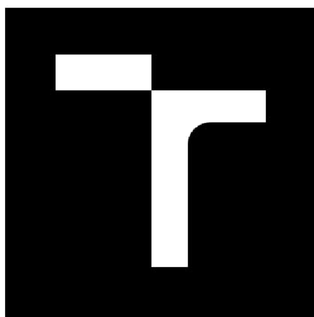


BRNO UNIVERSITY OF TECHNOLOGY

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

FAKULTA ELEKTROTECHNIKY
A KOMUNIKAČNÍCH TECHNOLOGIÍ

DEPARTMENT OF TELECOMMUNICATIONS

ÚSTAV TELEKOMUNIKACÍ

FILTERING AND AGGREGATION OF NETWORK TRAFFIC

FILTROVÁNÍ A AGREGACE SÍŤOVÉHO PROVOZU

MASTER'S THESIS

DIPLOMOVÁ PRÁCE

AUTHOR
AUTOR PRÁCE

Bc. Artem Zubov

SUPERVISOR
VEDOUCÍ PRÁCE

Ing. Zdeněk Martinásek, Ph.D.

BRNO 2017

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Telekomunikační a informační technika**
Ústav telekomunikací

Student: Bc. Artem Zubov ***ID:*** 171483
Ročník: 2 ***Akademický rok:*** 2016/17

NÁZEV TÉMATU:

Filtrování a agregace síťového provozu

POKYNY PRO VYPRACOVÁNÍ:

Cílem diplomové práce je nalézt a porovnat dostupné knihovny a softwarové nástroje pro filtraci síťového provozu. Síťový provoz bude cílen na protokoly HTTP a HTTPS. Dílčím cílem práce je také tento síťový provoz agregovat pomocí dostupných protokolů (např. LACP). Z analýzy současného stavu problematiky budou vybrány nejvhodnější nástroje, které budou implementovány a otestovány na experimentálním pracovišti. Experimenty budou předpokládat legitimní provoz HTTP(S) a nelegitimní vybrané útoky DDoS. Výstupem diplomové práce bude softwarová implementace schopná filtrovat a agregovat síťový provoz HTTP a HTTPS dle stanovených pravidel, tak aby byl DDoS útok potlačen. Dosažené výsledky přehledně zhodnoťte.

DOPORUČENÁ LITERATURA:

[1] PERLOFF, Ronald S.; FREDERIK, Anderson H.; CHRISTIAN, Thrysoee J. Multi-device link aggregation. U.S. Patent No 6,910,149, 2005.

[2] Debian - Documentation. Debian - Documentation [online]. 2016 [cit. 2016-09-12]. Dostupné z: https://www.debian.org/doc/index.en.html

Termín zadání: 1.2.2017 ***Termín odevzdání:*** 24.5.2017

Vedoucí práce: Ing. Zdeněk Martinásek, Ph.D.
Konzultant:

doc. Ing. Jiří Mišurec, CSc.
předseda oborové rady

UPOZORNĚNÍ:
Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRACT

Purpose of this work is to find, describe and test existing tools and features available in linux-based solution for filtering malicious traffic. As source of malicious traffic taken most widly-used Ddos attacks targeting Web servers. SYN, UDP and ICMP Flood attacks are described and different variants of they mitigation are explained. Available tools for manipulating with traffic, like *ebtables* and *iptables* tools are compared, based on each type of attack. 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. As a result, ebtables tool appears to be most productive, due to less resources it needed to process each packet (frame). Pointed out that separate detecting system is needed for this tool, in order to provide further filtering methods with data. As main conclusion, linux-based solutions provide full functionality for filtering traffic either in stand-alone state or combined with detecting systems.

KEYWORDS

Ddos, Netfilter, Ebtables, SYN Flood, UDP Flood, RPS_cpus, Irqbalance.

ABSTRAKT

V této práci jsou zkoumaní základní principy odporů servisních útoků, nejběžnějších typů a účelu použití. Popsané dostupné techniky zmírnění různých typu útoků, nástrojů a přístupů v operačních systémech postavených na Linuxu. Nakonfigurován filtrcni server a pro účely testování simulovan SYN Flood, UDP Flood a ICMP Flood útoky. Bylo zjištěno, vhodne techniky vyrovnání tehto druhu útoku a realizováné příslušna konfigurace filtrování.

KLÍČOVÁ SLOVA

Ddos, Netfilter, Ebtables, SYN Flood, UDP Flood, RPS_cpus, Irqbalance.

ZUBOV, A. *Filtrování a agregace síťového provozu.* Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. XY s. Vedoucí diplomové práce Ing. Zdeněk Martinásek, Ph.D.

DECLARATION

I declare that my master thesis on the topic of filtering and aggregation of network traffic, I developed independently under the leadership of my supervisor, using literature and other information sources, all of which are cited in the work and listed in the bibliography at the end of work.

As the author mentioned master thesis further declare that, in relation with the creation of this master thesis, I did not violate the copyrights of third parties, in particular, I did not intervened illegally in foreign copyrights of personal data and / or property, and I am fully aware of the consequences of violation of § 11 et seq Act no. 121/2000 Coll., on copyright, rights related to copyright and amending some laws (copyright Act), as amended, including possible criminal consequences arising from the provisions of part II, Title VI. Part 4 of the Penal Code no. 40/2009 Coll.

In Brno ....

(author signature)

CONTENT

List of figures i

Preface 8

1 Introduction 9

1.1 Ddos. Mitigation methods 9

1.1.1 UDP Flood Attack 9

1.1.2 SYN Flood Attack 10

1.1.3 ICMP Flood Attack 12

1.2 Linux-based firewall 13

1.2.1 Netfilter. Packetflow 13

1.2.2 Ebtables 14

1.2.3 Iptables 16

1.3 Aggregation 18

2 Preparaton **20**

2.1 Experimental network 20

2.2 Tunning 24

3 Implementation **25**

3.1.1 SYN Flood 26

3.1.2 UDP Flood 30

3.1.3 ICMP Flood 34

4 Conclusions **38**

REFERENCES **39**

LIST OF SYMBOLS, VALUES AND GLOSSARY **40**

LIST OF FIGURES

Figure 1.1 Establishing TCP connection........ 10
Figure 1.2 Principle of SYN Flood attack........ ..11
Figure 1.3 Packets flow through netfilter 13
Figure 1.4 Link aggregation. 18
Figure 2.1 Testing network infrastructure 20
Figure 2.2 Links aggregation and interface bonding........ 23
Figure 2.3 Increasing number of rx_queues example 24
Figure 2.4 Ring buffers size........ 24
Figure 3.1 Response time for HTTP request 25
Figure 3.2 Legitimate traffic generation 26
Figure 3.3 Legitimate traffic under SYN Flood 26
Figure 3.4 SYN Flood Filtering by ebtables 27
Figure 3.5 Iptables filtering........ 28
Figure 3.6 Response time under SYN flood........ 28
Figure 3.7 CPU utilization 28
Figure 3.8 SYN flood with smp_affinnity........ 29
Figure 3.9 Web server response time under UDP Flood 30
Figure 3.10 CPU utilization by user traffic 30
Figure 3.11 Web server response time with filter........31
Figure 3.12 Filtering UDP with ebtables........ ..31
Figure 3.13 User traffic response time 32
Figure 3.14 CPU load under UDP Flood with iptables........ 32
Figure 3.15 Response time under UDP Flood with iptables........33
Figure 3.16 Maximum udp traffic throughput........33
Figure 3.17 Web server load under ICMP Flood........34
Figure 3.18 HTTP traffic response time........34
Figure 3.19 ICMP Flood filtering using ebtables........ .35
Figure 3.20 CPU load under UDP filtering........ ..35
Figure 3.21 Response time under iptables filtering........ 36
Figure 3.22 ICMP Flood filtering with iptables 36

List of Tables

Table 2.1 Filters specifications 20

Table 2.2 User traffic 25

Table 3.1 SYN Flood filtering 29

Table 3.2 UDP Flood filtering 33

Table 3.3 ICMP Flood results 36

PREFACE

According to Akamai State of the Internet / Security Report the frequency of ddos attacks has increased by 71% worldwide in 2016. Along with recent event, involving *WannaCrypt* malicious software spreading, security and protection questions for internet users, becoming more important than ever before. Purpose of this diploma work to find out and test most efficient and reliable tools existing in Linux based systems for filtering and aggregation ddos attacks. Linux based solutions are considered as chip and easy to configured systems among available competitor. Most common Web servers focused ddos attacks will be taken into consideration such as SYN/UDP Flood and ICMP Flood. By configuring filter servers and applying suitable setup, most efficient and reliable solution will be chosen. Aggregation of traffic will be considered from a point of impact on filtering productivity.

