(c) and (d).**

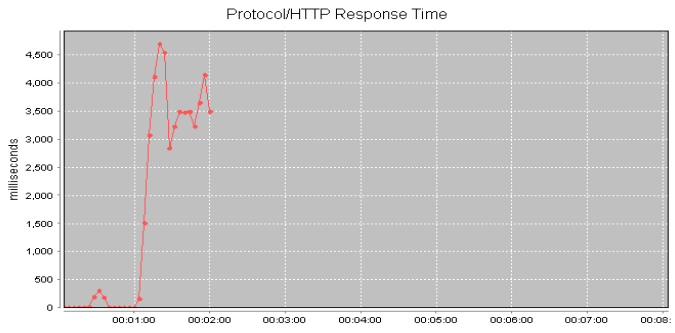
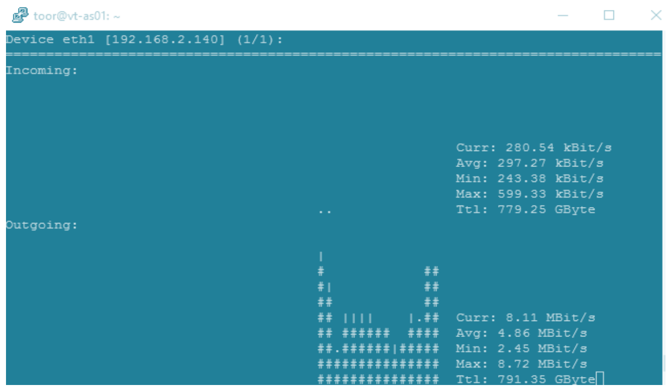


Fig. 11. (c) Shows Web Server milliseconds response time by data filtering.



```
toor@vt-as01: ~
Device eth1 [192.168.2.140] (1/1):
==========================================================================
Incoming:




                                                     Curr: 280.54 kBit/s
                                                     Avg: 297.27 kBit/s
                                                     Min: 243.38 kBit/s
                                                     Max: 599.33 kBit/s
                                       ..            Ttl: 779.25 GByte
Outgoing:

                                       |
                                       #            ##
                                       #|           ##
                                       ##           ##
                                       ## ||||    |.##  Curr: 8.11 MBit/s
                                       ## ######  ####  Avg: 4.86 MBit/s
                                       ##.######|#####  Min: 2.45 MBit/s
                                       ###############  Max: 8.72 MBit/s
                                       ###############  Ttl: 791.35 GByte
```

Fig. 11. (d) Shows results of Web Server incoming and outgoing users load.

- #ebtbales - f
- #ebtables - n udp_flooding
- #ebtables - p udp_flooding drop
- #ebtables - a forward -p ipv6 --ip-proto udp! --ip-dport 53 -j udp_flooding

As it can be seen on Web server data traffic that is still struggle to pass through filtering. All **4** filter cores server are loaded with 'Ksoftirqd-Program' and filtering is possible but web server has still big response delay.

To reduce CPUs utilization on filter, it's possible to apply dropping packets even before them being processed by kernel. Among ebtables hooks there is nat table with: the PREROUTING chain which is logically located between kernel network stack and NIC.

Applying same rules in table nat PREROUTING chain results for better performance, which can be shown in **Fig. 11(e) and (f).**

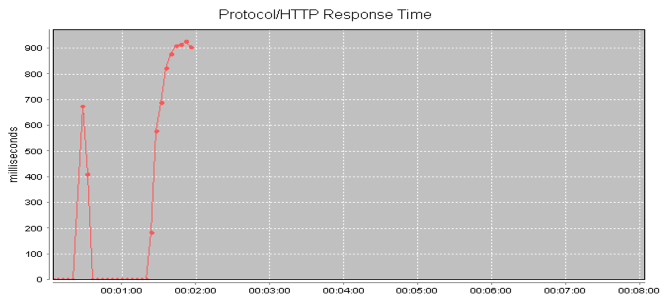
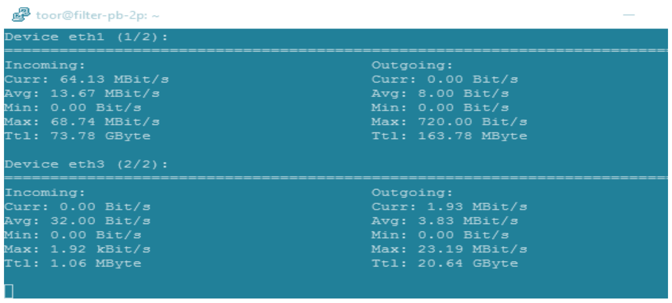


