International Journal of Computer Information Systems and Industrial Management Applications.
ISSN 2150-7988 Volume 10 (2020) pp. 023-032

Received: 4 Oct, 2019; Accept: 11 Feb, 2020; Publish: 25 Feb, 2020

Performance Analysis and Comparison of Snort on Various Platforms

Alka Gupta[1], Lalit Sen Sharma[2]

[1] Department of Computer Science and IT,
University of Jammu
alkagupta48@gmail.com

[2] Department of Computer Science and IT,
University of Jammu
lalitsen@yahoo.com

***Abstract*: *Snort* has emerged as a reliable technology for identifying malicious activities in networks. In this paper, a systematic approach has been followed to estimate the performance offered by *Snort*, an open-source network intrusion detection and prevention system on different platforms. Extensive experiments are conducted on Windows Server 2016, *Ubuntu* Server 16.04 and Virtual Windows Server 2016 to identify the characteristics of the network traffic that affects *Snort's* performance. The study establishes the incapacity of *Snort* to cope up with the large packet sizes and high-speed traffic. It is also found that *Snort* has tendency to drop packets on all the Servers for normal as well as malicious traffic but shows better performs on *Ubuntu* Server for both high-speed traffic and different packet sizes. The study experimentally exhibits poor performance of *Snort* on Virtual Windows Server.**

***Keywords*:** NIDS, NIDPS, Snort v2.X, D-ITG, Performance, virtual server.

I. Introduction

An intrusion detection system (IDS) monitors a system or a network for any malicious activity and report any intrusion attempt to the system administrator by generating logs. An Intrusion prevention system (IPS), has an added ability to block the intrusion attempts by either dropping the malicious packet, resetting the connection or blocking the source IP of the malicious packet etc. A system having the properties of both IDS and IPS are known as Intrusion detection and prevention system (IDPS). A good IDPS is characterized by its ability to identify true attacks, less number of false alerts and low value of dropped packets [1].

IDS are generally classified into host-based and network based systems. An IDS that is installed on a single system and monitors incoming traffic to that system only is known as Host-based Intrusion detection system (HIDS) [2]. Network based Intrusion Detection System (NIDS) monitor all the traffic in their network [2] and are installed on a system that receives all the traffic from switch via mirroring port. Incoming packets are matched to its rule-set and in case of a match, an alert is generated and logged. NIDS have better performance as compared to HIDS [2]. *Snort* is a popular and most widely employed IDPS, developed by Martin Roesch. It was initially launched as a lightweight cross-platform packet sniffing device [3] and was upgraded to an IDS in 2003. It is a developmental open source software and has now evolved into a powerful intrusion detection and prevention system. Its latest stable version is 2.9.11 and has more than 5 million downloads till date. *Snort* releases till now were single threaded [4] but its new developmental release *Snort* v3 is multi-threaded with more enhanced features but is still in its beta stage.

Snort 2.X is a single-threaded user-level application which works on TCP/IP stack. It sniffs and examines all incoming packets in order to identify any malicious activity. It uses deep packet inspection (DPI) [5] for examining packets wherein it first inspects the packet header only, but in cases where this is not sufficient it goes on to examine the packet payload as well. It works on Windows, Linux and FreeBSD operating systems. *Snort*, a signature-based IPS, receives network packets and normalizes the contents so that a set of rules can be applied on it to detect the presence of any intrusion.

In this paper, an evaluation approach has been presented to measure the performance of *Snort* on different operating systems under different traffic conditions. The study compares NIDPS *Snort* v2.9.11 on Windows Server 2016, virtual Windows Server and *Ubuntu* Server 16.04 for large packet sizes and high-speed traffic. The effect of more CPU allocation is also discussed for *Snort* installed on *Ubuntu* 16.04 Server. A real network has been set-up to evaluate and compare the performance based on a series of tests. The paper is organized into six sections where Section I contains the introduction, Section II contain the related work done by other researchers to study and evaluate the performance of *Snort*, Section III contain the experiment plan, Section IV contains the observations and infers the results obtained and finally Section V concludes the research work.

II. Related Work

In [6], K. Salah and A. Kahtani have evaluated and compared the performance of *Snort* on Windows 2003 sever and Linux

platforms in terms of throughput and packet loss. In Windows, they studied the effect of processor scheduling parameter on *Snort's* performance by configuring it to allocate more CPU scheduling time to kernel networking subsystem and user processes. In Linux, different values have been set for parameter NAPI budget, to study its impact on performance of *Snort*. They concluded that under normal traffic conditions Windows Server outperformed Linux operating system in terms of throughput but for malicious traffic, Linux with a small NAPI budget value of 2 outperformed Windows as well as other Linux configurations. However, the change in processor scheduling had negligible effect on *Snort* Performance for Windows.

Salah et al. in [7] have compared *Snort*'s performance on Windows2008 Server and Windows7 to help choose the best configuration for *Snort*. They evaluated *Snort*'s performance under UP (uni-processing) and SMP (Symmetric multiprocessing) environments and studied the effect of processor affinity on it. Under UP environment, Windows2008 gave better throughput as compared to Windows7 for low traffic rates but at higher rates, both of them performed poorly due to lack of CPU availability. *Snort* under SMP environment showed significantly better throughput than UP environment and the throughput further increased with static affinity rather than default dynamic affinity. They have not studied the effect of large packet size and high speed traffic on the performance of *Snort* for Windows Server.

The effect of varying packet size and rule set size on *Snort*'s performance has been studied in [8] and different methods have been suggested to reduce the dependency of *Snort*'s performance on increase in rule-sets. She proposed the use of sparse banded matrix data structure for implementing pattern-matching in Aho-Corasick algorithm of detection engine to increase both memory as well as time efficiency of the system. The reduction in memory requirements was intended to avoid cache misses and to make room for ever-growing number of rule-sets, which in turn would increase *Snort*'s performance. However, this resulted in a more complex implementation of the data structure which required more time for starting *Snort*.

In [9], *Snort* has been tested for high speed and heavy traffic and different architectures have been proposed to deal with the problem of dropped packets and increase performance. They showed that as the traffic increases, packet drop rate also increases. Also, for large packets the packet drop rate is high. They have tested on core i4, i5 and i7 and Windows7 and Windows Server operating systems. Their tests have focused on Windows OS only whereas *Snort* provides various enhanced features when deployed in Linux OS like barnyard which give better performance.

