USING CISCO NETWORK COMPONENTS TO IMPROVE NIDPS PERFORMANCE

Waleed Bul'ajoul[1], Anne James[1], Siraj Shaikh[1] and Mandeep Pannu[2]

[1]Faculty of Engineering and Computing, Coventry University, Coventry, UK
bulajouw@coventry.ac.uk
a.james@coventry.ac.uk
aa8135@coventry.ac.uk
[2]Department of Computer Science, Kwantlen Polytechnic University, Surrey, British Columbia, Canada
mandeep.pannu@kpu.ca

ABSTRACT

Network Intrusion Detection and Prevention Systems (NIDPSs) are used to detect, prevent and report evidence of attacks and malicious traffic. Our paper presents a study where we used open source NIDPS software. We show that NIDPS detection performance can be weak in the face of high-speed and high-load traffic in terms of missed alerts and missed logs. To counteract this problem, we have proposed and evaluated a solution that utilizes QoS, queues and parallel technologies in a multi-layer Cisco Catalyst Switch to increase NIDPSs detection performance. Our approach designs a novel QoS architecture to organise and improve throughput-forward-plan traffic in a layer 3 switch in order to improve NIDPS performance.

KEYWORDS

Network security, intrusion detection system, network intrusion detection system, open source, Cisco switch configuration & Quality of Service.

1. INTRODUCTION

Despite the existence of a variety of security protection measures, attackers often attempt to render services unavailable to the intended, legitimate users [1, 2, 3, 4]. In general, there are three types of network security techniques: prevention, detection and correction techniques. The prevention technique actively works to block intrusions, but it can also be used to battle a successful intrusion. A number of successful attacks can be controlled using the prevention technique if an attack is detected at the interim stage of prevention systems. Unfortunately, some successful attacks can get through the prevention system [2, 5]. In this instance, depending on preventive techniques is unlikely to resolve the issue, especially when an attacker has successfully obtained vulnerable information from the network; however, prevention can successfully and effectively maintain a network before an attack is launched. The correction technique is adopted to protect computer systems. It is used when the prevention technique has failed. In these cases, the system is attacked and compromised; consequently, it malfunctions. The correction technique restores the system to a stable state when an attack has been detected. Clearly, both prevention and correction require a detection phase, which should be constantly active to combat intrusions.

Network intrusion detection and prevention systems (NIDPS) are commonly used to detect and prevent attacks. A popular system is the open-source Snort NIDPS. Our previous studies [4, 5] have been carried out on the use of a Snort NIDPS in high speed networks. We found that in high speed and high volume environments, the Snort NIDPS drops packets. Our studies focused on improving the performance of the NIDPS in analysis mode. Improving the analysis phase for any security production is important, because it is difficult to detect or prevent threats or malicious traffic without analysing the traffic. This paper however focuses on NIDPS detection rather than analysis. Thus we have now considered rule-based actions as well as the passive analysis of the packets received. Our paper uses Snort NIDPS. We conducted experiments to test Snort's detection-mode performance reaction to ICMP, UDP and TCP headers and malicious packets under high-load and high-speed traffic. We further demonstrate that Snort's performance can be improved by using additional technologies such as a Quality of Service (QoS) configuration and parallel technologies.

The remainder of this paper is organised as follows. Section 2 gives an overview of previous related work. Section 3 provides a background on: intrusion detection and prevention systems (IDPSs); network intrusion detection and prevention systems (NIDPSs); and information about Snort NIDPS. Section 4 explains our experimental design. Section 5 presents a first set of experiments which demonstrated some NIDPS weaknesses. Section 6 presents our solution for combatting such weaknesses and also provides an experimental evaluation of our solution. This is followed by section 7 which provides more information about the technical aspects of the approach. Finally Section 8 concludes the paper and suggests further work.

2. RELATED WORK

Chen, et al. [17] proposed an application-specific integrated circuit (ASIC) design with parallel exact matching (PEM) architecture to accelerate throughput. The ASIC hardware has been designed to operate at 435MHz to perform up to 13.9 Gbps throughput. The aim is to manage the requirements of high speed and high accuracy for Snort IDS (Intrusion Detection System) and overcome the complexity of managing data received from the 10Gbps core network. They proposed the SRA (Snort Rule Accelerator) which processes rules in parallel to increase the performance of the Snort IDS. The SRA has a stateless parallel-matching scheme to perform high throughput packet filtering as an accelerator of the Snort detection engine. The ASIC is combined of five major modules, including the inspector, counter, parallel matching, conformity and compare modules. The functionality of the parallel matching scheme is to compare payloads of packets with the stored rules. When an entry packet is matched with Snort rules, the ASIC is in an idle state and sends a compare and signal to the conformity module, which integrates all signals and determines whether an abnormal payload is presented. Here the authors designed half mesh architecture in the parallel matching rules module, which allows the traffic to be compared with several rules at the same time. Our work addresses performance in a different way. Instead of processing rules in parallel, it processes traffic in parallel. It explores the use of hardware Layer-3 network switches and Cisco configuration with parallel queue technologies to improve QoS and, hence, NIDPS performance.

Jiang et al. [18] proposed a new NIDS architecture based on multi-parallel core processors. They exploited many-core computational power by adopting a hybrid parallel architecture combining data and pipeline parallelism. They designed a system for parallel network traffic processing by implementing an NIDS on the TILERAGX36 (a 36 core processor) [19]. The system was designed according to two strategies: first a hybrid parallel architecture was used, combining data and pipeline parallelism; and secondly a hybrid load-balancing scheme was used. They took advantage of the parallelism offered by combining data, pipeline parallelism and multiple cores, using both rule-set and flow space partitioning. They showed that processing speeds can handle and reach up to 7.2Gbit/S with 100-byte packets. Our approach differs from theirs in that we have shown how can exploit QoS and queues technologies in a multi-layer switch to improve packets processing throughput. Further enhancements can occur if we combine queuing together with parallel technologies. Our approach requires less specialised equipment.