Fig. 11. (e) Shows users traffic milliseconds response time.



```
toor@filter-pb-2p: ~
Device eth1 (1/2):
==========================================================================
Incoming:                              Outgoing:
Curr: 64.13 MBit/s                     Curr: 0.00 Bit/s
Avg: 13.67 MBit/s                      Avg: 8.00 Bit/s
Min: 0.00 Bit/s                        Min: 0.00 Bit/s
Max: 68.74 MBit/s                      Max: 720.00 Bit/s
Ttl: 73.78 GByte                       Ttl: 163.78 MByte

Device eth3 (2/2):
==========================================================================
Incoming:                              Outgoing:
Curr: 0.00 Bit/s                       Curr: 1.93 MBit/s
Avg: 32.00 Bit/s                       Avg: 3.83 MBit/s
Min: 0.00 Bit/s                        Min: 0.00 Bit/s
Max: 1.92 kBit/s                       Max: 23.19 MBit/s
Ttl: 1.06 MByte                        Ttl: 20.64 GByte
```

Fig. 11. (f) Shows results of filtering incoming and outgoing UDP flood attacks with ebtables.

In the part of user datagram protocol (UDP) Flood was decreased along with 'http-response time'. Although it stays in acceptable range under the **1** second further countermeasures in network are required.

Considering ebtables experience, iptables rules have to be applied on corresponding netfilter hook to have better result. So iptables table raw with: the PREROUTING chain should be configured as follows:

- #iptables - f
- #iptables – t raw –a pre – routing –p udp –dport 53 –d 192.168.2.145 -j
- Accept
- #iptables - t raw -a pre – routing –p udp –d 192.168.2.145 –j drop

For all passing user datagram protocol (UDP) packets going to web server should be dropped, except destined to domain name servers (DNS), we set the above headers. The Results output is shown in **Fig. 11(g) and (h).**

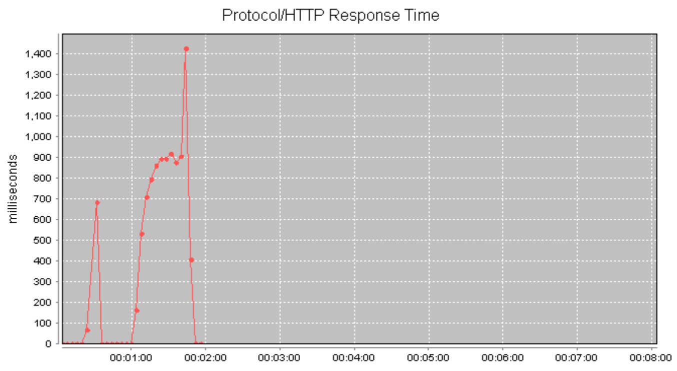
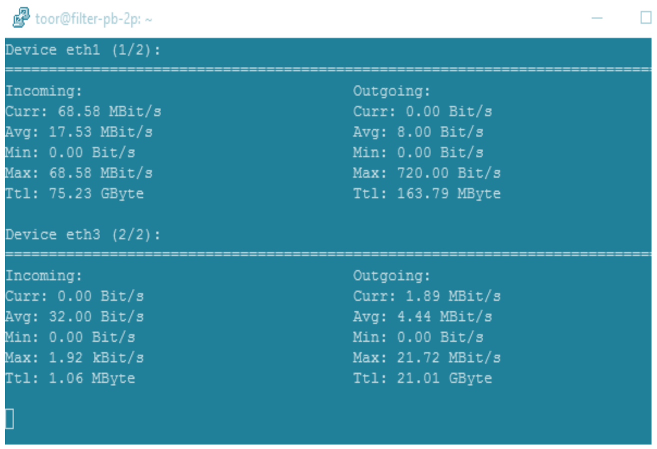


Fig. 11. (g) Shows milliseconds response time of the UDP flood with iptables.



```
Device eth1 (1/2):
================================================================================
Incoming:                               Outgoing:
Curr: 68.58 MBit/s                      Curr: 0.00 Bit/s
Avg: 17.53 MBit/s                       Avg: 8.00 Bit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 68.58 MBit/s                       Max: 720.00 Bit/s
Ttl: 75.23 GByte                        Ttl: 163.79 MByte

Device eth3 (2/2):
================================================================================
Incoming:                               Outgoing:
Curr: 0.00 Bit/s                        Curr: 1.89 MBit/s
Avg: 32.00 Bit/s                        Avg: 4.44 MBit/s
Min: 0.00 Bit/s                         Min: 0.00 Bit/s
Max: 1.92 kBit/s                        Max: 21.72 MBit/s
Ttl: 1.06 MByte                         Ttl: 21.01 GByte
```

Fig. 11. (h) Shows results of maximum incoming and outgoing UDP flood traffic through output.