Waleed Bul'ajoul et al in [10] [11] have established that when *Snort* runs in parallel in a multi-processor environment, its packet drop rate decreases. In their tests, they showed that when *Snort* was exposed to heavy and high-speed traffic, the number of packets analysed by it reduces and its packet drop rate increases. Similar results were obtained when large sized packets were run through it. However, when more than one *Snort* was run in a multi-processor environment with heavy and high speed traffic, the packet drop rate was drastically reduced. They have tested the performance only on Windows OS and the traffic rate at which they have performed the tests is very low.

In [12] five different test scenarios have been deployed to study *Snort* performance. Testing was done with different number of packets, different packet-sizes, traffic rates and combination of above. They observed an increase in the number of dropped packets with increase in number of packets to be processed, increase in packet-sizes and traffic rate. The network speed was less as compared to real network.

In [13], *Snort* has been tested in both host configuration as well as Virtual configurations for different hardware implementations and operating system by loading the systems with large packet-sizes and bandwidth. *Snort* showed lower detection ability and dropped packets in almost all the scenarios. Virtual *Snort* showed poor performance amongst all and so it was recommended not to use *Snort* in Virtual configuration. It was concluded that *Snort* performance gets degraded for high volume of traffic above 750 Mbps in all the network implementations.

In [14] *Snort* is compared with Suricata on the basis of scalability and performance. They performed a total of 8600 tests by varying the number of cores used (1 to 24 cores), the rule-sets used for signature comparison, the workload used to obtain results and the configuration of both the IDSs. The metrics used for comparison were packets per second (pps) as processed by each IDS, the amount of memory used by each IDS process and the CPU utilization. Results showed that both *Snort* and *Suricata* were scalable but *Suricata* outperformed *Snort* in almost all the test scenarios. *Suricata* also exhibited lower average memory usage and lower average CPU utilization.

Detection accuracy of three popular open-source intrusion detection systems- *Snort*, Suricata and Bro-IDS has been compared and analyzed in [15]. They studied the effect of number of active rules, different traffic rates and eight types of attacks on the evaluation efficiency of the Intrusion Detection Systems and concluded that use of different set of rules (active rules) for different attack types resulted in increased accuracy of the IDS. Also, Bro-IDS showed better performance amongst other IDS systems when evaluated under different attack types and using a specific set of active rules.

Snort's performance is also evaluated for detecting DoS and Port scan attacks in network [16] Snort has been evaluated in a high-speed network to determine its efficiency in detecting network attacks. The performance of *Snort* on Ubuntu server is good as it gives 100 % detection rate with zero false alarms in most of the attacks except Ping of Death.

In [6], [7], [9] and [13], snort has been tested on tested and compared on windows and Linux operating systems but the versions on which it is tested are very old. Also with time, new Snort versions are available with new and modified rule-sets as per the current threats. This work tests the performance of Snort on latest Windows and Linux operating systems along with the latest Snort rule-set to correctly identify the status of Snort in handling current traffic trends. Another motivation behind this work is to test Snort's performance on cloud platform which is extensively used these days. None of the previous works have tested Snort's performance on cloud platform.

In [9], [10] and [13], the effect of heavy traffic and large sized packets on performance of Snort has been tested but a higher traffic rate and large packet size are considered in this paper.

Our work is different from all the previous work for various reasons: (1) *Snort*'s performance of version v 2.9.11 has not been analyzed and compared on *Ubuntu* 16.04 Server and Virtual Windows 2016 Server. (2) The traffic rate at which experiments are carried out is sufficiently high (3) Studying the impact of different proportions of malicious traffic on *Snort*'s performance has not been considered before. (4) The effect of more CPU allotment on *Snort*'s performance in *Ubuntu* 16.04 Server has not been estimated.

III. Experiment Plan

We aim to evaluate *Snort* in network intrusion detection mode by analyzing its performance under high-speed and heavy load conditions for different operating systems. The effect of more CPU allocation on the performance of *Snort* on *Ubuntu* Server [17] has also been studied. *Snort v2.9.11* is installed in its default configuration with 9453 rules provided by *Snort* Vulnerability Research Team (VRT).

D-ITG used to generate both normal and malicious traffic. *D-ITG* (Distributed Internet Traffic Generator) generates IPv4 packets [18] at application, transport and network layer. It can generate a packet rate of 75000 packets per second where size of each packet is 1024 bytes [19].

The study is divided into three tests:

1. ***Test 1*****:** *Snort* is tested on *Ubuntu* 16.04 Server, Windows Server 2016 and Virtual Windows2016 Server for large sized packets of 512, 1024, 1536, 2048, 2560 and 3072 bytes for four combinations of normal and malicious traffic.
2. ***Test 2*****:** Snort is tested on Ubuntu 16.04 Server, Windows Server 2016 and Virtual Windows2016 Server for high speed traffic of 5000, 10000, 15000, 20000, 25000 and 30000 packets per second. The four traffic cases described above have also been considered here.
3. ***Test 3*****:** The effect of allocation of more CPU time is studied on the performance of *Snort* on *Ubuntu* 16.04 Server.

Test 1and 2 are further divided into four cases to study the effect of four different proportions of normal and malicious traffic. Every malicious packet triggers an alert from *Snort* whereas normal packets do not generate any alert. The four traffic combinations are:

- Normal traffic
- TCP and ICMP packets are normal but all UDP packets are malicious
- UDP and ICMP packets are malicious but TCP packets are normal
- All TCP, UDP, ICMP packets are malicious.

Three Performance metrics have been used to calculate and compare *Snort* performance in different test scenarios. These metrics are based on the parameters that impact *SNORT* performance. The evaluation parameters are:

- **Packet drop (%):** It is computed as

$$\textbf{Packet drop (\%)} = \frac{\textit{Total packets dropped by Snort}}{\textit{Total packets received by Snort}} * 100.$$

- ***Snort* efficiency (%):** It is the percentage of packets analysed by *Snort* and is calculated as:

$$\textbf{\textit{Snort} efficiency} = \frac{\textit{Total packets analysed by Snort}}{\textit{Total packets received by Snort}} * 100.$$

- **CPU utilization (%)**

3.1. EXPERIMENTAL SET-UP

Three separate experimental test benches have been setup to test the performance of *Snort v2.9.11* on different platforms in similar network conditions. The system description and specifications are enlisted in the table 1 below.