Master thesis consist of three parts. Ddos attacks overview, most common types and methods of mitigations, available Linux based solutions for traffic filtering and aggregation will be represented in part 1. Next part will describe experimental network components and kernel tuning. Last part will include implementation of selected solution on filter servers, differentiated by installed hardware.

1 INTRODUCTION

1.1 DDOS. MITIGATION METHODS.

A distributed denial-of-service (DDoS) attack [1] is malicious attempt to make an online service unavailable to users, usually by temporarily interrupting or suspending the services of its hosting 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 traffic to a target services which makes ddos mitigation complex, as it's hard to distinguish legitimate traffic from malicious.

According to 2016–2017 Global Application & Network Security Report by Radware [2], most wildly used types of attack in 2016 were SYN Flood, UDP and ICMP Flood.

1.1.1 UDP Flood

User datagram Protocol (UDP) is a connectionless networking protocol, providing checksums for data integrity and port numbers for addressing functions [1]. In the absence of an initial handshake, to establish a valid connection, there is no guarantee of data delivery, ordering or duplicate protection. Thus high volume of “best effort” traffic can be sent over UDP channels to any host, with no built-in protection to limit the rate of the UDP DDoS flood. This means that not only are UDP flood attacks highly-effective, but also that they could be executed with a help of relatively few resources.

In UDP Flood attack attacker sends large number of UDP packets to a victim system, which leads to network saturation and the depletion of available bandwidth for legitimate service requests to the victim system [1]. When the victim system receives a UDP packet, it will determine what application is waiting on the destination port. When server determines that there is no application listening on the port, it will generate an ICMP packet of “destination unreachable” to the forged source address. If enough UDP packets are delivered to ports of the victim, system will go down.

Another way to perform an attack is to send huge amount of UDP packet on certain (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 destined to target server, except DNS.

1.1.2 SYN Flood

SYN Flood attack aiming on TCP three-way handshake mechanism, by sending connection requests faster than target machine can process them, causing network saturation [1]. Under normal conditions clients system begins by sending a SYN message to the server. The server then acknowledges the SYN message by sending SYN-ACK message to the client. The client then finishes establishing the connection by responding with an ACK message. The connection between the client and the server is then open, and the service-specific data can be exchanged between the client and the server. Fig.1.1 shows the view of this message flow:

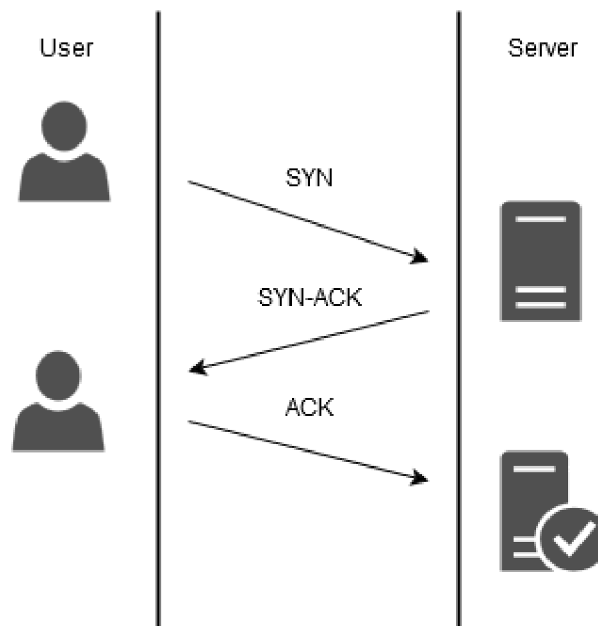


Figure 1.1 Three-way handshake mechanism

The potential weak point is where the server system has sent an acknowledgment (SYN-ACK) back to client but has not yet received the ACK message. This is known as half-open connection.

The server has built in its system memory a data structure (SYN queue) describing all pending connections [3]. This data structure is of limit size, and it can be overflowed by intentionally creating too many partially-open connections. Creating half-open connections is easily accomplished with IP spoofing. The attacking system sends SYN messages to the victim server system; these appear to

be legitimate but in fact reference a client system that is unable to respond to the SYN-ACK messages. This means that the final ACK message will never be sent to the victim server system, as shown in Fig.1.2:

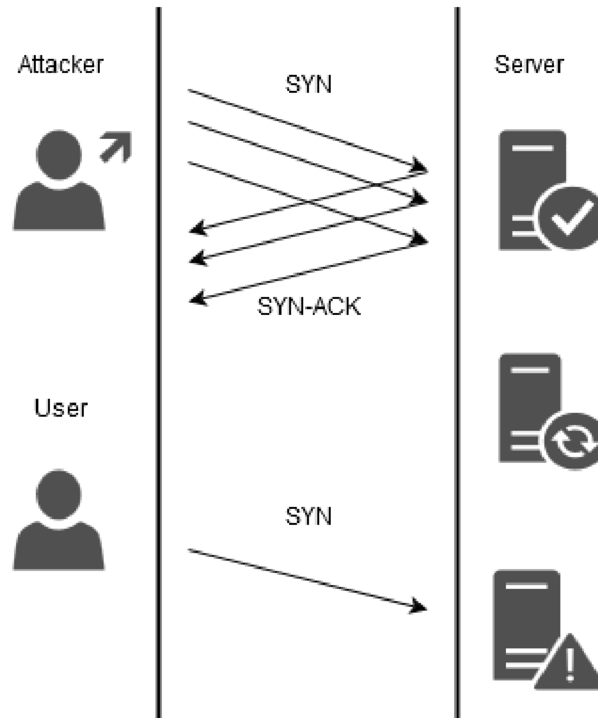


Figure 1.2 Principle of SYN Flood attack

Among possible solutions for mitigation this kind of attacks are using firewalls with SYN cookie feature enabled, filtering by limitation of possible SYN packets per second passing accepted by server and blocking attacker source IP addresses.

SYN cookies is technique how initial TCP sequence numbers will be chosen by TCP servers, developed for mitigating SYN flood attacks[3]. Basic differences in initial sequence number created by server and client are [4]:

- Top 5 bits: t mod 32, where t is a 32-bit time counter that increases every 64 seconds;
- Next 3 bits: an encoding of an MSS selected by the server in response to the client's MSS;
- Bottom 24 bits: a server-selected secret function of the client IP address and port number, the server IP address and port number, and t.

Thus server which uses SYN cookies doesn't have to drop connections when its SYN queue is filled. It will send back a SYN+ACK, exactly as if the SYN queue

was larger. When the server receives ACK, it checks that the secret function works for a recent value of t, and then rebuilds the SYN queue from the encoded MSS.