How can be seen both ebtables and iptables are able to filter UDP, flood attacks. However, using iptables gives us litter bigger delay while communicating with web server. That is consequences of that iptables uses more code to process each packet, so it need more calculating time for the rest of server. Same tests were conducted and results are combined in **Table IV.**

TABLE IV. CONDUCTED REST OF SERVERS UDP FLOOD FILTERING TESTS RESULTS

UDP FLOOD (Mbit/s)	The New Testing Filter V1	The New Testing Filter V2	The New Testing Filter V3
Ebtables	67	67	67
Iptables	68	64	65

As this attack doesn't consume many resources, each server was able to completely filter UDP flood on maximum available any through output.

C. *Testing Result's of Internet Control Message Protocol (ICMP) Flood Detection*

Internet control message protocol (ICMP) flood attacks rely on constantly sending 'echo_request' to force our web server to respond and consume additional resources. Generating attacks along with sending users data traffic flow also consumes, as shown in **Fig. 12 (a) and (b).**

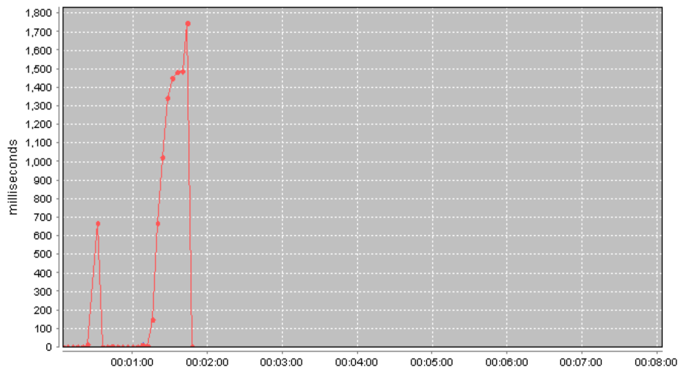
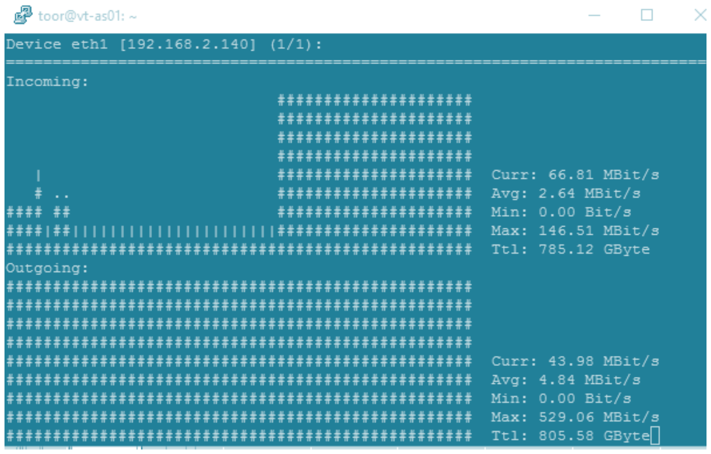


Fig. 12. (a) Shows HTTP network traffic milliseconds response time.



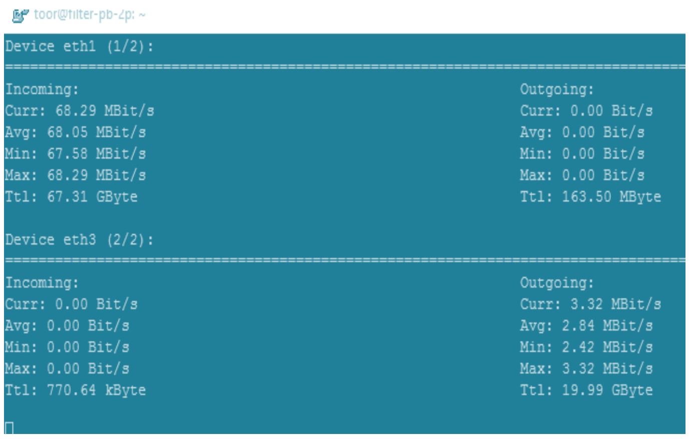
```
Device eth1 [192.168.2.140] (1/1):
================================================================================
Incoming:
                             #####################
                             #####################
                             #####################
                             #####################
   |                         #####################  Curr: 66.81 MBit/s
   # ..                      #####################  Avg: 2.64 MBit/s
#### ##                      #####################  Min: 0.00 Bit/s
####|##||||||||||||||||||||||#####################  Max: 146.51 MBit/s
##################################################  Ttl: 785.12 GByte
Outgoing:
##################################################
##################################################
##################################################
##################################################
##################################################  Curr: 43.98 MBit/s
##################################################  Avg: 4.84 MBit/s
##################################################  Min: 0.00 Bit/s
##################################################  Max: 529.06 MBit/s
##################################################  Ttl: 805.58 GByte
```