In previous work [4], we presented experiments that demonstrated that Snort dropped packets when it was deployed in high speed traffic in analysis mode. Packets were dropped or left outstanding when the in-coming speed of traffic is higher than NIDPS processing speed limit. We used QoS technology in a Cisco Catalyst switch to improve NIDPS performance and proposed a solution which reduces the packet arrival speed to the NIDPS node processor limit. This approach reduces dropped packets or the number of packets left outstanding by using queues to reduce the arrival rate to a speed that can be processed successfully. We then used parallel technology to increase the throughput rate.

This paper presents similar technology to that applied in our previous work [4] but this time it is applied to Snort in detection mode. The QoS configuration in layer 3 Cisco switches provides the capability to differentiate among different classes of traffic and to prioritize the traffic in times of network congestion, according to relative importance. The research described in this paper uses QoS queue technology to increase monitoring speed rather than reducing or slowing down traffic speed [4]. We used parallel technology to speed up the throughput of NIDPS packets processing to the level of the arrival traffic speeds. Our novel architecture addresses the weakness of NIDPS performance detection caused by increasing network traffic speed. The strength of this work over previous work is that in previous work [4, 5], it was shown that analysis can fail in high speed and high volume traffic in terms of the fact that many packets are dropped or left outstanding and not analysed. But in analysis mode, dropped packets refer to packets that eventually do not get through to the destination and outstanding packets are those left unprocessed in the system. In this paper we show that detection can also fail. This is more significant because failing to detect, i.e. failing to recognize malicious packets, means that attacks can penetrate the system. The contribution of this work is to offer a solution to the problem of detection in high speed traffic based on network switch, QoS and parallel technologies. In the next section the background to our research is provided.

3. BACKGROUND

Security products, such as firewalls and antivirus programs, are less efficient than intrusion detection and prevention systems (IDPSs) and have different functionalities. IDPSs analyse collected information and infer more useful results than other security products [4, 5]. However, some researchers have indicated that whilst IDPSs have significantly improved with the passage of time, they still often produce an unacceptable quantity of false positives and false negatives [4, 10, 11, 12]. In addition, it is difficult to detect suspicious activities in the midst of high traffic and other such adverse circumstances in the network, consequently resulting in an inaccurate detection mechanism.

IDPSs consist of either software applications or hardware that listen to and detect malicious activities at the gateway (incoming and outgoing) of individual or network systems. IDPSs are capable of monitoring, identifying and reporting evidence of malicious activities and attacks, such as flood attacks, unauthorised log-ins, privilege escalation, illegitimate access, modification of data and data-driven attacks, [4, 5]. Therefore, an IDPS sniffing mechanism is effectively applied at the network gateway, which provides useful information about packets and traffic to security professionals [4].

The specialised IDPS mechanism is based on how, where and what it detects/prevents, along with mandatory requirements. In particular, IDPSs should be based on flexible and scalable network components to accommodate the drastic increase in today’s network environments. They should also provide straightforward management and operational procedures and steps, instead of procedures that complicate the underlying tasks. Lastly, they should provide user-friendly IDP mechanisms [4, 5].

3.1 Network Intrusion Detection and Prevention System (NIDPS)

NIDPSs are used to analyse the traffic in all Open Systems Interconnection (OSI) layers, to enable differentiation of normal traffic as opposed to suspicious activities. They are also used to detect and react to the unauthorised access to network systems, [4, 12]. There are three modes of NIDPS: analysis mode (sniffer mode); detection mode (passive mode); and prevention mode (inline mode).

Analysis mode is used to recognise and display the type of packets coming into the network. Various levels of detail can be displayed on the console, for instance application data which is attached to the packet additionally to TCP, UDP and ICMP header information [4, 7]. The detection system is capable of detecting suspicious activity and generating alerts based on recognised signatures and rules [4, 7]. Signature analysis is generally based on patterns inside the data packet. This technique aims to detect multiple kinds of attacks such as the presence of scripts in packets destined for web services [4, 7]. Alternatively, anomaly-based NIDPSs notice packet anomalies available in the header parts of the protocol [7]. Logging and alerts depend on the nature of what is detected inside the packets. If any suspicious activity is found inside a packet, the packet usually logs the malicious activity and/or generates an alert. Logs are usually stored in simple text-based files [4, 7, 8]. Output modules (plug-ins) are capable of performing multiple operations depending on the results generated by the logging and alerting system. In general, output modules control the form of outcome produced by the logging and alerting system [7, 8].Network intrusion prevention systems (NIDPSs) are active, inline devices in a network that can drop, block or reject packets and or stop malicious connections before these reach the targeted system [4, 12]. NIDPSs are further classified into software and hardware based. Hardware based NIDPSs are effective and can overcome some performance issues of software-based NIDPS but high cost is an issue. One of the most popular software-based NIDPSs is Snort [4].

3.2 Snort Network Intrusion Detection and Prevention System (Snort-NIDPS)

Snort is released as an open-source, rule-centred NIDPS, which stores information in text files that can be modified by a text editor [4, 8]. Snort rules activate on the network IP layer and TCP/UDP layer protocols. Rules are grouped into categories, and the rules belonging to each category are stored as information in separate files; these files are then integrated into the main configuration file, named "snort.conf". The data is captured in terms based on described rules, which are read at the initialisation of Snort and are used to construct the internal data structure [4, 7, 8]. Furthermore, Snort is a combination of both basic signature code analysis and content-driven rules [7, 8]. Snort can execute a protocol analysis and a search and match of the content. It can be utilised for the detection of various attacks and probes, such as those regarding stealth port scans, buffer overflow, SMB probes, CGI attacks, fingerprinting attempts of OS and many more [4, 5]. Snort uses the rules, which have been written by using a flexible language that can be managed by developers and the executer. These rules identify the traffic types that can be passed or collected, and they can function as a detection engine, as well [4, 5, 7].