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

1.1.3 ICMP Flood

Internet control message protocol (ICMP) is used by devices, includes router, to send operational information messages for maintaining networks, such as diagnostic or control purposes [4]. ICMP Flood attack can be performed in "ping of dead" way, by sending large volumes of ICMP_ECHO_ REQUEST messages to the victim, which force target server to reply and thus results in saturation of victim bandwidth network connection. During an ICMP flood attack the source IP address may be spoofed.

Possible solutions for mitigations are limit size of ping requests as well as the rate at which they can be accepted and denying all Icmp_Echo_Requests from all source ip addresses, except local network.

1.2 LINUX-BASED FIREWALL

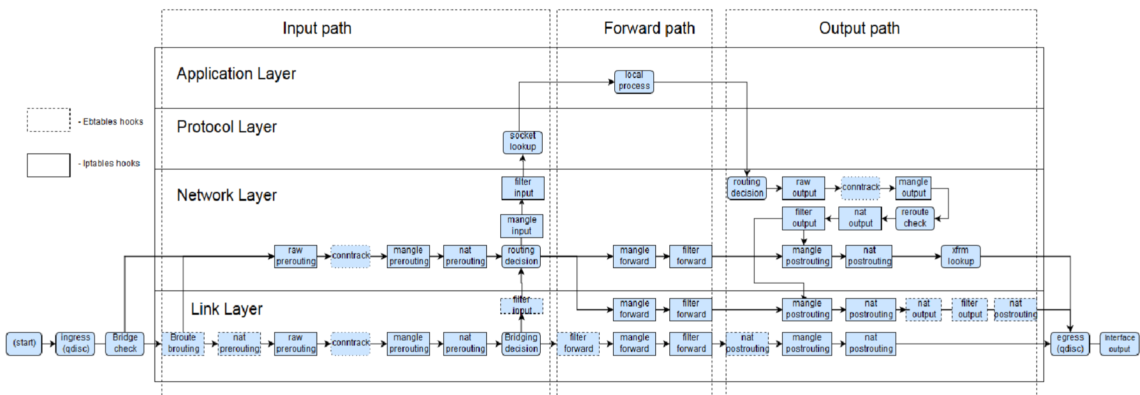
While deciding which best solution to choose for traffic filtering and ddos attacks mitigations, Linux kernel based products can become more preferable considering price tag and user friendly environment. There are a lot of tools and tuning options for working with traffic, on which we need to have a closer look.

Linux distributions are operating systems made from software collection [5], based on Linux kernel and package management system. Linux kernel is a computer program and is the core of an operating system, with complete control over everything in the system.

Base for all manipulating with network stack in linux kernel, is netfilter. To be able to apply suitable techniques for filtering traffic, it's crucial to know, how incoming and outgoing packet to linux network stack are being handled by netfilter.

1.2.1 Netfilter. Packetflow

Netfilter is a framework provided by the Linux kernel and represents a set of hooks inside it, to allow specific kernel modules to register callback functions with the kernel's networking stack [5]. General packets flow path through netfilter's hooks is shown on pic.1.3:



Picture 1.3 Packets flow through netfilter

Let's have a closer look on most essential functions for our testing purposes:

- Qdisc – scheduler and major building block in Linux traffic control process. It queues all packets based on appropriate queuing discipline and transmits packet as soon as it can. There are ingress (inbound traffic) and egress

(outbound traffic) qdiscs. The default queuing discipline for all interfaces under Linux is *pfifo_fast*. It based on a conventional FIFO qdisc and provides prioritization. There are three different bands for separating traffic. The highest priority traffic are placed into band 0 and are always serviced first.

- Bridge check - simply check, if interface, from which packet was received, belongs to bridge or not. If so, frame will not be processed at this point and will be send to bridging decision function.
- Conntrack - connection tracking subsystem. It stores information about the state of a connection, including source and destination IP addresses, port number pairs, protocol types, state, and timeout in structured memory [5].
- Bridging decision - at this point frame is being investigated whether it's destination is local process or its destination MAC address is located on another side of the bridge. It can do with frame 4 thing:
    1. Bridge it
    2. Flood it over, if the destination MAC address is unknown to the bridge.
    3. Pass it to the higher protocol code (IP code)
    4. Ignore it, if a destination MAC address is on the same side of the bridge.
- Routing decision - based on IP address it decides if packet destination is local process or it should be forwarded. Packet will be send through bridge interface at this point, if forwarded.

1.2.2 Ebtables

Linux kernel built-in filtering tool, since kernel version 2.2 [7], which allows to set up and maintain tables of rules that inspect Ethernet frames. It's similar to iptables tools but with less functions due to fact that Ethernet frame header is less complex than IP packet header. Ebtables rules are working only with bridged frames and compare to iptables , frame is processed earlier in the stack, consuming less resources.

There are three tables named *filter*, *nat* and *broute* [7]. Syntax for managing with ebtables rules is the same as with iptables rules.

- *Filter* is the default table and contains three built-in chains: INPUT (for frames destined for the bridge itself), OUTPUT (for locally-generated or (b)routed frames) and FORWARD (for frames being forwarded by the bridge).
- *Nat* is mostly used for changing the mac addresses and contains three built-in

chains: PREROUTING (for altering frames as soon as they come in), OUTPUT (for altering locally generated or (b) routed frames before they are bridged) and POSTROUTING (for altering frames as they are about to go out).

- *Broute* table has one built-in chain: BROUTING. The targets DROP and ACCEPT have a special meaning in the broute table. DROP actually means the frame has to be routed, while ACCEPT means the frame has to be bridged. The BROUTING chain is traversed very early. However, it is only traversed by frames entering on a bridge port that is in forwarding state.

Now let's have a closer look on command line arguments [7], which will be used in this work.

- **-p, --protocol** - The protocol which is responsible for creating frame. It can be hexadecimal number, above 0x0600, name (arp) or LENGHT. When the value of protocol field is below or equal 0x0600, the value equal the size of the header and shouldn't be used as protocol number. Readable names with corresponding hexadecimal numbers can be find under */etc/ethertypes*. For example, *0x0800* will be represented by *ipv4* (not case sensitive).
- **-i, --in-interface** - The interface (bridge port) via which a frame is received.
- **--ip-source/--ip-destination** - The source and destination ip addresses.
- **--ip-sorce-port/--ip-destination-port** - The source port or port range for the IP protocols 6 (TCP), 17 (UDP) or 132 (SCTP). The **--ip-protocol** option must be specified as *TCP*, *UDP*, *DCCP* or *SCTP*.
- **--limit** - Maximum average matching rate: specified as a number, with an optional */second*, */minute*, */hour*, or */day* suffix; the default is *3/hour*.
- **--limit-burst** - Maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is *5*.
- **--log-ip** - Will log the ip information when a frame made by the ip protocol matches the rule. The default is no ip information logging.

1.2.3 Iptables

Netfilter iptables is a user-space command line utility to configure packet filtering rules [7]. It's the default firewall management utility on Linux systems. Iptables is used to set up, maintain, and inspect the tables of IPv4 packet filter rules in the Linux kernel. There are four tables *filter*, *nat*, *mangle* and *raw*. Each table contains a number of built-in chains and may also contain user-defined chains. Each chain is a list of rules which can match a set of packets. Each rule specifies target [7], meaning what action to do with a packet that matches.