Fig. 12. (b) Shows results of Web Server incoming and outgoing load in ICMP flood attacks.

ICMP, flood data traffic flow is crucial for network control and it cannot be dropped at all. The best way is to filter data flow using following headers and ***'link-layer'*** mitigating technique for set limits:

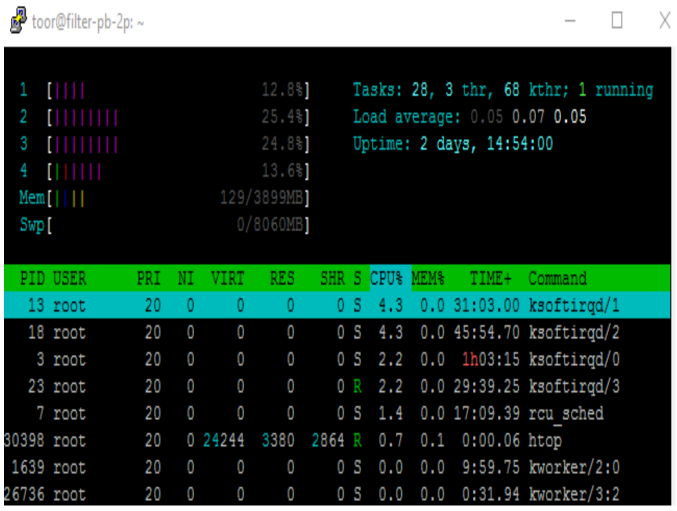
As a standard icmp_echo request rate is 12 packet per 10 second, our limit is letting users to ping servers and describing filtering process which is shown in **Fig. 12 (c) and (d)**.

- #iptables -n icmp_flooding
- #iptables -a forward -p icmp -j icmp_flooding
- #iptables -a icmp_flooding -p icmp --icmp-type echo-request -s 192.168.2.0/24 -j Accept
- # Iptables -a icmp_flooding -j drop



```
toor@filter-pb-2p: ~
Device eth1 (1/2):
================================================================================
Incoming:                                             Outgoing:
Curr: 68.29 MBit/s                                    Curr: 0.00 Bit/s
Avg: 68.05 MBit/s                                     Avg: 0.00 Bit/s
Min: 67.58 MBit/s                                     Min: 0.00 Bit/s
Max: 68.29 MBit/s                                     Max: 0.00 Bit/s
Ttl: 67.31 GByte                                      Ttl: 163.50 MByte

Device eth3 (2/2):
================================================================================
Incoming:                                             Outgoing:
Curr: 0.00 Bit/s                                      Curr: 3.32 MBit/s
Avg: 0.00 Bit/s                                       Avg: 2.84 MBit/s
Min: 0.00 Bit/s                                       Min: 2.42 MBit/s
Max: 0.00 Bit/s                                       Max: 3.32 MBit/s
Ttl: 770.64 kByte                                     Ttl: 19.99 GByte
```

Fig. 12. (c) Shows ICMP flood incoming and outgoing filtering using ebtables.



```
toor@filter-pb-2p: ~
  1  [||||                    12.8%]    Tasks: 28, 3 thr, 68 kthr; 1 running
  2  [||||||||                25.4%]    Load average: 0.05 0.07 0.05
  3  [||||||||                24.8%]    Uptime: 2 days, 14:54:00
  4  [||||||                  13.6%]
  Mem[||||               129/3899MB]
  Swp[                     0/8060MB]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
   13 root       20   0     0     0     0 S  4.3  0.0 31:03.00 ksoftirqd/1
   18 root       20   0     0     0     0 S  4.3  0.0 45:54.70 ksoftirqd/2
    3 root       20   0     0     0     0 S  2.2  0.0  1h03:15 ksoftirqd/0
   23 root       20   0     0     0     0 R  2.2  0.0 29:39.25 ksoftirqd/3
    7 root       20   0     0     0     0 S  1.4  0.0 17:09.39 rcu_sched
30398 root       20   0 24244  3380  2864 R  0.7  0.1  0:00.06 htop
 1639 root       20   0     0     0     0 S  0.0  0.0  9:59.75 kworker/2:0
26736 root       20   0     0     0     0 S  0.0  0.0  0:31.94 kworker/3:2
```

Fig. 12. (d) Shows results of CPUs load in ICMP filtering.