There are several features available for Snort; the most common feature is its real-time alerting mechanism. Alerts can also be collected by using a mechanism for syslog, which allows the reporting of suspicious activities in logs for additional investigation, a UNIX socket, a specified file of user, or a WinPopup message to the window client [7, 8]. Hence, Snort is different from other packet sniffers, due to the tcpdump sniffer, which has the capability to be run by different operating systems, and the use of the hexdump payload dump that tcpdump has employed during recent years. Snort also has the capability to display packets via different networks through the same method. Snort is a multi-mode IDPS and NIDPS providing analysis, detection and prevention [4, 7, 8]. It is capable of reading chains (internal data structures), which have to be matched against all packets. If a packet does not match any rule, it will be passed; otherwise appropriate action is taken. The default detection method of Snort NIDPS is the signature-based detection system, which can utilise rules to search or match any errant packets on the controlled network. In turn, the alerts are activated and sent to a receiver such as system log, database, management team or even a trap. Many studies have used Snort NIDPS to detect attacks such as DoS and DDoS by developing and designing new rules [13, 14, 15, 16].

4. Experiment Design

This research carried out two sets of experiments. The first set was carried out to show the weakness of Snort NIDPS in detection mode in high speed and high volume traffic (results are given in section 5). The second set was carried out to show the effectiveness of our novel architecture (results given in section 6.2). A set of facilities was needed in order to demonstrate Snort NIDPS weakness and later show how such weakness can be overcome through the use of our novel architecture.

The experimental set up supports acquisition, analysis of the data, and detection of various types of traffic. Tools included: the Snort 2.9.7.2, which was issued in October 2014; the WinPcap tool, to capture packets on Windows and Linux OSs; the NetScanPro tool, to manage traffic in different time scales; the Packets Generator tool, to generate (ICMP, UDP and TCP) traffic at different speeds and values; and the Flooder Packets tool to generate flood traffic and malicious UDP packets (threads) in high-load traffic and high speed.

Figure 1 shows our experimental set up. Snort NIDPS was implemented in parallel on four work stations while a fifth workstation was used to generate traffic. All workstations were connected through a CISCO SW 3560 switch. Performance metrics were evaluated in the experiments to measure the ability of Snort NIDPS detection mode. The Snort breakdown analysis includes the following metrics: the number of packets analysed of the total packets received; the number of Eth (Ethernet) packets received of the total packets analysed; the number of IP packets received of the total Eth packets received; and the number of TCP, UDP and ICMP packets analysed. The Snort action statistics provides: the number of packet alerts of the total TCP/IP packets analysed; and the total packets logged of the total TCP/IP packets analysed. These parameters indicate NIDPS performance.

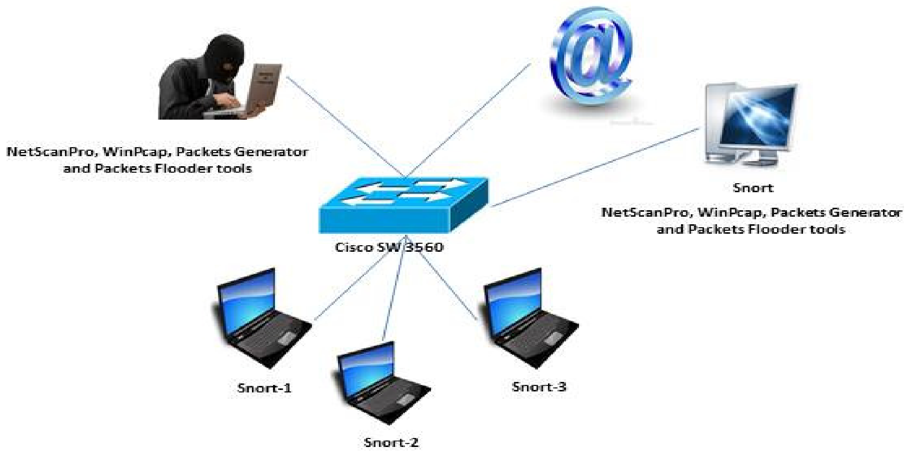

NetScanPro, WinPcap, Packets Generator and Packets Flooder tools
Snort
NetScanPro, WinPcap, Packets Generator and Packets Flooder tools
Cisco SW 3560
Snort-1
Snort-2
Snort-3

Figure 1. Simple network design

5. EXPERIMENTS, IMPLEMENTATION AND RESULTS

We ran (4) experiments to test Snort NIDPS reactions to (1) detect ICMP header under different speed traffic; (2) detect UDP header under different speed traffic; (3) detect TCP header under different speed traffic; and (4) detect malicious packets under different speed traffic. For each experiment, we ran three (3) consecutive tests; for each test the speed at which the packets were sent was increased each time. The packets were sent at interval times of various millisecond (ms). Each packet carries1KByte. Snort was run in detection mode. We created some rules to alert and log unwanted traffic. We also used the Packet Generator, WinPcap and Flooder packets tools to generate packets and threads (malicious UDP packets) through the network and hosts at different speeds.

5.1 Experiment 1: Detecting (alert and log) ICMP Header

In this experiment, more than 1 million IP/ICMP packets have been sent at different speeds (10ms, 5ms and 1ms intervals). In fact, this first common rule does a good job of testing if Snort is working well and if it is able to generate all alert actions:

Alert icmp any any ->any any (msg: ”Detect ICMP Packets”; sid:100001;).

Snort will alert and detect any ICMP packets from any sources to any destinations address from and to any ports.