- **Filter** - default table. It contains the built-in chains INPUT (for packets destined to local sockets), FORWARD (for packets being routed through the box), and OUTPUT (for locally-generated packets)
- **Nat** - table is consulted when a packet that creates a new connection is encountered. It consists of three built-ins: PREROUTING (for altering packets as soon as they come in), OUTPUT(for altering locally-generated packets before routing), and POSTROUTING (for altering packets as they are about to go out)
- **Mangle** - table is used for specialized packet alteration. Since kernel 2.4.18, three other built-in chains are also supported: INPUT (for packets coming into the box itself), FORWARD (for altering packets being routed through the box), and POSTROUTING (for altering packets as they are about to go out), PREROUTING (for altering incoming packets before routing) and OUTPUT (for altering locally-generated packets before routing)
- **Raw** - This table is used mainly for configuring exemptions from connection tracking in combination with the NOTRACK target. It registers at the netfilter hooks with higher priority and is thus called before *ip_conntrack*, or any other IP tables. It provides the following built-in chains: PREROUTING (for packets arriving via any network interface) OUTPUT (for packets generated by local processes).

Now let's have a closer look on most widely used arguments[7], which were used in this work.

- **-p, --protocol** - Protocol of the rule or of the packet to check. The specified protocol can be one of tcp, udp, udplite, icmp or the special keyword "all". Also it can be a numeric value, representing one of these protocols or a different one. A protocol name from /etc/protocols is also allowed. A "!" argument before the protocol inverts the test. "All" will match with all protocols and is taken as default when this option is omitted.

- **-s, --source** - Source specification. Address can be either a network name, a hostname, a network IP address (with **/***mask*), or a plain IP address. Hostnames will be resolved once only, before the rule is submitted to the kernel. The *mask* can be either a network mask or a plain number, specifying the number of 1's at the left side of the network mask. A "!" argument before the address specification inverts the sense of the address.
- **-m, --match** - Specifies a match to use, that is, an extension module which tests for a specific property. The set of matches make up the condition under which a target is invoked. Matches are evaluated first to last as specified on the command line and work in short-circuit fashion, i.e. if one extension yields false, evaluation will stop.
- **--src-range** - Match source IP in the specified range.
- **--limit** - Maximum average matching rate: specified as a number, with an optional `/second', `/minute' etc.
- **--limit-burst** - Maximum initial number of packets to match: this number gets recharged by one every time the limit specified above is not reached, up to this number; the default is 5.
- **--destination-ports,--dports** - Match if the destination port is one of the given ports. Multiple ports or port ranges are separated using a comma, and a port range is specified using a colon
- **-m recent --hitcount** - This option must be used in conjunction with one of **--rcheck** or **--update**. When used, this will narrow the match to only happen when the address is in the list and packets had been received greater than or equal to the given value. This option may be used along with **--seconds** to create a match requiring a certain number of hits within a specific time frame. The maximum value for the hitcount parameter is given by the "ip_pkt_list_tot" parameter of the xt_recent kernel module. Exceeding this value on the command line will cause the rule to be rejected.
- **--set** - This will add the source address of the packet to the list. If the source address is already in the list, this will update the existing entry. This will always return success (or failure if **!** is passed in).
- **--state** - Where state is a comma separated list of the connection states to match. Only a subset of the states understood by "conntrack" are recognized: INVALID, ESTABLISHED, NEW, RELATED or UNTRACKED.
- **--syn** - Only match TCP packets with the SYN bit set and the ACK,RST and FIN bits cleared. Such packets are used to request TCP connection initiation; for example, blocking such packets coming in an interface will prevent incoming

TCP connections, but outgoing TCP connections will be unaffected. It is equivalent to --tcp-flags SYN, RST, ACK, FIN SYN.

- **--destination-port**,**--dport** - Destination port or port range specification.

1.3 LINK AGGREGATION

Link aggregation include various methods of combining multiple network connections in parallel in order to increase throughput beyond what a single connection could sustain, and to provide redundancy in case one of the links should fail [13]. A LAG (Link Aggregation Group) combines a number of physical ports together to make a single high-bandwidth data path, so as to implement the traffic load sharing among the member ports in the group and to enhance the connection reliability. Principle of link aggregation is shown in fig.1.4:

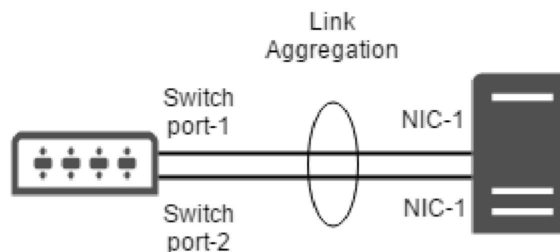


Figure 1.4 Link aggregation

Most important benefits from link aggregation:

1. Multiple Connections With Little Speed Loss.

   Single file transfers or one-at-a-time transfers do not benefit much 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 encouter 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 speeds until the issue is resolved.

3. Load Balancing

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

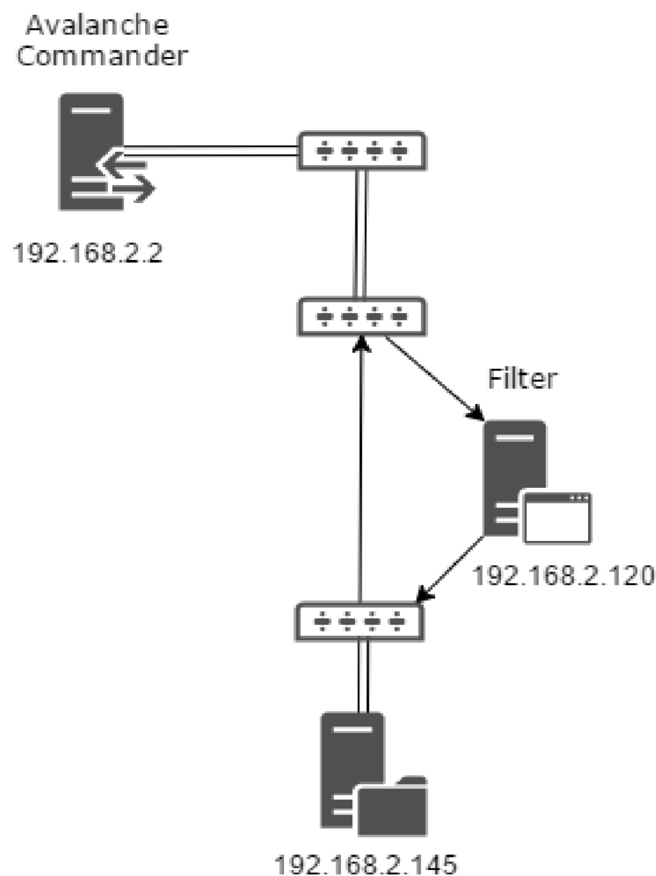
Linux bonding provides a method for aggregating multiple network interfaces (*slaves*) into a single logical bonded interface (*bond*). Linux supports two bonding methods[17]:

- The IEEE 802.3ad link aggregation mode, which allows one or more links to be aggregated together to form a *link aggregation group* (LAG), such that a media access control (MAC) client can treat the link aggregation group as if it were a single link.
- The balance-xor mode, where the bonding of slave interfaces are static and all slave interfaces are active for load balancing and fault tolerance purposes.

2 PREPARATON

2.1 Experimental network

All tests and measurements carried out on a specifically design and built network, equipped with network tester "Spirent Avalanche 3100B", which allows generating traffic by 1Gbit/s links. General view of network is shown on pic.2.1.



Picture 2.1 - Testing network infrastructure

Filter is presented in a form of three separated server machines which are connected in parallel. Same filtering tools will be used on all machines to compare hardware influence and link aggregation on filtering process. Hardware specifications for filters show in tab.2.1:

	Filter V1	Filter V2	Filter V3 Aggregation
Number of cores	4	16	2
Clock speed(min/max)	2800/3600	1600/2200	3400
Network Interface Controller	2x NC7782 Gigabit Server Adapter	2x 10-Gigabit X540-AT2	4x 82546EB Gigabit Ethernet
NIC drivers	E1000	Ixgbe	E1000
Bus	PCI-X(66-MHz)	PCIe(16)	PCI-X

Table 2.1 - Filters specifications

PCI-X - 64-bit parallel computer bus with theoretical maximum of 1.06 GB/s data exchange speed between computer processor and peripherals.