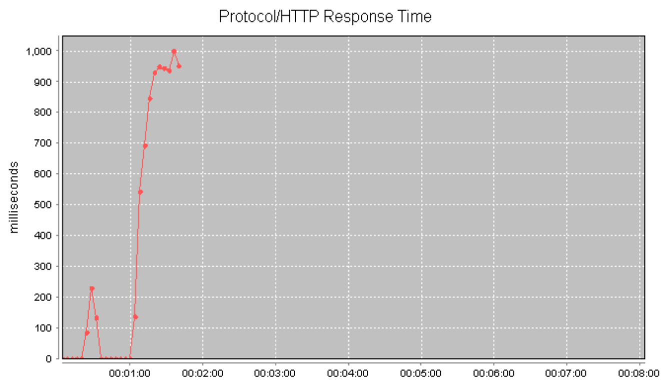
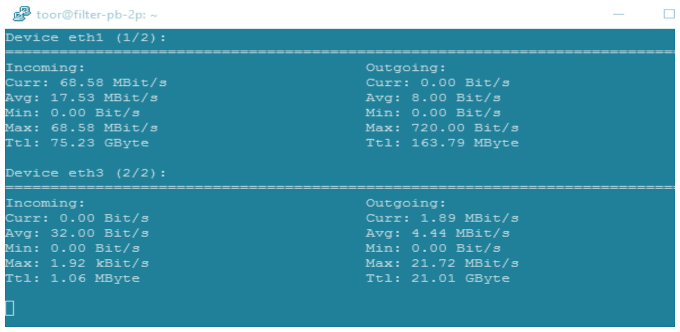


Fig. 12. (e) Shows milliseconds response time of ICMP flood filtering with iptables.



```
toor@filter-pb-2p: ~
Device eth1 (1/2):
================================================================================
Incoming:                                  Outgoing:
Curr: 68.58 MBit/s                         Curr: 0.00 Bit/s
Avg: 17.53 MBit/s                          Avg: 8.00 Bit/s
Min: 0.00 Bit/s                            Min: 0.00 Bit/s
Max: 68.58 MBit/s                          Max: 720.00 Bit/s
Ttl: 75.23 GByte                           Ttl: 163.79 MByte

Device eth3 (2/2):
================================================================================
Incoming:                                  Outgoing:
Curr: 0.00 Bit/s                           Curr: 1.89 MBit/s
Avg: 32.00 Bit/s                           Avg: 4.44 MBit/s
Min: 0.00 Bit/s                            Min: 0.00 Bit/s
Max: 1.92 kBit/s                           Max: 21.72 MBit/s
Ttl: 1.06 MByte                            Ttl: 21.01 GByte
```

Fig. 12. (f) Shows results of the incoming and outgoing response time in iptables filtering.

As can be seen there is no restriction from hardware on filtering side, but still there is a small delay related with packets processing. On network layer it's possible to deny any: echo_request packets from outside of our network, since we want to leave troubleshooting options for network administrator. For that purpose we set following headers and need to follow below rules and regulations. Desired results are described in **Fig. 12 (e) and (f)**.

As expected all malicious internet control message protocol (ICMP) traffic was filtered by adding a small delay in response time. The results for rest of the servers are described in **Table V.**

TABLE V. ICMP FLOOD NEW TESTING RESULTS

ICMP FLOOD (Mbit/s)	**The New Testing Filter V1**	**The New Testing Filter V2**	**The New Testing Filter V3**
Ebtables	68	70	69
Iptables	68	68	68

A tiny part of computing resources is required for processing internet control message protocol (ICMP) flood, which makes this type of attacks filtering less complex and available across networks worldwide.

VI. CONCLUSION AND OUTLOOKS

In this paper, we discussed how Linux-based resolution approaches for Web Servers WSYRs, tools and mitigation technique principles can be applied to effectively control DoS and DDoS attacks.

These approaches can give several benefits to WWW, users. The proposed approaches pose a number of challenges for DDN, management and cooperative communities on the internet technology losses.