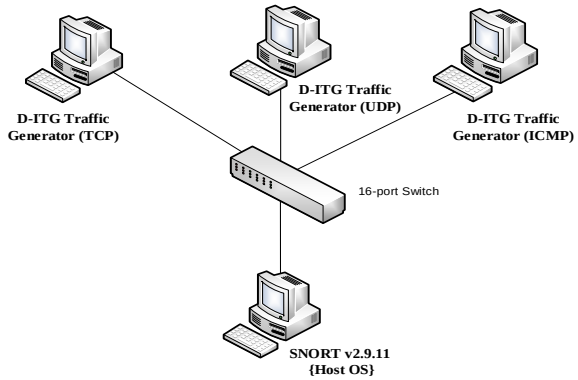

Figure 1. Experimental set-up

Machine	Description	Specifications
TCP traffic generator	Dell Intel(R) core(TM) i3-3110M CPU @ 2.40GHz, 8 GB RAM	*Ubuntu* 16.04 desktop with *D-ITG* traffic generator
UDP traffic generator	Dell Intel(R) core(TM) i3-3110M CPU @ 2.40GHz, 8 GB RAM	*Ubuntu* 16.04 desktop with *D-ITG* traffic generator
ICMP traffic generator	Dell Intel(R) core(TM) i3-3110M CPU @ 2.40GHz, 8 GB RAM	*Ubuntu* 16.04 desktop with *D-ITG* traffic generator
Snort v2.9.11	Hp Intel(R) core(TM) i5-3210M CPU @ 2.40GHz, 8 GB RAM	Windows Server 2016
Snort v2.9.11	Hp Intel(R) core(TM) i5-3210M CPU @ 2.40GHz, 8 GB RAM	*Ubuntu* 16.04 Server

Table 1. System Specifications

Setup 1: The test bench consists of four computers forming a LAN via a D-link web smart DGS-1210-16 16-port switch as shown in Figure 1. Three systems have been used as traffic generators and have *D-ITG* installed on them, where each machine is generating packets pertaining to one protocol. Machine 1 generates TCP packets, machine 2 generates UDP packets and machine 3 generates ICMP traffic. *Snort* is installed on the fourth machine with *Ubuntu* 16.04 Server as Host OS.

Setup 2: The test bench is similar to the test bench in Setup 1 except that the Host OS is Windows Server 2016 on which *Snort* is installed.

Setup 3: The Virtual Windows Server is set-up on a commercialized cloud (Azure) with three traffic generators and a switch. *Snort* is installed on Virtual Windows 2016 Server. All the machines are Standard_D2s_v3 with 2 VCPU and 8GB RAM.

IV. Results And Discussions

A. Test 1

For this experiment, TCP, UDP and ICMP packets of different sizes are sent at a rate of 15000 packets per second to all the three Servers. Total 1800000 packets of different sizes are send and number of packets analysed and dropped by *Snort* are recorded. Four test cases are considered here, by sending different amounts of malicious traffic to all the three Servers under evaluation.

1) Case 1

In this case only normal traffic was used for evaluation. The results of sending normal traffic is tabulated in table 2(a) for the three Servers and is graphically represented in Figures 2 and 3. Figures 2 shows that as the size of packet increases from 512 to 3072 bytes, the packet drop percentage increases. Also, the values of packet drop percentage is less than 1% for all the Servers but amongst them, *Ubuntu* Server shows better performance by dropping less number of packets. The CPU utilization increases with the increase in packet size and is more for *Ubuntu* 16.04 Server (Figure 3). Snort efficiency is similar for all the three operating systems in this case.

(a) Normal traffic

	***Ubuntu* Server**			**Virtual Windows Server**			**Windows Server**		
Packet size	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**
512	0.0065	99.99	22.07	0	100	20	0	100	22
1024	0.0021	100	25.935	0.009	99.99	25.5	0	100	27
1536	0.0038	100	39.93	0.016	99.98	41.36	0.016	99.98	44
2048	0.004	100	58.88	0.019	99.98	43	0.015	99.99	46
2560	0.004	100	68.68	0.021	99.98	47	0.019	99.98	48
3072	0.0043	100	76	0.024	99.98	54	0.133	99.87	55

(b) UDP **malicious traffic**

	***Ubuntu* Server**			**Virtual Windows Server**			**Windows Server**		
Packet size	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**
512	0.405	99.6	30.28	45.95	54.05	64	0.092	99.91	28
1024	1.165	98.84	38.46	46.065	53.94	69	2.563	97.44	38
1536	1.122	98.88	52.14	47.212	52.79	69	2.867	97.13	56
2048	1.542	98.46	67.08	47.439	52.56	69	3.881	96.12	57
2560	1.38	98.62	76.74	47.2	52.8	76	5.038	94.96	58
3072	2.004	98	84.85	49.59	50.41	76	6.886	93.11	65

(c) UDP **and** ICMP **malicious traffic**

	***Ubuntu* Server**			**Virtual Windows Server**			**Windows Server**		
Packet size	Packet Drop (%)	Snort efficiency (%)	CPU Utilization (%)	Packet Drop (%)	Snort efficiency (%)	CPU Utilization (%)	Packet Drop (%)	Snort efficiency (%)	CPU Utilization (%)
512	2.7727	97.23	34.76	48.277	51.72	67	2.629	97.37	35
1024	2.1596	97.84	43.02	48.4	51.6	75	5.779	94.22	52
1536	2.7505	97.25	60.4	48.66	51.34	75	9.963	90.04	67
2048	5.64	94.36	71.94	48.7	51.3	76	11.365	88.64	69
2560	11.24	88.76	81.56	48.46	51.54	76	19.6	80.4	70
3072	12.02	87.98	90.95	50.42	49.58	79	26.9	73.1	73

(d) All malicious traffic

	***Ubuntu* Server**			**Virtual Windows Server**			**Windows Server**		
Packet size	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort efficiency (%)**	**CPU Utilization (%)**
512	3.7406	96.26	37.4	48.756	51.24	67.8	3.319	96.68	39
1024	2.9298	97.07	44.85	48.87	51.13	76	3.974	96.03	77
1536	4.6754	95.32	57.12	49.142	50.86	76	6.616	93.38	81
2048	4.5604	95.44	78.01	49.156	50.84	78	16.8	83.2	86
2560	11.077	88.92	91.52	49.17	50.83	79	17.672	82.33	91
3072	20.671	79.33	93.25	53.566	46.43	83	18.864	81.14	96

Table 2. Comparison of three Servers for different packet sizes and different proportions of malicious traffic

2) Case 2

For this case, UDP traffic from UDP traffic generator was malicious and both TCP and ICMP traffic was normal. The results are tabulated in table 2(b) and represented in Figures 4 and 5. Figure 4 shows that the packet drop percentage increases a bit from size 512 to 1024 and is similar for sizes 1024, 1536, 2048 and 2560 bytes and then shows a slight increase for packet size of 3072 bytes for all the three Servers. *Ubuntu* Server shows better performance by dropping less packets as compared to Windows Server and Virtual Windows Server. Virtual Windows Server drops almost 48% of incoming traffic and so almost half of the incoming packets go unchecked. CPU utilization increases with the increase in packet size and is maximum for Virtual Windows Server. Snort efficiency decreases with increase in packet size and is lowest for Virtual Windows Server.