Table 1. Snort reaction to ICMP header

Traffic speed per milliseconds	Machine packets received	% packet analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packet analysed	UDP packets analysed	Number of packets alert	Number of packets logged	% packet alerts	% packet logs
10ms	100%	4.300%	100%	1874	0	44459	1874	1874	100%	100%
5ms	100%	1.120%	100%	345	0	13463	231	231	66.96%	66.96
1ms	100%	0.141%	100%	730	0	1144	405	405	55.47%	55.47

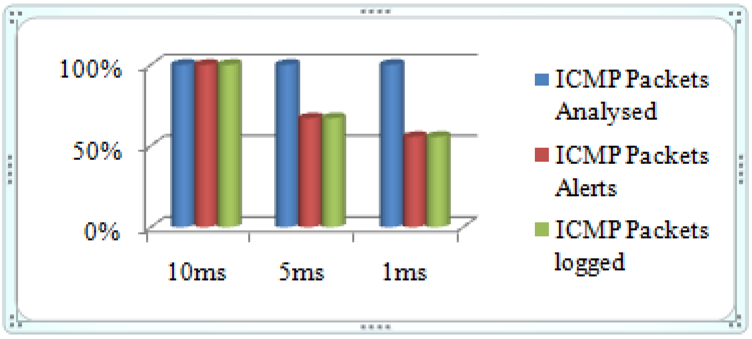


Figure 2. ICMP packets detection.

As the results show in Figure 2, Snort analysed every packet that reached the wire. When ICMP traffic was sent at 10ms, Snort alerted and logged nearly 100% of the total ICMP packets analysed (see Table 1). As the speed increased from 10ms to 1ms, Snort started missing alerts and logged packets. Also, Figure 2 shows that the number of missed alerts increased when the speed increased. The experiment shows that Snort detected 55.47 % of the total ICMP packets that it analysed (see Table 1).

5.2 Experiment 2: Detecting (alert and log) UDP Header

In this experiment, more than 1 million IP/UDP packets were sent at different speeds (10ms, 5ms, 3ms and 1ms), and the following rule was written to allow Snort to detect any UDP packets from any sources to any destination address and to any source and destination ports:

Alert udp any any ->any any (msg: ”Detect UDP Packets”; sid:100002;).

Table 2. Snort reaction to UDP header.

Traffic speed per milliseconds	Machine packets received	% packets analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Number of packets alerts	Number of packets logged	% packets alerts	% packets logged
10ms	100%	11.293%	100%	0	0	7854	7798	7798	99.28%	99.28%
5ms	100%	3.128%	100%	0	0	2958	2896	2896	97.90%	97.90%
3ms	100%	1.274%	100%	0	0	1358	946	946	69.66%	69.66%
1ms	100%	1.006%	100%	0	0	65	30	30	46.15%	46.14%

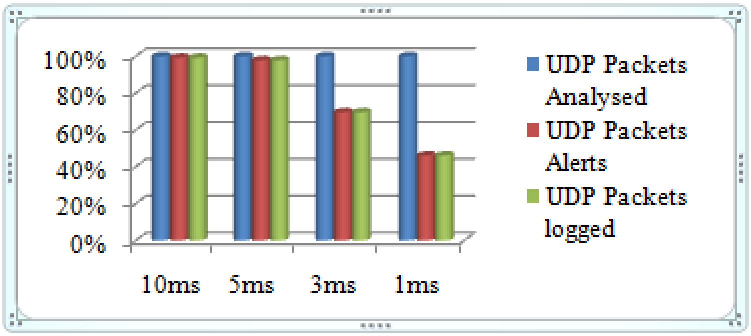


Figure 3. UDP packets detection.

As shown in Figure 3, when UDP traffic was sent at a speed of 10ms, Snort alerted and logged nearly 100% of the total UDP packets that it analysed (see Table 2). When the traffic’s speed increased to 5ms, Snort detected 97.90% of the total UDP packets analysed (see Table 2). Figure 3 shows that, as the speed increased, missed alerts and logs also increased. This experiment shows that Snort detected 46.14% of the total UDP packets that it analysed (see Table 2).

5.3 Experiment 3: Detecting (alert and log) UDP Header

Here, more than 1 million IP/TCP packets were sent at different speeds (10ms, 5ms and 1ms). The following rule was made to allow Snort to detect any TCP packets from any sources to any destinations, from and to any ports:

Alert tcp any any ->any any (msg: ”Detect tcp Packets”; sid:100003;).

As shown in Figure 4 Snort analysed every packet that reached the wire. The experiment shows that Snort detected all TCP packets that it analysed, even if the speed increased (see Table 3). This effectiveness occurred because TCP does not send the next packet until it receives an acknowledgement that the previous package has been received. These acknowledgements make the TCP packet slower than the UDP and ICMP packets.

Table 3. Snort reaction to TCP header

Traffic Speed per milliseconds	Machine packets received	% packets analysed	Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Number of packets alerts	Number of packets logged	% of packets alerts	% of packets logged
10ms	100%	51.567%	100%	128	51070	25	5170	5170	100%	100%
5ms	100%	3.122%	100%	110	33113	31	33113	33113	100%	100%
1ms	100%	0.981%	100%	0	14622	249	14622	14622	100%	100%

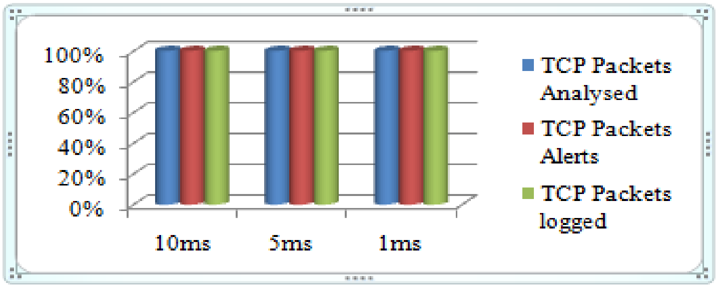


Figure 4. TCP packets detection.