Although DDoS attacks are effectively achieving such goals, you can find reasonable protection against certain types of DoS and DDoS attacks. In other words, the frequency, power, harshness of bandwidth, processor speed, and the number of available systems that can be attacked and compromised will continue to grow, as well as attack tools complexity to compromise with computers and their usage for attacks.

To mitigate DoS and DDoS attacks usually requires many different security mechanisms. However, the situation remains no more advisable toward softly picking a huge usually like protection devices in contradiction of DoS and DDoS attacks.

This smart-stated research work focus was to test the correct tools to filter data traffic flow. Maximum throughput achieved with SYN (Synchronize) filtering flood attack is **187 Mbit/s,** user datagram protocol (UDP) flood max. Performance is **67 Mbit/s,** and internet control message protocol (ICMP) flood is filtered to the maximum **71 Mbit/s**.

In view of the major consideration, we can say in this study that the Linux-based resolution approaches provide the complete set of tools needed to filter traffic flow. For example, the SYN Flood DDoS attack can be reduced by packet filtering using iptables without additional hardware or software.

"But by installing additional detection systems on the network, the best filtering performance can be achieved with the ebtables tool."

The difference lies in the packet flow within the central network stack.

DDoS attacks on the UDP, thread can be alleviated with all available tools, but additional kernel configuration is required. The ability to add aggregates with Linux-based software can be used in cases where a high-voltage network is used.

An extra speed can be achieved for users who use so-called: **'bonding'**.

In real time, proposed methods to mitigate DDoS should be considered temporary measures due to limited resources in the provided systems.

The latest DDoS speeds in the world outperform the possibilities of any filtering equipment or software. Therefore, good cooperation between all parties involved in global WWW, work is required to successfully filter malicious traffic.

VII. Future Scope

Expressive netfilter or iptables coordination stays perfect model for Linux-based method proprietors, system proprietors. It may also be suitable for all users which need towards design firewalls, allowing near their explicit requirements. In the direction of excluding change arranged firewall resolutions, besides entire device packets IP filtering.

Although this research paper provides, adequate protection against the refusal of attacks, many areas remain under the protection mechanism used for further investigation that can be expanded by this work by using iptables advanced features such as network address translation (NAT), IP masking, redirect IP Tables packets.

These have not only the ability to redirect packets, such as IP chains, but also have a destination NAT (DNAT) allocation, which allows to freely change the distribution of the IP address and port-numbers.

So you can actually hide web address where the service packets go through DNAT. Additional dynamic exploration part could be the development of policy scripts in the ***'Bro-language'*** to detect DoS attacks using ***'Bro-IDS'*** and develop effective against Anti DDoS ***'Snort-IDS'*** rules.

Acknowledgment

The Publication of this research was support by Nature Science Foundation of China under Grant No. 61572095.

Abbreviations

The following abbreviations are used in this research manuscript:

ACK	Acknowledgement Data Networks
LAPS	Linux - Availability Protection System
BSD	Berkeley Software Distribution
CPUs	Central Processing Units
CUSUM	Cumulative Sum Control Chart
DoS	Denial-of-Service
DDoS	Distributed Denial-of-Service
DDN	Data-Driven Networks
DNS	Domain Name Servers
DIT	Digital Information Technology
DNAT	Destination Network Address Translation
FIFO	First In, First Out
ICMP	Internet Control Message Protocol
ISPs	Internet Service Providers
IRC	Internet Relay Chat
IP	Internet Protocol
IETF	Internet Engineering Task Force
LPS	Linux Protection System
LAG	Linux Aggregation Group
LCPUs	Logical Central Processing Units
MIB	Management Information Base
MSS	Ministry of State Security
MSS	Maximum Segment Size
MAC	Media Access Protocol
NIC	Network Interface Card
NAT	Network Address Translation
OS	Operating System
OSI	Open System Interconnections
PDOS	Pulsing Distributed Denial-of-Service
P2P	Peer-to-Peer
QDISCS	Queuing Disciplines
RSTP	Rapid Spanning Tree Protocol
RFC	Request for Comments
RC	Remote Control
SYN	Synchronize
SDBot	Spartan-Dominion Robot
SQLi	SQL Injection
TFN	Tribe Flood Network
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
VPN	Virtual Private Network
WSVRs	Web Servers
WWW	World Wide Web

References

[1] Cobb, S., things to know about the October 21 IoT DDos attacks,". WeLiveSecurity, In press release. 24.

[2] Wu, L., Designing Effective Security and Privacy Schemes for Wireless Mobile Devices. 2017, Temple University.