3) Case 3

In this case, both UDP and ICMP packets are malicious and TCP traffic was normal. The results are represented in Figures 6 and 7 and are tabulated in table 2(c). Initially, the size of packet has very little effect on packet drop percentage (512 to 1536bytes) for the three Servers but when size becomes large, the drop percentage follows a ramp for *Ubuntu* and Windows Server. Virtual Windows Server performs poorly and drops almost half of the packets for all packet sizes, however, *Ubuntu* Server shows better performance as compared to Windows Server (Figure 6). Snort efficiency decreases with increase in packet size and is lowest for Virtual Windows Server. The CPU utilization increases with the increase in packet size and is maximum for Virtual Windows Server for all packet sizes except 3072 bytes.

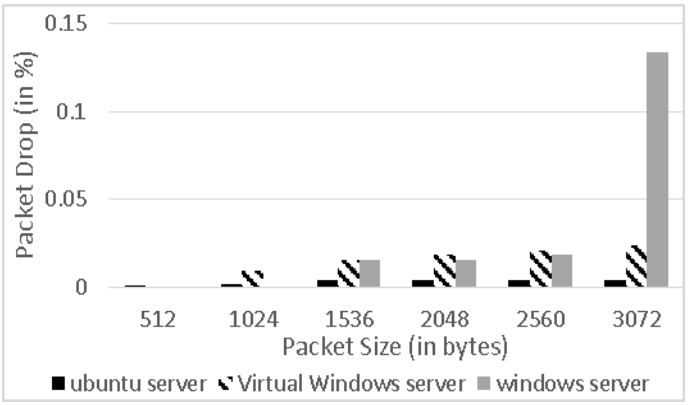

Figure 2. Packet drop percentage for normal traffic

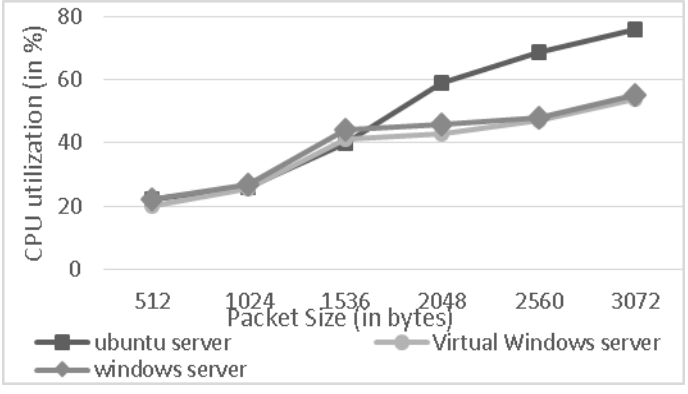

Figure 3. CPU utilization for normal traffic

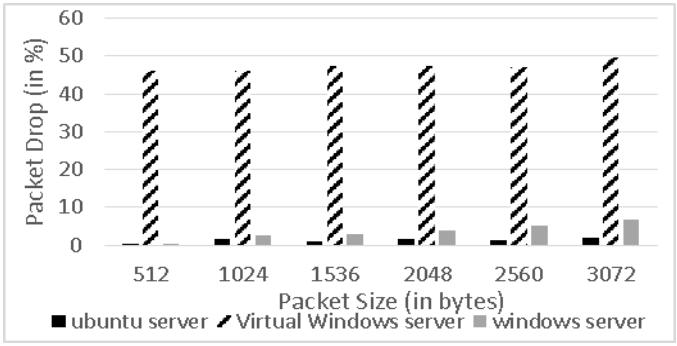

Figure 4. Packet drop percentage for malicious UDP traffic

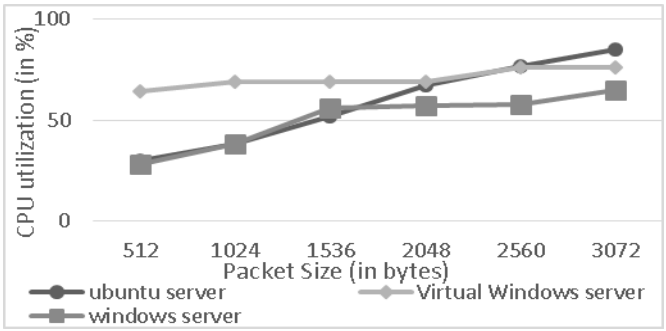

Figure 5. CPU utilization for malicious UDP traffic

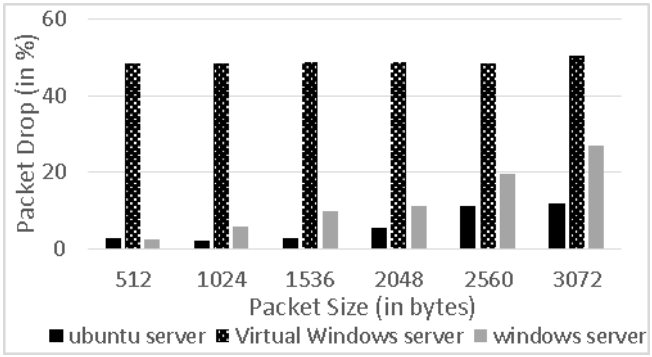

Figure 6. Packet drop percentage for malicious UDP and ICMP traffic

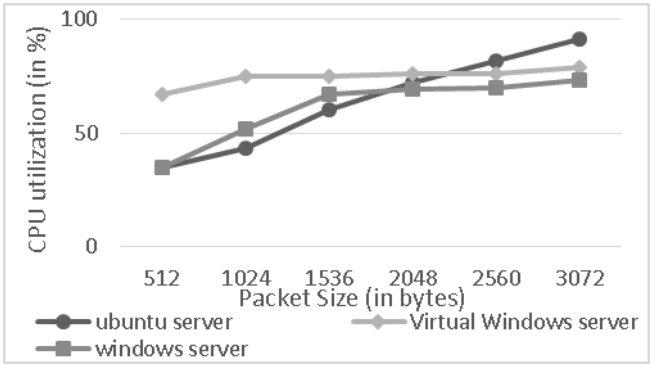

Figure 7. CPU utilization for malicious UDP and ICMP traffic

4) Case 4

The traffic consists of malicious packets of TCP, UDP and ICMP as shown in table 2(d). The results show that small sized packet have very little effect on packet drop percentage for the three Servers but when packet size becomes large (2560 and 3072 bytes), the drop percentage follows a linear increase for Windows and *Ubuntu* Servers. *Ubuntu* Server shows better performance and Virtual Windows Servers

performs poorly (Figure 8). CPU utilization increases with increase in packet size and is maximum for Windows Server (Figure 9). Snort efficiency decreases with increase in packet size and is lowest for Virtual Windows Server.