5.4 Experiment 4: Detecting Malicious Packet (UDP Threads)

In this experiment, WinPcap and Flooder packet tools were used to send flood traffic with malicious UDP packets (threads) to specific hosts or networks at different speeds. The UDP malicious packets contain variables and time to live 128. The following rule is written to permit Snort to alert and log any UDP threads or malicious packets that contain the variables 'abcdef' and time to live (TTL) 128 that comes from any source and port address and goes to any destination address and ports:

Alert udp any any ->any any (msg: "Detect Malicious UDP Packets"; ttl: 128; content:|' 61 62 63 64 65 66 '|; Sid: 100004 ;)

This experiment is different from the previous ones. The previous experiments tried to detect headers, such as TCP, UDP and ICMP. The system received the TCP, UDP and ICMP packets at different speeds, but in this experiment, we sent flood traffic in different bandwidths (speeds) with malicious UDP packets (threads) in interval packets with a delay of 1 microsecond (1 mSec), and then we tried to detect only the UDP threads by using two conditions of additional rules (TTL and content). These two key rules will detect any UDP malicious packet that is matched in order to determine that the TTL value is equal to 128 and to determine if a data pattern inside the malicious packet has variables ('abcdef'). However, the hexadecimal number ('61 62 63 64 65 66'), which the rule contained, is equal to the ASCII characters ('a b c d e f').

Table 4. Snort reaction to udp malicious packets.

flood traffic (Byte PerSeconds) With 255 UDP malicious packets in (1mSec)	Total Eth received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	Number of the malicious packets Alerts	Number of the malicious packets logged	%of packets alerts	%of packets logged
16 Bps	100%	0	0	9868	9820	9820	99.51%	99.51%
32 Bps	100%	0	0	8702	8654	8654	99.44%	99.44%
200 Bps	100%	0	0	7166	7083	7083	98.84%	98.84%
1200 Bps	100%	0	0	6024	5854	5854	97.17%	97.17%
4800 Bps	100%	0	0	2876	1421	1421	49.40%	49.40%
60000 Bps	100%	0	0	7560	2810	2810	35.75%	35.75%

As shown in Figure 5, Snort analysed every packet that reached the wire. When malicious UDP packets were sent at a speed of 1 mSec and flood traffic at 16 bytes per second (Bps), Snort alerted and logged more than 99% of the total UDP packets that it analysed. As the flood traffic (speed) was increased to 200, 1200, 4800 and 60000 bytes per second (Bps), Snort alerted and logged packets to a decreasing degree, respectively, at 98.84, 97.17, 49.40 and 35.75% of the total malicious packets analysed (see Table 4). Figure 5 shows that the number of missed malicious packet alerts increased when the speed increased. The experiment shows that, when the speed was 60000 Bps, Snort only detected nearly 35 of 100% of the malicious packets analysed (see Table 4).

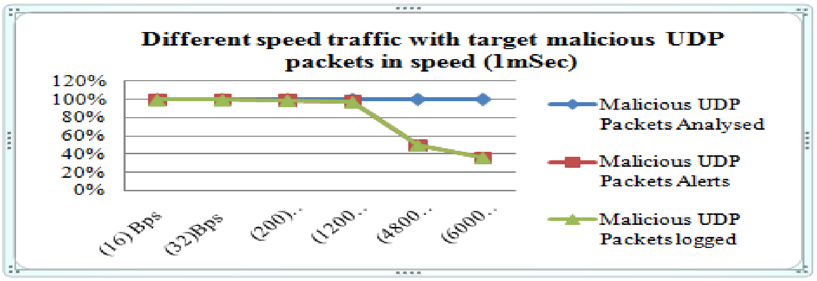


Figure 5. Malicious packets detection.

6. Proposed Solution and Evaluation

This section proposes a novel configuration as a solution to the problem of NIDPS dropped packets illustrated in section 5. It also presents an evaluation of the solution.

6.1 Proposed Solution

A critical analysis was conducted for experiments 1, 2 and 4 (see Figures 2, 3 and 5, respectively). It was found that Snort's performance detection throughout was affected by high-speed traffic, and there were more missed alerts and logs for packets as the speed of traffic increased. Also, when the malicious traffic was sent at high speed, it was found that Snort increased its missed malicious packet alerts and logs (see Figure 5), because Snort is capable of performing as a real-time traffic processor on the network. It is a multimode packet tool that can perform network traffic analysis, detection and content searching/matching in both real-time and for forensic post-processing [4, 5]. Although Snort has a limited time frame for processing packets and then alerting and logging any packets and malicious traffic successfully, because of the limitation of buffer size and processor speed (section 7 explains this more). If a network's traffic speed is higher than Snort's limit, Snort will miss alerts and logs.

To address this problem, A QoS configuration has been suggested in Layer-3 Cisco switches with parallel NIDPS nodes to increase packets throughput processing speed, even if traffic arrives at a high speed. Some mechanisms that QoS offers are queue, classification, policing and marking technologies, which can give a switch a new logical throughput-traffic-forwarding plan. A configuration of QoS offers two (2) input queues (ingress queues) and four (4) output queues (egress queues) at the physical switch interfaces (SVI or ports).As shown in Figure 6, the switch has been configured to two ingress queues and four egress queues to load a set of bytes (packets) into a number of processor input queues equally and to divide traffic (as a number of bytes) into parallel streams in order to speed up packets processing. It then uses parallel NIDPS to analyse each portion of traffic individually to determine whether it is free of malicious codes. A novel class map (marking) and a policy (policy map) were made for each input queue. The class map recognises and classifies a certain type of traffic for each input queue, and the policy map controls and recognises the speed limit for each input queue and applies it to interfaces [4]. The bandwidth, threshold, buffer, memory allocated and priority were configured for each ingress queue and each egress queue to treat and control traffic in order to help prevent congestion or disabled traffic in the input queues, even if traffic comes in at high load and speed.