[3] Bonguet, A. and M. Bellaiche, A Survey of Denial-of-Service and Distributed Denial of Service Attacks and Defenses in Cloud Computing. Future Internet, 2017. 9(3): p. 43.

[4] Saleh, M.A. and A. Abdul Manaf, A novel protective framework for defeating http-based denial of service and distributed denial of service attacks. The Scientific World Journal, 2015. 2015.

[5] Tahir, M., et al., The Novelty of A-Web based Adaptive Data-Driven Networks (DDN) Management & Cooperative Communities on the Internet Technology. INTERNATIONAL JOURNAL OF ADVANCED COMPUTER SCIENCE AND APPLICATIONS, 2017. 8(5): p. 16-24.

[6] Alam, M.F., Application layer ddos a practical approach & mitigation techniques. bdHUB Limited, 2014.

[7] Hadi, A.D.A., F.H. Azmat, and F.H.M. Ali. IDS Using Mitigation Rules Approach to Mitigate ICMP Attacks. in Advanced Computer Science Applications and Technologies (ACSAT), 2013 International Conference on. 2013. IEEE.

[8] Zargar, S.T., J. Joshi, and D. Tipper, A survey of defense mechanisms against distributed denial of service (DDoS) flooding attacks. IEEE communications surveys & tutorials, 2013. 15(4): p. 2046-2069.

[9] Bhuyan, M.H., D.K. Bhattacharyya, and J.K. Kalita, Network anomaly detection: methods, systems and tools. Ieee communications surveys & tutorials, 2014. 16(1): p. 303-336.

[10] François, J., I. Aib, and R. Boutaba, FireCol: a collaborative protection network for the detection of flooding DDoS attacks. IEEE/ACM Transactions on Networking (TON), 2012. 20(6): p. 1828-1841.

[11] Senie, D. and P. Ferguson, Network ingress filtering: Defeating denial of service attacks which employ IP source address spoofing. Network, 1998.

[12] Park, K. and H. Lee. On the effectiveness of route-based packet filtering for distributed DoS attack prevention in power-law internets. in ACM SIGCOMM computer communication review. 2001. ACM.

[13] Savage, S., et al., Network support for IP traceback. IEEE/ACM transactions on networking, 2001. 9(3): p. 226-237.

[14] Song, D.X. and A. Perrig. Advanced and authenticated marking schemes for IP traceback. in INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. 2001. IEEE.

[15] Bellovin, S.M., M. Leech, and T. Taylor, ICMP traceback messages. 2003.

[16] Mahajan, R., et al., Controlling high bandwidth aggregates in the network. ACM SIGCOMM Computer Communication Review, 2002. 32(3): p. 62-73.

[17] Siris, V.A. and F. Papagalou. Application of anomaly detection algorithms for detecting SYN flooding attacks. in Global Telecommunications Conference, 2004. GLOBECOM'04. IEEE. 2004. IEEE.

[18] Wang, H., D. Zhang, and K.G. Shin. Detecting SYN flooding attacks. in INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. 2002. IEEE.

[19] Luo, X. and R.K. Chang. On a New Class of Pulsing Denial-of-Service Attacks and the Defense. in NDSS. 2005.

[20] Cabrera, J.B., et al. Proactive detection of distributed denial of service attacks using mib traffic variables-a feasibility study. in Integrated Network Management Proceedings, 2001 IEEE/IFIP International Symposium on. 2001. IEEE.

[21] Russell, R., Equity in Eden: can environmental protection and affordable housing comfortably cohabit in suburbia. BC Envtl. Aff. L. Rev., 2002. 30: p. 437.

[22] Russell, R. and H. Welte, Linux netfilter hacking howto. Disponıvel em http://www. netfilter. org/documentation/HOWTO//netfilter-hacking-HOWTO. letter. ps (Junho de 2005), 2002.

[23] Fouda, K.M., Payload based signature generation for DDoS attacks. 2017, University of Twente.

[24] Lau, F., et al. Distributed denial of service attacks. in Systems, Man, and Cybernetics, 2000 IEEE International Conference on. 2000. IEEE.

[25] Tunc, C. and S. Hariri, CLaaS: Cybersecurity Lab as a Service. J. Internet Serv. Inf. Secur., 2015. 5(4): p. 41-59.

[26] Xu, R., W.-l. Ma, and W.-l. Zheng. Defending against UDP flooding by negative selection algorithm based on eigenvalue sets. in Information Assurance and Security, 2009. IAS'09. Fifth International Conference on. 2009. IEEE.