PCIe is a serial point-to-point connection bus with possible 4 GB/s bandwidth in each direction.

Ixgbe - NIC driver with abilities to reduces the number of queues per interface-direction to the number of logical CPUs. The reasoning for this reduction is that each queue entails some overhead, and there is no advantage in maintaining more queues than there are CPUs

As a filter machines software was used one of linux based distributions, Debian OS, as it is composed entirely of free software most of which is under GNU General Public License.

In order to have traffic passing through filtering server, first bridge needed to be configured. Configuration file for all interfaces is located at */etc/network/interfaces*. There are 3 network interfaces on current filtering server. One is serving for VPN connection (remote control), other two services for carrying traffic.

Needed bridge configuration for filter V1 should be as follows:

```
# The primary network interface
auto eth0
iface eth0 inet static
   address  12.12.13.26
   netmask  255.255.0.0
   gateway  12.12.12.254

#auto eth1
```

```
iface eth1 inet manual

#auto eth3
iface eth3 inet manual

#Bridge setup
auto br0
iface br0 inet static
  bridge_ports eth1 eth3
  address    192.168.2.120
  broadcast  192.168.2.255
  netmask   255.255.255.0
  gateway   192.168.1.121
```

Configuring Filter V2:

```
auto eth1
iface eth1 inet static
address 147.229.7.213
 netmask 255.255.255.248
 gateway 147.229.7.209
dns-nameservers 147.229.71.10
#auto eth2
iface eth2 inet manual


#auto eth3
iface eth3 inet manual


auto br0
iface br0 inet static
 bridge_ports eth2 eth3
 address 192.168.2.110
 broadcast 192.168.2.255
 netmask 255.255.255.0
 gateway 192.168.2.111
```

Filter V3 setup:

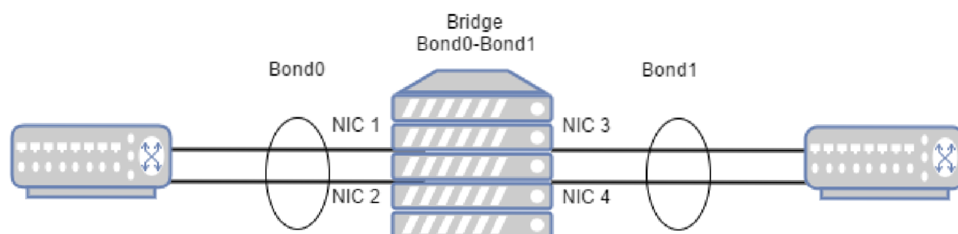
```
source /etc/network/interfaces.d/*
# The loopback network interface
auto lo
iface lo inet loopback
```

```
# The primary network interface
auto enp3s1f0
allow-hotplug enp3s1f0
iface enp3s1f0 inet static
 address 12.12.13.24
 netmask 255.255.0.0
 network 12.12.0.0
 broadcast 12.12.255.255
 gateway 12.12.12.254

auto eth1
iface eth1 inet static
   address 192.168.2.117
   netmask 255.255.255.0

auto eth3
iface eth3 inet static
  address 192.168.2.116
  netmask 255.255.255.0
```

Bonding interface setup , as follows:



Picture 2.2 - Links aggregation and interface bonding

To be able to affect bridged frames, we needed to install ebtables tool and briedge-netfilter infrastructure. Also for tracking bandwidth and cpu utilization, tools such as "nload" and"htop" will be installed on all servers.

```
#apt-get install bridge-utils
#apt-get install ebtables
#apt-get install htop
#apt-get install nload
```

2.2 Tunning

In order to increased traffic throughput of NIC and linux kernel, we need to improve packet reception process and all filters.

First check multiqueue mode and enable, if it's possible on receiving network devices:

```
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
```

Picture 2.3 - Increasing number of rx_queues example

Incoming frames on eth3 will be processed in 63 queues, which increase possible amount of filtering traffic.

Increasing NIC ring buffers size:

```
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
```

Table 2.4 Ring buffers size

Thus will help prevent network data drops at the NIC during periods, when large numbers of data frames are received.

Enabling Receive Packet Steering:

```
#echo 1 > /sys/class/net/eth3/queues/rx-0/rps_cpus
```

Prevents the hardware queue of a single network interface card from becoming a bottleneck by creating hash from the IP addresses and port numbers, which it then uses to determine to which CPU to send the packet.

The use of the hash ensures that packets for the same stream of data are sent to the same CPU, which helps to increase performance.

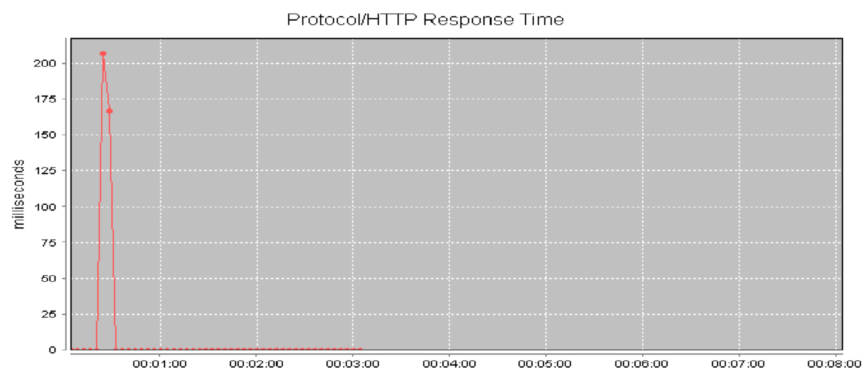
3 IMPLEMENTATION

To make testing consistent, we will start tests from layer 2 OSI model, meaning 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 to filter Ethernet frames and iptables rules to filter IP packets as they are well-known and reliable.

Legitimate 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 on table.3.1, pic.3.3 and 3.2:

Transactions				Time (ms)					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	0.0	0.094	0.225	0.0	Attempted	91879
Successful	86031	460	Maximum	6017.0	6017.0	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								

Table 3.1 - User traffic



Picture 3.1 - Response time for HTTP request

First big spike corresponding to unsuccessful transactions tab which are related with Avalanche Commander specific functioning. Amount of traffic generated by legitimate users on WEB server is show on pic.3.2:

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

Picture 3.2 Legitimate traffic generation

3.1 SYN Flood

In this test Avalanche Commander is sending packets with SYN flags set from ip addresses range 192.168.2.163-192.168.2.167. By generating a big amount of SYN Flood packets along with legitimate traffic it able to increased server response time, which is shown on pic.3.3 and:

Protocol/HTTP Response Time

Picture 3.3 - Legitimate traffic under SYN Flood

Basically, there is no way to distinguish malicious packets trafic with SYN flag set on the Ethernet layer, so for filtering needed detections system which will provide us with malicious IP addresses:

```
#ebatables -F
#ebtables -N syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp --ip-dport 80 -j syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp --ip-dport 443 -j syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp -j DROP
#ebtables -A syn_flood -p ipv4 --among-src-file data -j DROP
#ebtables --atomic-file syn_flood -t filter --atomic-save
```

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 on filter server and how much is dropped. Pic.3.4 is showing desired:

```
toor@filter-pb-2p: ~
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
```

Picture 3.4 SYN Flood Filtering by ebtables

As we see only user 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, which corresponds to maximum throughput of NIC. Another way to filter syn flood is limiting passing traffic based on packets/s, to decrease some load from target server. This solution is affecting user traffic also and is not preferable.