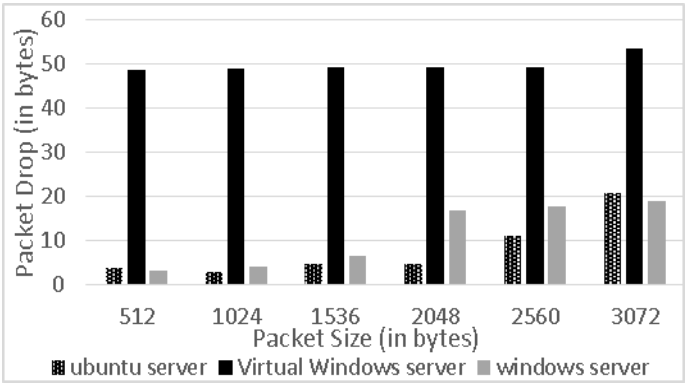

Figure 8. Packet drop percentage for all malicious traffic

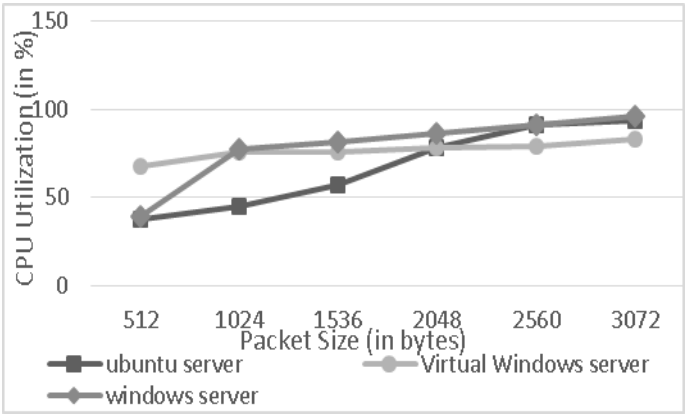

Figure 9. CPU utilization for all malicious traffic

The performance of *Snort* is affected by the size of incoming packets for all the three Servers. The packet drop increases slightly as we go from 512 to 1024 bytes packets because as the payload size increases, the amount of time taken by *Snort* to process the contents for matching it against the rule set also increases. As we go from 1024 bytes to 1536 bytes, the packets drop percentage shows a raise. The reason for this behavior is that a packet of size 1536 bytes undergoes fragmentation before reaching *Snort* as its size is greater than one MTU (Maximum Transfer Unit). *Snort* takes more time in processing fragmented packets and so packet drop increases. Packets of size 1536 and 2048 bytes have similar values of packet drop but an increase is observed for 2560 bytes as for this size the amount of payload to be matched is high as compared to other two sizes. With the increase the packet size from 2560 to 3072 bytes, the packet drop percentage increases abruptly because for this size each packet has to be fragmented into three MTUs which need more processing from *Snort*.

On comparing *Snort's* performance on Windows and *Ubuntu* Server, it is found that *Snort* shows a low performance on Windows Server for malicious traffic. This is because of the sub-process of writing alerts on the disk. In *Ubuntu*, logs are written in *unified2* binary format which are then read by *barnyard2* [20] whereas no such option is available in Windows. Writing logs in unified2 binary format is the fastest mode of outputting alert data [21], which takes less time to write logs thereby increasing *Snort* processing efficiency and reducing packet drop rate.

Also, the performance of *Snort* in cloud environment is very poor as almost 50% of the packets are dropped. The prime reason behind the poor performance of *Snort* could be convincingly attributed to the fact that all Host Operating Systems in Cloud are Virtual in nature (i.e. Virtual Machines) and hence are resource constrained. Given this reason it could be extrapolated that the performance of *Snort* in cloud environment could be improved by making it run on a Virtual machine with generous amount of computing and memory resources

A high value of CPU utilization on a standalone host is desirable as it indicates better resource utilization but while working on a shared host, a higher value of CPU utilization means more waiting time for other processes, which is a problem. It is deduced from table 2 that *Snort* on Virtual Windows Server uses up most of the CPU processing time which is an undesirable feature as it increases delay for other processes that require processing on a shared host.

Also, with increase in the content of malicious packets, both the packet drop percentage and the CPU utilization increases. As malicious packets require more content matching time and for each packet an alert is to be written on the disk, so *Snort* takes more time in processing them thereby increasing CPU utilization and Packet drop percentage for all the three Servers under test.

B. Test2

Equal amounts of TCP, UDP and ICMP packets are send to *Ubuntu* 16.04 Server, Windows2016 Server and Virtual Windows Server 2016 at speeds of 5000, 10000, 15000, 20000, 25000 and 30000 packets per second for 120 seconds. All the packets are of size 256 bytes. Performance of *Snort* has been studied and compared for normal traffic and different proportions of malicious traffic.

1) Case 1

For normal traffic, all the Servers performed good with no packet loss for all the different speeds as shown in table 3 (a). The value of CPU utilization increased with the increase in speed of sending packets for the three Servers. For small traffic rate, Windows Server has high CPU utilization but for high traffic rates, Virtual Windows Server has higher value of CPU utilization (Figure 10).

2) Case 2

For UDP malicious traffic as shown in table 3(b), Virtual Windows Server dropped almost half of the packets and performed poorly in all the cases as depicted by Figures 12. *Ubuntu* performed better by dropping less packets. CPU utilization increased with the increase in speed of sending packets for both Windows and *Ubuntu* Servers (Figure 11), but for Virtual Windows Server it showed very less variation and was maximum amongst the three Servers.

3) Case3

As shown in table 3(c) when malicious ICMP and UDP traffic was send, Windows showed poor performance than *Ubuntu* by dropping more packets (Figure 13). Virtual Windows performed poorly again by dropping maximum number of

packets. CPU utilization is highest for Virtual Windows Server and minimum for *Ubuntu* Server (Figure 14).