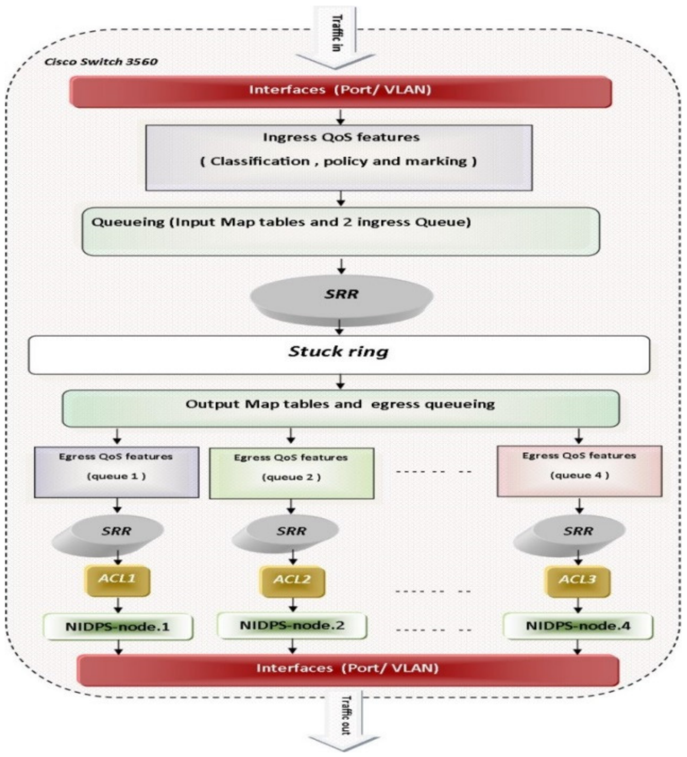

Traffic in
Cisco Switch 3560
Interfaces (Port/ VLAN)
Ingress QoS features
(Classification , policy and marking)
Queueing (Input Map tables and 2 ingress Queue)
SRR
Stuck ring
Output Map tables and egress queueing
Egress QoS features (queue 1)
Egress QoS features (queue 2)
Egress QoS features (queue 4)
SRR
SRR
SRR
ACL1
ACL2
ACL3
NIDPS-node.1
NIDPS-node.2
NIDPS-node.4
Interfaces (Port/ VLAN)
Traffic out

Figure 6. Parallel NIDPS node with QoS architecture.

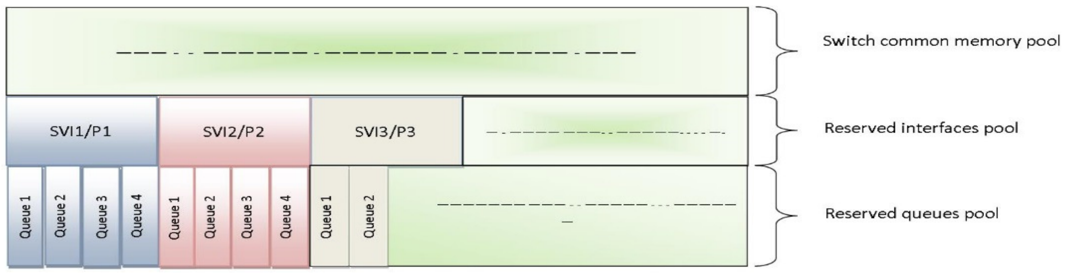


Figure 7: novel egress Queue buffer space reservation

QoS Shaped/Share Round Robin (SRR) technology was used to guarantee the bandwidth for an interface, and also for each ingress queue and egress queue. A network device must be able to identify packets in all the IP traffic flowing through it. The Shaped task only exists on output queues, and a queue books a percentage of a total port's bandwidth. One queue may be set as priority and queue do not share bandwidth. The Share function (SRR) is offered on both input and output queues. It provides a queue a portion of a total port's bandwidth, but unused bandwidth can be used by any queue. In our novel architecture (see Figure 6), the Share queue is used in input queues to help prevent congestion, and each output queue has an individual processing by using the Shaped queue to control speed traffic (bandwidth) A single NIDPS node is implemented for each output queue. One of the queues can be the expedited queue, which is serviced until empty before the other queues are serviced.

Also a memory buffer queue reservation was configured for each queue which provides more buffer space over its limit (over 100% of buffer space) when needed by reserving more space from available queue buffer, ports or SVI memory buffer or switch common memory pool buffer (see Figure 7). However, headers, such as ICMP, TCP and UDP, and even hackers have different characteristics, features and techniques. Using SRR, Threshold and Priority methods for each output queue can offer a wider range to deal with the behaviour of different IP headers and hackers.

The main aim of our novel architecture design is to manage and allocate a traffic load (number of bytes) into each input queue and process each output queue individually in order to permit a limited group of bytes that are divided into output queues to be processed at same time, thereby increasing NIDPS throughput processing time and reducing traffic congestion, even if the traffic is high load and speed.

6.2 Evaluation of Solution

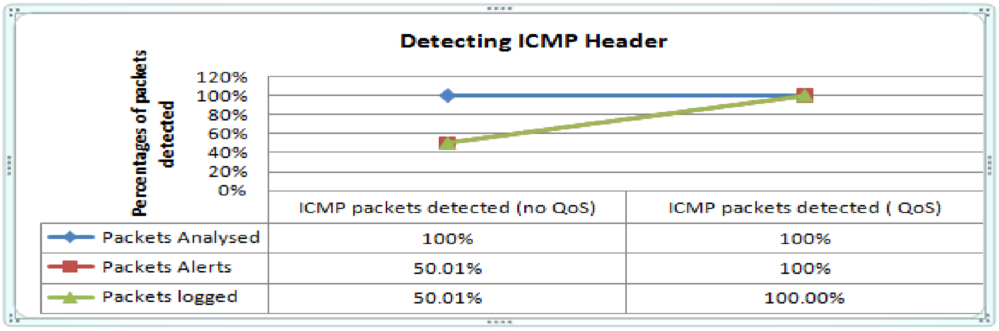
We ran (4) experiments to test performance of our novel NIDPS architecture design to detect: (1) ICMP header; (2) UDP header; (3) TCP header; and (4) Malicious packets. Each experiment tests Snort's detection rate without and with QoS and parallel technologies under high-speed traffic.