By default, only ebtables code is able to process bridged frames, so to let iptables rules receive traffic from bridged ports, we need to enable *bridgen-nf-call-iptables* feature.

To compare filtering with iptables rules based on ip addresses:

```
#iptables –F
#iptables –P FORWARD ACCEPT
#iptables -N syn_flood
#iptables -A FORWARD -p tcp –m tcp --syn  -j syn_flood
#iptables -A syn_flood -m iprange --src-range 192.168.2.163-
 162.168.2.167 -j DROP
```

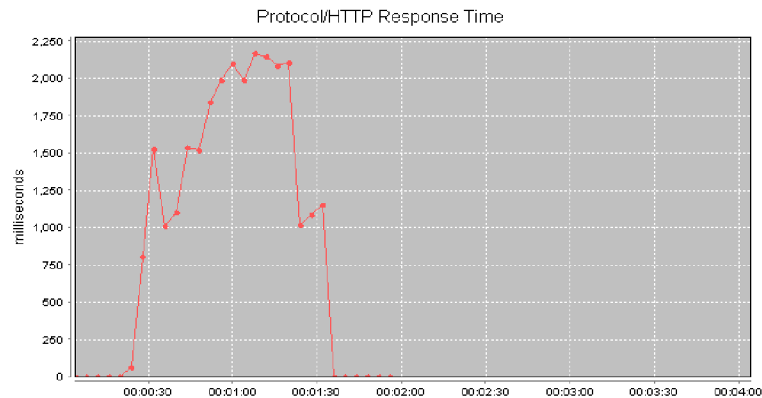
The result is shown on pic.3.5:

```
toor@filter-pb-2p: ~
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
```

Picture 3.5 SYN Flood with Iptables filtering

Although we are able to filter traffic on speed 90-100Mbit/s, the web server response time is still greatly increased due to CPU overloading, which is made by *ksoftirqd* - per-cpu kernel thread that runs when the machine is under heavy soft-interrupt load. Increased response time and CPU utilization is shown on pic.3.7 and 3.8:



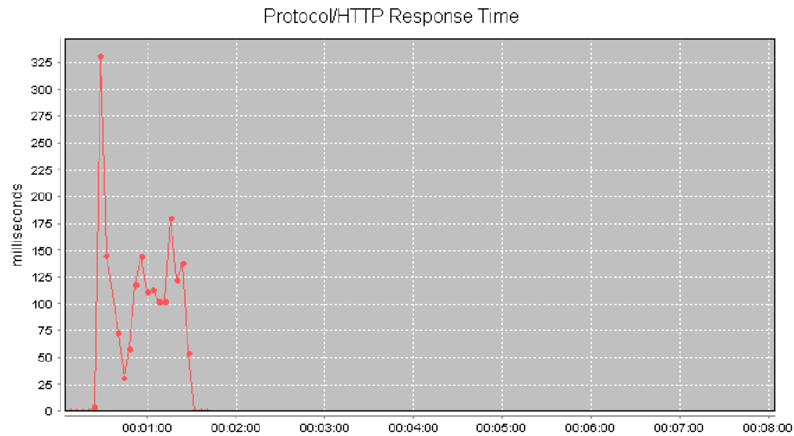
Picture 3.6 Response time under SYN flood

```
toor@filter-pb-2p: ~
  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
```

Picture 3.7 CPU utilization

Previously we disabled rpc_cpus feature to test cpu utilization. Now we enable it back an run same test again. The result is shown on pic.3.8:



Picture 3.8 – SYN flood with smp_affinnity

Only 3 out of 4 cores are loaded with processing packets, which decreasing response time to acceptable range. However maximum amount of possible filtered traffic is increased from 90Mbit/s to 130Mbit/s. Assuming using more complex iptables rules with same user traffic will lead to denial of service.

More complex rule with using conntrack tool inside linux kernel, as follows:

```
#iptables -F
#iptables -N syn_flood
#iptables -A FORWARD -p tcp -m state --state NEW -j syn_flood
#iptables -A  syn_flood -m connlimit --connlimit-above 500 -j DROP
```

Rule sets limit on amount connections per second coming from one ip address, assuming all 500 users will start simultaneously sending connection requests. The traffic filtering speed in this case is around 100Mbit/s with enabled load spread.

Same tests were conducted on all filter servers and compared result are shown in tab.3.1:

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

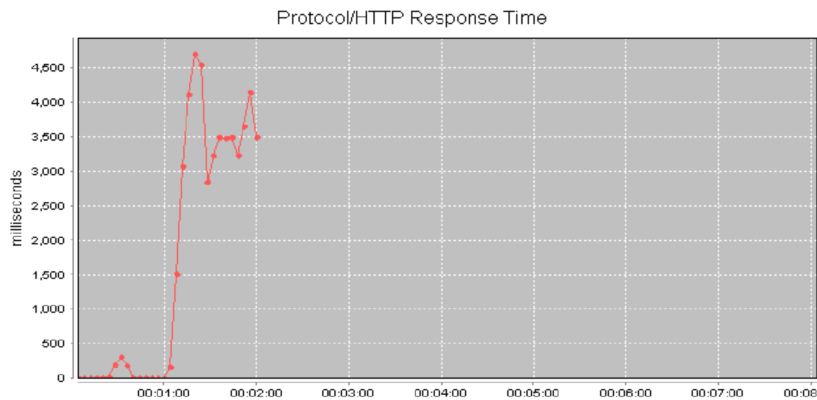
Table 3.1 - SYN Flood filtering

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 rulse we are able to filter in more stand-alone way but it requires more hardware resources to use in order to filter same amount of traffic per second. Filter with aggregated links can benefit from having 1Gbps links instead of one, since the bottleneck is not in

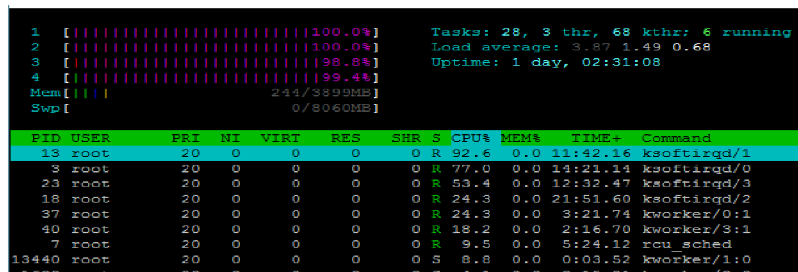
filtering. More of that due to less computing resources it show lower filtering throughput.