(a) Normal traffic

	Ubuntu Server			Windows Server			Virtual Windows Server	
Packets received per second	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**
5000	0	100	12.9	0.036	99.96	27	0	100	16
10000	0	100	16.62	0.055	99.95	28	0	100	20
15000	0	100	21.26	0.052	99.95	28	0	100	30
20000	0	100	26.5	0.025	99.98	31	0	100	50
25000	0	100	39.4	0.067	99.93	32	0	100	53
30000	0	100	44.96	0.069	99.93	37	0.0157	100	59

(b) Malicious UDP **traffic**

	Ubuntu Server			Windows Server			Virtual Windows Server	
Packets received per second	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**
5000	0	100	21.33	0.235	99.77	28	39.72	60.28	69.5
10000	0.0987	99.9	25.8	0.9282	99.07	32	45.44	54.56	69.47
15000	0.356	99.64	39.9	0.9519	99.05	33	47.95	52.05	70.88
20000	0.655	99.35	48.06	0.969	99.03	35	48.23	51.77	72.36
25000	1.06	98.94	65.2	1.439	98.56	40	48.786	51.21	75.2
30000	1.876	98.12	74.8	1.54	98.46	42	48.892	51.11	80

(c) Malicious UDP **and** ICMP **traffic**

	Ubuntu Server			Windows Server			Virtual Windows Server	
Packets received per second	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**
5000	0	100	22.33	0.367	99.63	28	46.4	53.6	70.9
10000	0.0985	99.9	33.35	0.9491	99.05	31	47.67	52.33	72.24
15000	0.156	99.84	52.47	1.207	98.79	34	49.02	50.98	74.32
20000	1.02	98.98	71.35	5.078	94.92	37	49.14	50.86	79.65
25000	1.375	98.63	80.63	8.927	91.07	43	49.26	50.74	80.57
30000	3.52	96.48	93.94	8.6	91.4	52	49.4	50.6	81.21

(d) All Malicious traffic

	Ubuntu Server			Windows Server			Virtual Windows Server	
Packets received per second	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**	**Packet Drop (%)**	**Snort Efficiency (%)**	**CPU Utilization (%)**
5000	0.13468	99.87	25	0.0526	99.95	36	46.383	53.62	69
10000	0.14088	99.86	35.11	1.053	98.95	37	48.108	51.89	74
15000	0.36327	99.64	56	0.949	99.05	40	48.886	51.11	74
20000	1.35879	98.64	79.15	1.58	98.42	42	49.257	50.74	83
25000	4.695	95.31	93.2	7.018	92.98	45	49.35	50.65	85.51
30000	9.9978	90	96	19.403	80.6	55	49.49	50.51	85.21

Table 3. Comparison of the three Servers for high-speed traffic and different proportions of malicious data

4) Case 4

For malicious traffic as shown in table 3(d), Windows Server continued to perform poorly. However, the dropped packets increased abruptly for *Ubuntu* Server and Windows Server at the speed of 30000 packets per second as in Figure 15. CPU utilization is high for Virtual Windows Server initially but for higher traffic rates of 25000 and 30000 packets per second, *Ubuntu* Server takes over (Figure 16).