6.2.1 Experiment 5: Parallel Snort with QoS Reaction to Detect ICMP Header

In this experiment, more than 38,000 ICMP/IP packets were sent in high-speed traffic (1ms). Each packet carried 1KByte.

Table 5: Snort with QoS reaction to ICMP header in high speed traffic

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of the total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	ICMP packets alerts	ICMP packets logged	% packets alerts	% Packets logged
Without QoS	39121	37.259%	100%	14438	0	97	7220	7220	50.007%	50.007%
	Snort Processor Times = 155s -> (Pkts/min:7228 – Pkts/sec:94)									
With QoS	38668	99.943%	100%	37393	0	757	37393	37393	100.00%	100.00%
	Snort Processor Times = 293s -> (Pkts/min:9661 – Pkts/sec:131)									



	ICMP packets detected (no QoS)	ICMP packets detected (QoS)
Packets Analysed	100%	100%
Packets Alerts	50.01%	100%
Packets logged	50.01%	100.00%

Figure 8: Snort with QoS reaction to detect ICMP packets in 1ms.

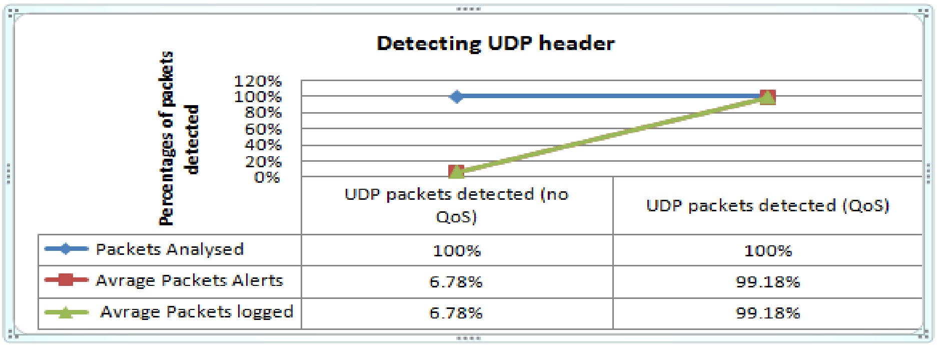
As the results show in Figure 8 and Table 5, when more than 38,000 ICMP/IP packets were sent in an interval time of 1ms, packet processing speed was 94 packets per second (94 Pkts/sec) and number of alerts and logs was 7220 of the 14,438 ICMP packets that were analysed (see Table 5). When the same number of packets was sent at the same speed, using QoS and parallel technologies, packet processing speed was increased from 94 to 131 Pkts/sec and Snort detected all of the ICMP packets that it analysed (see Figure8). This experiment shows that when Snort NIDPS was used without QoS, it only detected 50% of the total packets analysed, but when Snort was used with QoS, Snort detected 100% of the total packets that it analysed (see Figure 8 and Table 5).

6.2.2 Experiment 6: Parallel Snort with QoS Reaction to Detect UDP Header

In this experiment, more than 38,000 UDP/IP packets were sent in high-speed traffic (0.5ms). Each packet carried 1KByte.

Table 6: Snort with QoS reaction to UDP header in high speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	UDP packets Alerts	UDP packets logged	% packets alerts	% packets logged
Without QoS	36213	12.391%	100.00%	763	0	59	4	4	6.780%	6.780%
	Snort Processor Times = 59s -> (Pkts/sec:76)									
With QoS	37783	99.942%	100.00%	0	0	37564	37257	37257	99.182%	99.182%
	Snort Processor Times = 292 -> (Pkts/min:9440 – Pkts/sec:129)									



	UDP packets detected (no QoS)	UDP packets detected (QoS)
Packets Analysed	100%	100%
Avrage Packets Alerts	6.78%	99.18%
Avrage Packets logged	6.78%	99.18%

Figure 9: Snort with QoS reaction to detect UDP packets in 0.5ms

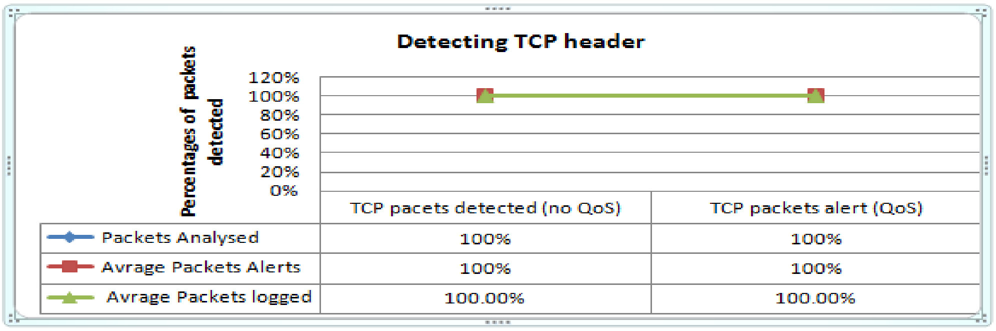
As the results show in Figure 6 and Table 6, when the traffic (packets) was sent at an interval time of 0.5ms, the number of packet alerts and logs was nearly 4 of the 59 UDP packets that were analysed with packet processing speed of 76 Pkts/sec (see Table 6). It detected fewer than 7% of all UDP packets that it analysed (see Figure 9). When QoS architecture was implemented and packets were sent again in the same traffic and interval speed of 1ms, the packet possessing speed was increased to 129 Pkts/sec and Snort detected more than 99% of the total UDP packets analysed (see Figure 9). This experiment shows that the Snort NIDPS performance detection improved from 7 to 99% when the novel QoS configuration was used.