[27] Douligeris, C. and A. Mitrokotsa, DDoS attacks and defense mechanisms: classification and state-of-the-art. Computer Networks, 2004. 44(5): p. 643-666.

[28] Bitzer, J. and P.J. Srhroder, Linux vs. Windows: A Comparison of Application and Platform Innovation Incentives for Open Source and Proprietary Software Platforms "published in Elsevier BV. 2006.

[29] Chakeres, I.D. and E.M. Belding-Royer. AODV routing protocol implementation design. in Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on. 2004. IEEE.

[30] Rivero de la Cruz, A., High available GNU/Linux systems. 2014, Universitat Oberta de Catalunya.

[31] Radhakrishnan, S., Linux–advanced networking overview version 1. 1999.

[32] de Schuymer, B. and N. Fedchik, Ebtables/Iptables interaction on a Linux-based Bridge. 2003.

[33] Hoffman, D., D. Prabhakar, and P. Strooper. Testing iptables. in Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research. 2003. IBM Press.

[34] Andreasson, O., Iptables Tutorial 1.1. 19. Internet article at: http://people. unix-fu. org/andreasson/iptables-tutorial/iptables-tutorial. html, 2001.

[35] Diekmann, C., et al. Verified iptables firewall analysis. in IFIP Networking Conference (IFIP Networking) and Workshops, 2016. 2016. IEEE.

[36] Shannon, F.M., et al., Link aggregation protection. 2012, Google Patents.

[37] Williams, M., Linux ethernet bonding driver HOWTO. 2006, April.

[38] Chen, S.-J., et al., A Behavior-Based Web Application Firewall Total Solution to Detect Various Web Application Service Attacks.

[39] Perloff, R.S., A.H. Frederik, and T.J. Christian, Multi-device link aggregation. 2005, Google Patents.

[40] Dow, E., S. Loveland, and G. Markos, A reference implementation architecture for deploying a highly-available networking infrastructure for cloud computing and virtual environments using OSPF. IBM Platform Test-z Systems library, 2009.

[41] Bowen, J.L., et al., NLOAD: AN INTERACTIVE, WEB- BASED MODELING TOOL FOR NITROGEN MANAGEMENT IN ESTUARIES. Ecological Applications, 2007. 17(sp5).

[42] Aust, S., et al. Evaluation of Linux bonding features. in Communication Technology, 2006. ICCT'06. International Conference on. 2006. IEEE.

AUTHORS' PROFILES

Muhammad Tahir: Was born in 1987. He received his Bachelor of Science BS Degree in Software Engineering from University of Sindh, Jamshoro, Pakistan in 2008. And Master of Engineering Degree in Software Engineering from CHONGQING University, Chongqing P.R. China in 2014. Currently he is a Ph.D. Research Scholar in School of Software Technology, Dalian University of Technology, P.R. China. His research interest includes: Cooperative Game Theory, Web based Software Defect Prediction Models, Web-based Security & its Testing, IOT, Cloud & Fog Computing, Software Defined Networks (SDN) & Networking.

Mingchu Li: Received the BS Degree in Mathematics, from Jiangxi Normal University, P.R. China and the MS Degree in Applied Science from University of Science and Technology, Beijing, P.R. China. He worked as an Associate Professor from 1989 to 1994. He worked for School of Software, Tianjin University, as a full Professor, (from 2002 to 2004) & from (2004 to now), for School of Software Technology, Dalian University of Technology as a full Professor, and Vice Dean. His main research interests include Theoretical Computer Science & Information Security, and Trust Models and Cooperative Game theory.

Naeem Ayoub: Became the student member of IEEE in 2017. He is pursuing his Ph.D. Degree at School of Computer Science and Application Technology, Dalian University of Technology, P.R. China. He received his Master's Degree in information technology from University of Education Lahore, Pakistan in 2013. His research interest includes: Computer

Vision, Image Processing, Saliency Detection, Iris Recognition, IOT and Wireless Sensor Networks (WSNs).

Usman Shehzaib: Is an Assistant Professor of Computer Science, in the Department of Computer Science, COMSATS Institute of Information Technology Lahore, Pakistan. His research interest includes: Databases, Data Mining and Big Data.

Atif Wagan: Is a Ph.D. Research Scholar in the School of Computer Science and Engineering, Nanjing University of Science & Technology, (NUST), Nanjing, P.R. China. His research interest includes: Software Testing, Big Data, Artificial Intelligence, Machine Learning and Deep Learning.