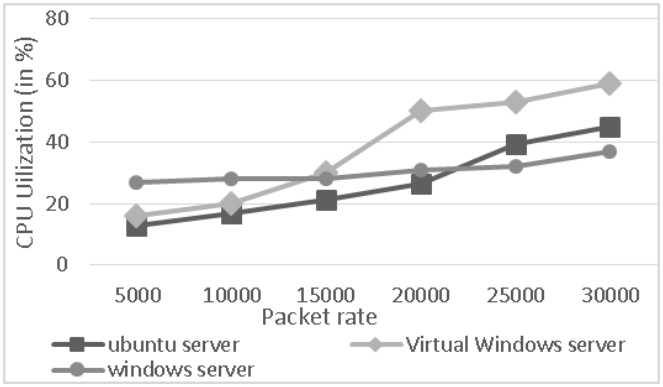

Figure 10. CPU utilization for high-speed normal traffic

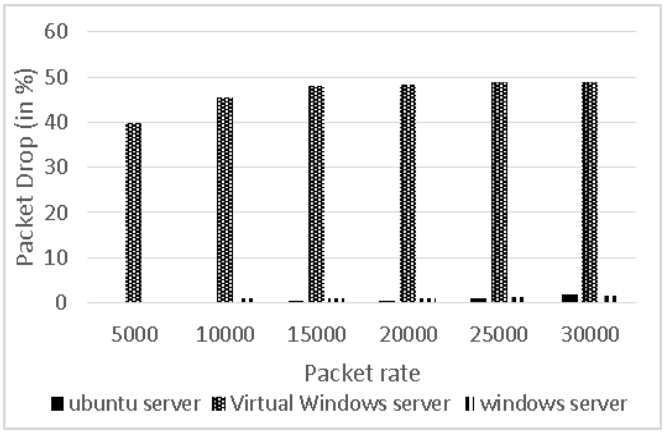

Figure 11. Packet drop percentage for UDP malicious traffic

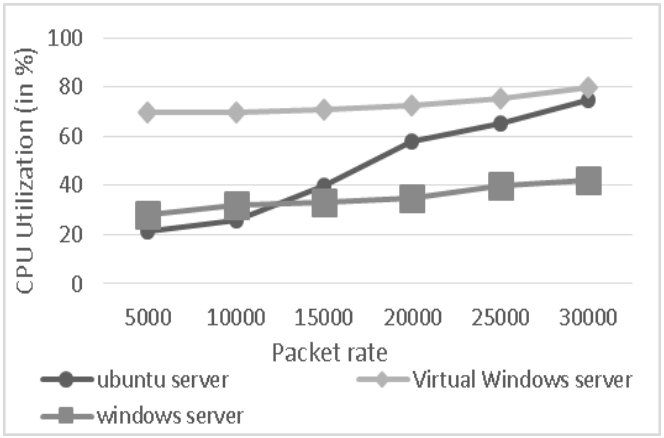

Figure 12. CPU utilization for malicious UDP traffic

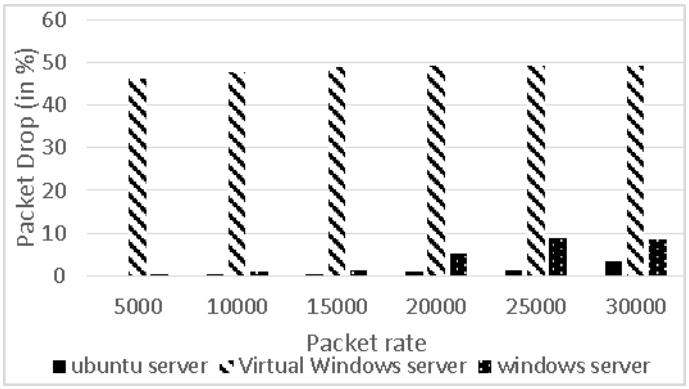

Figure 13. Packet drop percentage for malicious UDP and ICMP traffic

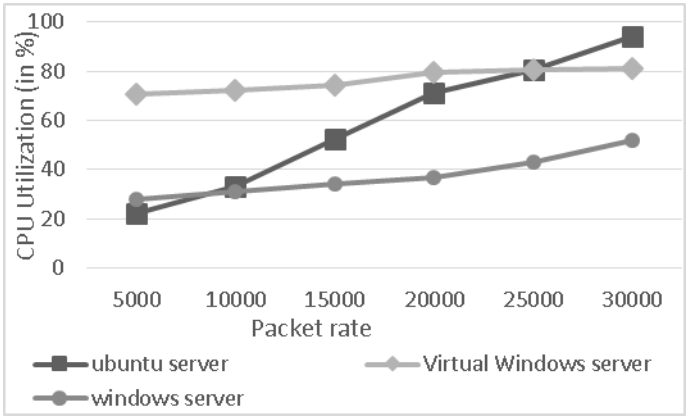

Figure 14. CPU utilization for malicious UDP and ICMP traffic

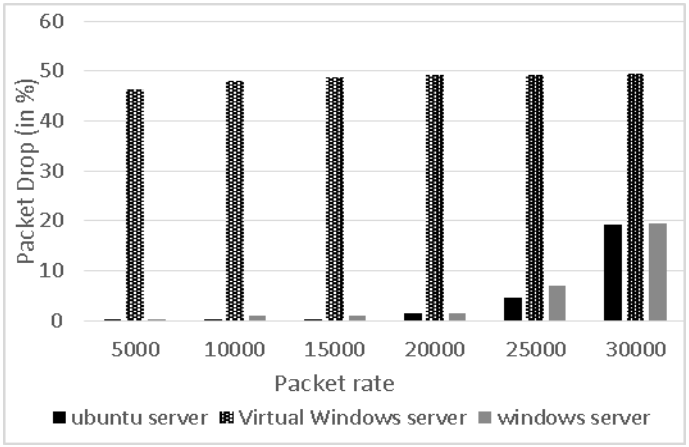

Figure 15. Packet drop percentage for high-speed malicious traffic

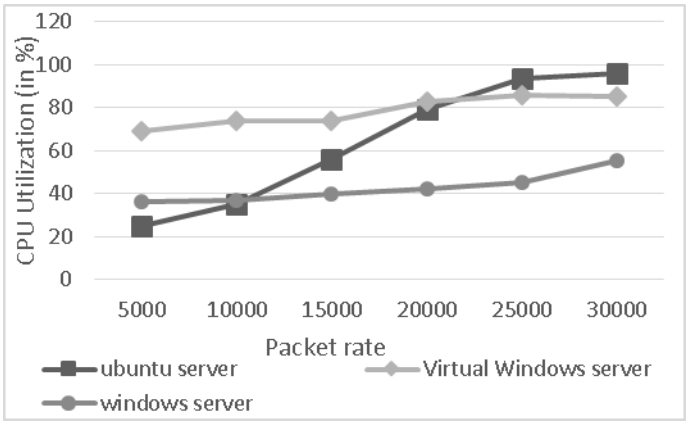

Figure 16. CPU utilization for high-speed malicious traffic

Packet drop percentage is linearly dependent on the traffic rate of incoming packets for all the three Servers. It is found that for Windows and *Ubuntu* Server, the drop increases after the speed of 20000 packets per second. Also, with the increase in the amount of malicious traffic, both the packet drop rate and CPU utilization increases.

In cases 2, 3 and 4, Snort efficiency decreases with increase in speed of traffic because as traffic rate increases, the load on Snort detection engine increases thereby resulting in dropping of some packets. However, the decrease in efficiency is low. But for Virtual Windows server the efficiency is almost half due to lack of resource availability making it least favorable platform for IDS deployment.

C. Test 3

The aim of this test is to find the effect of amount of CPU allocated to *Snort* on its performance, for *Ubuntu* 16.04 Server. All the previous studies related to the performance of Snort have not considered the effect of change in the amount of CPU allocation to Snort process, so this test provides an optimized way of running Snort to get improved performance.

The default processor time allocated to the Snort process by Ubuntu server is changed by changing the priority of Snort process. The nice value of *Snort* process on *Ubuntu* 16.04 Server is changed to -20 (highest priority as kernel-based processes) which schedules more CPU time to it as compared to the default CPU allotment. Malicious traffic at 5000, 10000, 15000 and 20000 packets per second (equal number of TCP, UDP and ICMP packets) are send to *Snort* for both the cases and results are recorded in table 4. It is found that when more CPU was allotted to *Snort* process, the packet drop percentage decreased as shown in Figure 17.

As *Snort* is a single threaded application, so on a dedicated standalone host, it gives better performance when it is run with more CPU resource at its disposal rather than in default CPU allocation.

Packets received per second	Packet Drop (%) with more CPU allocation	Packet Drop (%) with default CPU allocation
5000	0	0.135
10000	0.095101159	0.141
15000	0.290918429	0.363
20000	0.887286586	1.358

Table 4. Effect of more CPU allocation on performance of *Snort*

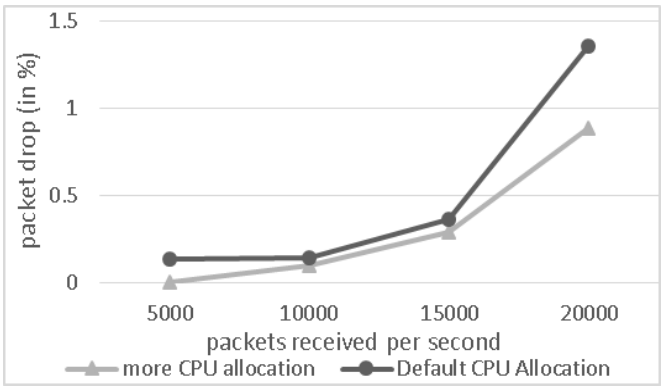

Figure 17. Packet drop percentage for *Snort* with different CPU allocations

V. Conclusion

This study focusses on determining the performance of *Snort v2.9.11* on Windows 2016 Server, *Ubuntu* 16.04 Server and Virtual Windows 2016 Server. A series of tests were conducted and it was found that with the increase in the size of packet, the number of dropped packets increase on *Ubuntu* 16.04 Server, Windows Server 2016 and Virtual Windows Server. Similar results were obtained for high-speed traffic which also showed an increased tendency to drop packets with increase in traffic speed. The work under study also determined that *Snort*'s performance on Virtual Windows 2016 Server, under high content of incoming malicious traffic, doesn't meet the expectations as demanded by the security administrators. Amongst the three, *Snort* performs best on *Ubuntu* 16.04 Server.