3.2 UDP Flood

UDP Flood testing in our network based on sending as much packets as possible on web server port 80 with spoofed source ip addresses. Main goal is to utilize all filter server CPU usage. Effect from generating UDP Flood can be seen on pic.3.9 and pic.3.10:



Picture 3.9 - Web server response time under UDP Flood



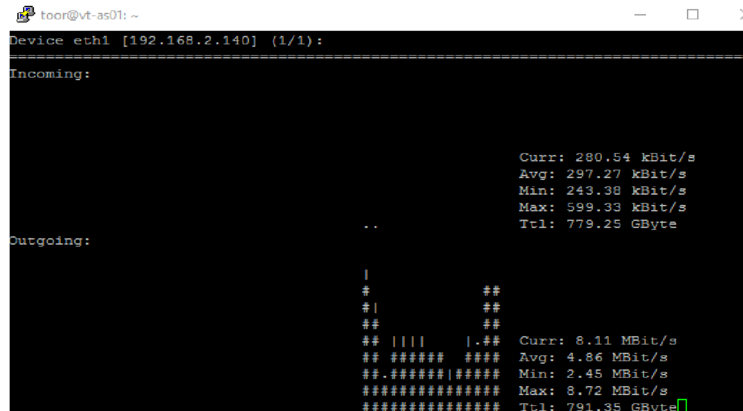
Picture 3.10 - CPU utilization by UDP Flood

On link layer to filter UDP flood we able to set following rules:

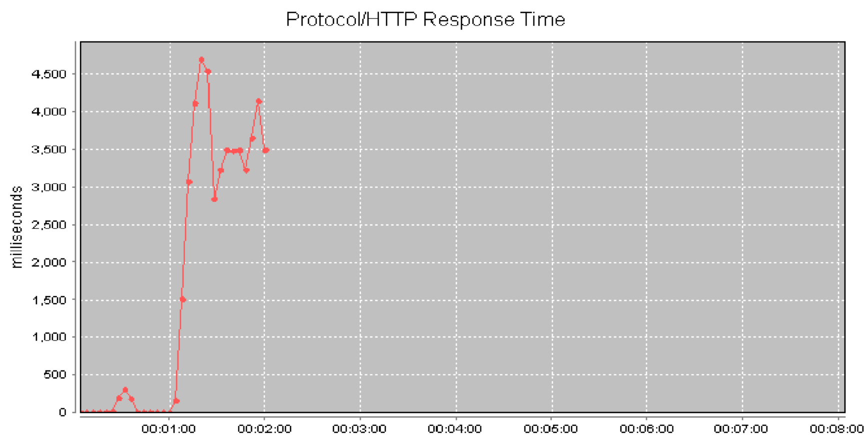
```
#ebtbales - F
#ebtables - N udp_flood
#ebtables - P udp_flood DROP
#ebtables - A FORWARD -p ipv4 --ip-proto udp  ! --ip-dport 53 -j udp_flood
```

Udp packets which are destined to DNS server on port 53 will pass, all others udp packet should be dropped.

Results are shown on pic.3.11 and 3.12:



Picture 3.11- Web server user load



Picture 3.12 - Web server response time with filter

As can be seen on WEB server traffic still struggling to pass through filter. All 4 filter server cores are loaded with ksoftirqd program and filtering is possible but web server has still big response delay.

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

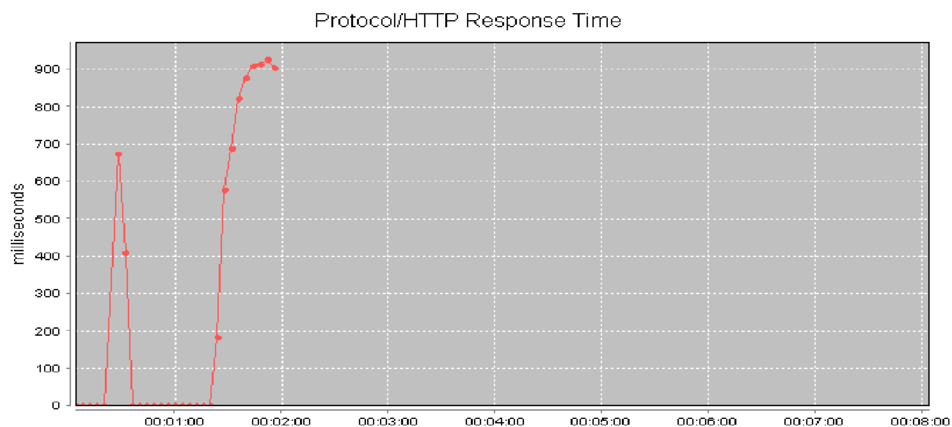
Applying same rules in table nat PREROUTING chain results in better performance, which can be seen on pic. 3.13 and 3.14:

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
```

Picture 3.13 - Filtering UDP with ebtables



Picture 3.14 - User traffic response time

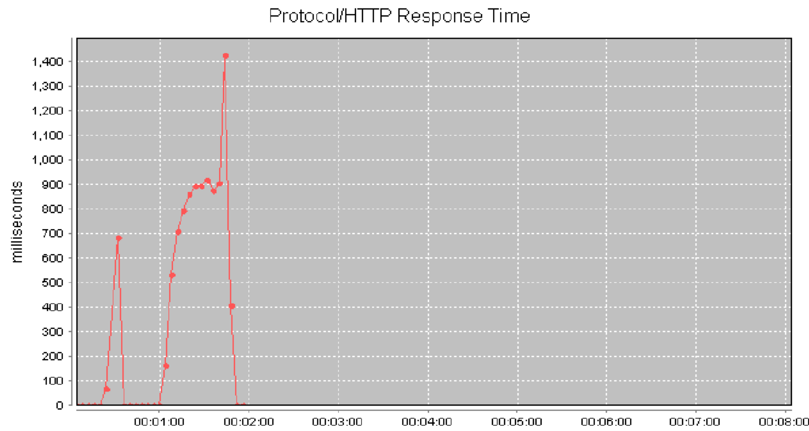
Part of UDP Flood was decreased along with http response time. Although it stays in acceptable range under 1 second further countermeasures in network are required.

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

```
#iptables - F
#iptables -t raw -A PREROUTING -p udp --dport 53 -d 192.168.2.145  -j
ACCEPT
#iptables -t raw -A PREROUTING -p udp -d 192.168.2.145 -j DROP
```

All passing udp packets going to web server should be dropped, except destined to DNS.

Results output is shown on pic.3.15 and 3.16:



Picture 3.15 - Response time under UDP Flood 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
```

Picture 3.16 - Maximum udp traffic throughput

How can be seen both ebtables and iptables are able to filter UDP flood attack. 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 servers same tests were conducted and result is combined in tab. 3.2:

UDP Flood(Mbit/s)	Filter V1	Filter v2	Filter V3
Ebtables	67	67	67
Iptables	68	64	65

Table 3.2 - UDP Flood filtering

As this attack doesn't consume much resources, each server was able to completely filter UDP flood on maximum available througput.

3.3 ICMP Flood

ICMP attack relies on constantly sending echo_request to force our web server to respond and consume additional resources.

Generating attack along with sending user traffic also consumes, as shown on followed pic.3.17 and pic.3.28 show:

Picture 3.17 - Web server load under ICMP Flood

Picture 3.18 - HTTP traffic response time

ICMP traffic is crucial for network control and it can't be dropped at all. Best way to filter on link layer is to set limit:

```
#ebtables -F
#ebtables -N icmp_flood
#ebtables -A FORWARD -p ipv4 --ip-proto icmp -j icmp_flood
#ebtables -A icmp_flood -p ipv4 --limit 2/s --limit-burst 10 -j ACCEPT
#ebtables -A icmp_flood -j DROP
```

As a standard *icmp_echo* request rate is 12 packet per 10 second, our limit is letting users to ping server. Pic.3.19 and pic.3.20 describing filtering process:

```
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
```

Picture 3.19 - ICMP Flood 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
```

Picture 3.20 - CPU load under UDP filtering

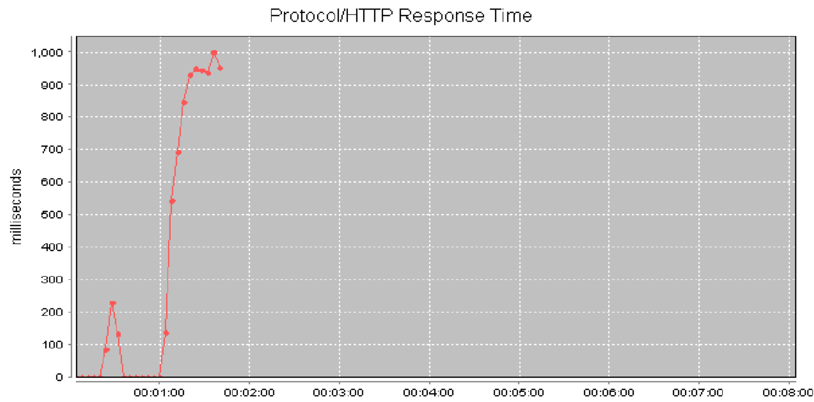
As can be seen there is no restriction from hardware on filtering side, but still there is a small delay related with packets processing.

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.

Needed rule as follows:

```
#iptables -N icmp_flood
#iptables -A FORWARD -p icmp -j icmp_flood
#iptables -A icmp_flood -p icmp --icmp-type echo-request -s 192.168.2.0/24 -j ACCEPT
# iptables -A icmp_flood -j DROP
```

Desired result are described in pic.3.21 and pic.3.22:



Picture 3.21 - Response time under iptables filtering

```
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
```

Picture 3.22 - ICMP Flood filtering with iptables

As expected all malicious ICMP traffic was filtered with adding a small delay to response time.

The result for rest of the servers are described in tab.3.3:

	Filter V1	Filter V2	Filter V3
ebtables	68	70	69
iptables	68	68	68

Table 3.3 - ICMP Flood results

A tiny part of computing resources is required for processing icmp flood, which, makes this type of attack filtering less complex and available across networks worlwide.

4 CONCLUSION

Main goal of master thesis, was to test appropriate tools for filtering traffic. Maximum achieved traffic throughput while filtering SYN Flood attack is 187 Mbit/s, UDP Flood max. throughput is 67 Mbit/s and ICMP Flood was filtered in max. 71 Mbit/s.

Considering received data, in this master thesis, it can be said that linux-based solution provide full scale of tools needed for filtering traffic. For example, SYN Flood ddos attack can be mitigated by using packet filtering with iptables without need of using any additional hardware or software. However, by installing additional detecting systems in network, better filtering performance can be achieved by using ebtables tool. The difference lies in packets flow inside the kernel network stack. UDP Flood ddos attacks can be mitigated using any available tool, but requirements for additional kernel tuning exists. Opportunity for links aggregation using linux software can be used in cases, where highly loaded network is being used. Additional speed can be achieved for user experience with using so called “bonding’.

In real-time environment proposed ddos mitigation techniques should be consider as a temporarily measurements, due to limit resources capacity in provided systems. Recent average ddos rates in world are far beyond capabilities of any single filtering hardware or software. Thus to successfully filter malicious traffic is needed great cooperation among all parties, involved in global WEB work.

REFERENCES

[1] F. Lau, S. H. Rubin, M. H. Smith and L. Trajkovic, "*Distributed Denial of Service Attacks*" IEEE International Conference on Systems, Man, and Cybernetics, Nashville, 8-11 October 2000, pp. 2275-2280.

[2] Portál radware.com.[online].2016[cit.2017-05-20]. Available on: https://www.radware.com/newsevents/pressreleases/2017/ert2016-2017/

[3] Portál incapsula.com. [online]. 2013 [cit.2017-05-20]. Available on: https://www.incapsula.com/ddos.

[4] Felix Lau, Rubin H. Stuart, Smith H. Michael, and et al., "*Distributed Denial of Service Attacks*" in Proceedings of 2000 IEEE International Conference on Systems, Man, and Cybernetics, Nashville, TN, Vol.3, pp.2275-2280

[5] Portál linuxaria.com [online]. 2014 [cit. 2016-11-29]. Available on: https://linuxaria.com/article/mitigating-ddos-attacks.

[6] J. Markovic, J. Martin, and L. Reiher, " *ATaxonomy of DDoS Attack and DDoS Defense Mechanisms*" ACM SigComm Computer Communication Review, Vol. 34, No. 2, 2004, pp. 39-53.

[7] Portál ebtables.netfilter.org [online]. 2006 [cit. 2016-11-29]. Available on: http://ebtables.netfilter.org/misc/ebtables-man.html.

[8] CERT advisory CA-1998-01 "*Smurf IP Denial-of-Service Attacks*". Available at http://www.cert.org/advisories/CA-1998-01.html, Jan. 1998

[9] Portál people.netfilter.org [online]. 2006 [cit. 2016-11-29]. Available on: https://people.netfilter.org/pablo/docs/login.pdf.

[10] Portál cr.yp.to. [online]. 2013 [cit. 2016-11-29]. Available on: https://cr.yp.to/syncookies.html.

[11] Portál researchgate.net [online]. 2012 [cit. 2016-11-29]. Available on: https://www.researchgate.net/publication/265181297_Mitigating_DoSDDoS_attacks_using_iptables.

[12] Hayoung Oh, Inshil Doh, Kijoon Chae, " *Attack Classification Based on Data Mining Technique and its Application for Reliable Medical Sensor Communication*" International Journal of Computer Science and Applications, Volume 6, No. 3, pp.20-32, 2009.

[13] Portál wikipedia.org [online]. 2016 [cit. 2016-11-29]. Available on https://en.wikipedia.org/wiki/Link_aggregation.

[14] Portál linux.com [online]. 2015 [cit. 2016-11-29]. Available on https://www.linux.com/what-is-linux

LIST OF SYMBOLS, VALUES AND GLOSSARY

CPU	Central Processing Unit
DDOS	Distributed Denial of Service
HTTP	Hypertext Transfer Protocol, protokol HTTP
SYN	Synchronize sequence numbers
ICMP	Internet Control Message Protocol, protokol ICMP
RPS	Receive Packet steering.
LAG	Link aggregation group
MSS	Maximum segment size
OSI	Open Systems Interconnection
UDP	User Datagram Protocol

LIST OF ANNEXES

A Ebtables rules 42

B Iptables rule 43

A Ebtables rules

```
##SYN Flood mitigation
#ebatables -F
#ebtables -N syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp --ip-dport 80 -j syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp --ip-dport 443 -j syn_flood
#ebtables -A FORWARD -p ipv4 --ip-proto tcp -j DROP
#ebtables -A syn_flood -p ipv4 --among-src-file data -j DROP
##UDP Flood mitigation
#ebtables - N udp_flood
#ebtables - P udp_flood DROP
#ebtables - A FORWARD -p ipv4 --ip-proto udp ! --ip-dport 53 -j udp_flood
##ICMP Flood mitigation
#ebtables -N icmp_flood
#ebtables -A FORWARD -p ipv4 --ip-proto icmp -j icmp_flood
#ebtables -A icmp_flood -p ipv4 --limit 2/s --limit-burst 10 -j ACCEPT
#ebtables -A icmp_flood -j DROP
#ebtables --atomic-file syn_flood -t filter --atomic-save
```

B Iptables rules

```
##SYN Flood protection
#iptables -F
#iptables –P FORWARD ACCEPT
#iptables -N syn_flood
#iptables -A FORWARD -p tcp -m tcp --syn -j syn_flood
#iptables -A syn_flood -m iprange --src-range 192.168.2.163-
 162.168.2.167 -j DROP
##Connection limit
#iptables -N syn_flood
#iptables -A FORWARD -p tcp -m state --state NEW -j syn_flood
#iptables -A syn_flood -m connlimit --connlimit-above 500 -j DROP
//UDP Flood protection
#iptables -t raw -A PREROUTING -p udp --dport 53 -d 192.168.2.145  -j
ACCEPT
#iptables -t raw -A PREROUTING -p udp -d 192.168.2.145 -j DROP
##ICMP Flood
#iptables -N icmp_flood
#iptables -A FORWARD -p icmp -j icmp_flood
#iptables -A icmp_flood -p icmp --icmp-type echo-request -s 192.168.2.0/24 -j
ACCEPT
# iptables -A icmp_flood -j DROP
```