Also, the packet drop percentage and CPU utilization increases with the increase in the amount of malicious packets in the incoming traffic. Further, by allotting more CPU time to *Snort* process on *Ubuntu* Server, it was found that with more CPU, the packet drop percentage was reduced. Results show many limitations of *Snort* in handling large packet size of 3072 bytes or more and high-speed traffic of 25000 pps or more. The results establish the incapacity of Snort to cope up with large packet sizes and high-speed traffic and its tendency to drop packets.

References

[1] P. Innella, "An Introduction to IDS," 5 dec 2011. [Online]. Available: https://www.symantec.com/connect/articles/introduction-ids. [Accessed Dec 2017].

[2] S. Chakrabarti, M. Chakraborty and I. Mukhopadhyay, "Study of *Snort*-based IDS," in *Proceedings of the International Conference and Workshop on Emerging Trends in Technology, ACM*, 2010.

[3] M. Roesch, "*Snort*-Lightweight Intrusion detection for Networks," in *13th Systems Administration Conference (LISA)*, Seattle, Washington, USA,, 1999.

[4] M. Roesch, "*SNORT* 3 User manual," SourceFire.Inc, 2017. [Online]. Available: https://*Snort*-org-site.s3.amazonaws.com/production/release_files/files/000/005/901/original/*Snort*_manual.pdf?X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=AKIAIXACIED2SPMSC7GA%2F20180115%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-Date=20180115T055124Z&X-Am. [Accessed Nov 2017].

[5] S. Hafeez, "Deep Packet Inspection using *Snort*," 2016.

[6] K.Salah and A.Kahtani, "Performance evaluation comparison of *Snort* NIDS under Linux and Windows Server," *Journal of Network and Computer Applications*, vol. 33, no. 1, pp. 6-15, January 2010.

[7] K. Salah, M.-A.-R. Al-Khiaty, R. Ahmed and A. Mahdi, "Performance Evaluation of *Snort* under Windows7 and Windows Server 2008," *Journal of*

Universal Computer Science, vol. 17, no. 11, pp. 1605-1622, 2011.

[8] S. Sen, "Performance Characterization & Improvement of *Snort* as an IDS," Bell Labs, 2006.

[9] I. Karim, Q.-T. Vien, T. A. Le and G. Mapp, "A Comparative Experimental Design and Performance Analysis of *Snort* Based Intrusion Detection Systems Detection System in Practical Computer Networks," *Computers*, vol. 6, no. 1, 7 feb 2017.

[10] W. Bul'ajoul, A. james and m. pannu, "Improving Network Intrusion Detection System Performance through Quality of Service Configuration and Parallel Technology," *Journal of Computer and System Sciences, ACM*, vol. 81, no. 6, pp. 981-999, september 2015.

[11] W. Bul'ajoul, A. James and M. Pannu, "Network intrusion detection systems in high-speed traffic in computer networks," in *IEEE 10th International Conference on e-Business Engineering*, 2013.

[12] M. Saber, M. G. Belkasmi, S. Chadli, M. Emharraf and I. E. Farissi, "Implementation and Performance Evaluation of Intrusion Detection Systems under high-speed networks," in *Proceedings of the 2nd international Conference on Big Data, Cloud and Applications*, 2017.

[13] M. Akhlaq, F. Alserhani, I. Awan, J. meller, A. J. Cullen and A. Al-Dhelaan, "Implementation and Evaluation of Network Intrusion Detection Systems," in *Network Performance Engineering*, vol. 5233, Springer, Berlin, Heidelberg, 2011, pp. 988-1016.

[14] J. S. White, T. Fitzsimmons and J. N. Matthews, "Quantitative analysis of intrusion detection systems: *Snort* and Suricata," in *Proceedings of the SPIE Defence Security and sensing*, May 2013.

[15] K. Thongkanchorn, S. Ngamsuriyaroj and V. Visoottiviseth, "Evaluation Studies of Three Intrusion Detection Systems under Various Attacks and Rule Sets," in *IEEE*, 2013.

[16] A. Gupta and L. S. Sharma, "Mitigation of DoS and Port Scan Attacks Using Snort," *International Journal of Computer Sciences and Engineering*, vol.7, no. 4, April 2019

[17] D. P. Bovet and M. Cesati, "Understanding the Linux kernel", 3. Edition, O'Rilley Press, 2005.

[18] D. Emma, A. Pescape and G. Ventre, "*Analysis and experimentation of an open distributed platform for synthetic traffic generation"*, Suzhou, 2004, pp. 277-283.

[19] S. Avallone, S. Guadagno, D. Emma, A. Pescap and G. Ventre, "*D-ITG* Distributed Internet Traffic GeneratorS. Avallone S. Guadagno D. Emma A. Pescap `eG. Ventre," in *1st International Conference on Quantitative Evaluation of Systems* , Enschede, The Netherlands, 27-30 September 2004.

[20] J. Turnbull, "Improving *Snort* performance with barnyard," Techtarget, 21 may 2007. [Online]. Available: https://searchdatacenter.techtarget.com/tip/Improving-*Snort*-performance-with-Barnyard.

[21] A. S. Singh and M. B. Masuku, "Applications of Modeling and Statistical Regression Techniques in Research," *Research Journal of Mathematical and Statistical Sciences* , vol. 1, no. 6, pp. 14-20, July 2013.

Author Biographies

Dr. Lalit Sen Sharma has obtained Master of Science in Mathematics and MCA from Guru Nanak Dev University, Amritsar (India). He has also obtained Doctorate of Philosophy (PhD) from Guru Nanak Dev University in 2008. Currently, he is working as a Professor and Head of Department in the department of Computer Science and Information Technology in University of Jammu, India. He has been teaching to postgraduate students of computer applications for fifteen years. He is a member of Indian Science Congress Association, Institute of Electronics and Communication Engineer, India and National HRD network, India.

Alka Gupta has obtained her B.E. in Computer Science from University of Jammu, India and has received her M. Tech in Computer Science from Shri Mata Vaishno Devi University Katra, J&K, India she has been pursuing her Doctorate of Philosophy (PhD) from Department of Computer Science and IT, University of Jammu since 2016. Her areas of interest include Computer Networks, Network Security, Data structures and mobile computing.