6.2.3 Experiment 7: Parallel Snort with QoS Reaction to Detect TCP Header

In this experiment, more than 38,000 IP/TCP packets were sent in high-speed traffic (0.5ms); each packet carried 1KByte.

Table 7: Snort with QoS reaction to TCP Header in high speed traffic

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	TCP packets Alerts	TCP packets logged	% Packets alerts	% Packets logged
Without QoS	32108	1.884%	100.00%	0	497	85	497	497	100.00%	100.00%
	Snort Processor Times = 34s -> (Pkts/sec:17)									
With QoS	31058	99.997%	100.00%	0	30057	779	30057	30057	100.00%	100.00%
	Snort Processor Times = 322s -> (Pkts/min:6211 – Pkts/sec:96)									



	TCP pacets detected (no QoS)	TCP packets alert (QoS)
Packets Analysed	100%	100%
Avrage Packets Alerts	100%	100%
Avrage Packets logged	100.00%	100.00%

Figure 10: Snort with QoS reaction to detect TCP packets in 0.5ms.

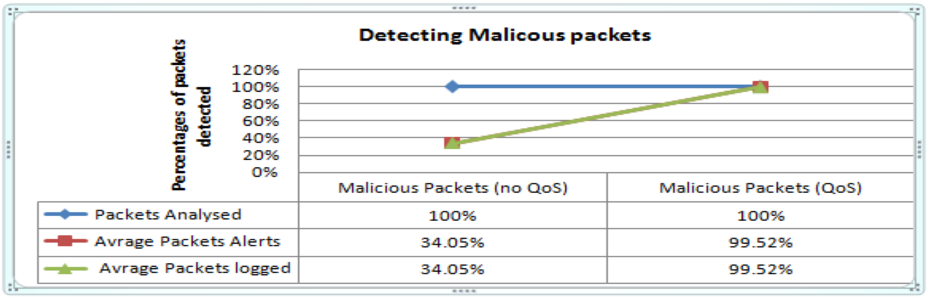
Figures 10 and Table 7 show that Snort detected 100% of the total TCP packets analysed, even without QoS. The packet processing speed is improved from 17 to 96 Pkts/sec with QoS.

6.2.4 Experiment 8: Parallel Snort with QoS Reaction to Detect Malicious Packets

In these experiments, a flood traffic was generated with UDP malicious packets (threads) in high-speed traffic (60000Bps, flooded traffic with 225 threads, sent at an interval time of 1mSec) by using NetScanPro, WinPcap and Flooder packets tools. Two tests were conducted: one test of Snort without QoS and one of Snort with QoS.

Table 8: Snort with QoS reaction to malicious packets in high speed traffic.

Test type	The number of Packets received	Total packets analysed of the total the packets received	Total Eth packets received of total packets analysed	ICMP packets analysed	TCP packets analysed	UDP packets analysed	UDP malicious packets Alerts	UDP malicious packets logs	% packets alerts	% packets logged
Without QoS	985686	1.004%	100.00%	0	0	793	270	270	34.048%	34.048%
	Snort Processor Times = 125s -> (Pkts/min:5090 – Pkts/sec:82)									
With QoS	996005	99.999%	100.00%	0	0	995155	990344	990344	99.517%	99.517%
	Snort Processor Times = 19.22m -> (Pkts/min:52421 – Pkts/sec:857)									



	Malicious Packets (no QoS)	Malicious Packets (QoS)
Packets Analysed	100%	100%
Avrage Packets Alerts	34.05%	99.52%
Avrage Packets logged	34.05%	99.52%

Figure 11: Snort with QoS reaction to detect malicious packets in high speed traffic.

As the results show in Figures 11 and Table 8, when malicious traffic was sent at high speed and volume, the number of malicious packets detected was 270 of the 793 malicious packets analysed and the packet processing speed was 82 Pkts/sec (see Table 8). Snort detected fewer than 35% of the total malicious packets analysed (see Figure 11). When the same traffic was generated with the same speed and value, but Snort was supported by the novel QoS architecture, the packet processing speed wan improved from 82 to 857 Pkts/sec and Snort detected more than 99% of the total malicious packets that it analysed (see Table 8). This experiment showed that the Snort NIDPS performance detection improved while novel QoS was used.

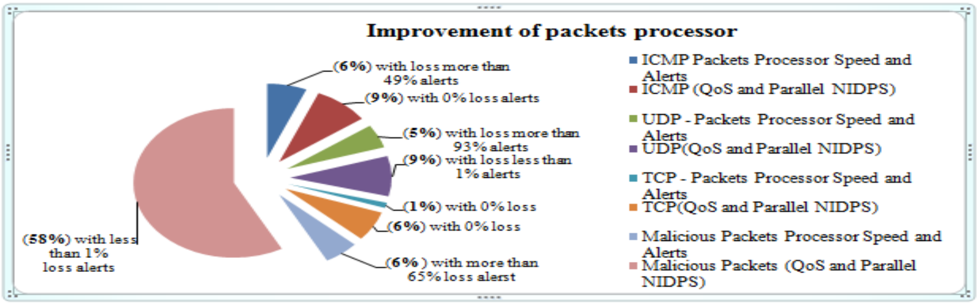


Figure 12: Improve NIDPS detection performance and packets processor through QoS and Parallel technologies.

Figure 12 provides a synopsis of our results. The experiments show that packets processor speed and Snort NIDPS detection performance have been improved when we used QoS, queue and parallel NIDPS technologies (see Figure 12).

7. TECHNICAL DISCUSSION

The performance of an NIDPS could be described as ineffective if the NIDPS is unable to detect or stop unwanted packets that could reach the system. There are two main causes of ineffective NIDPS: buffer size and processing speed.

When traffic moves through the network interface card (NIC) to the NIDPS node, the packets are stored on the buffer until the other relevant packets have completed transmission to processing nodes. In the event of high speed and heavy traffic in multiple directions, the buffer will fill up. Then packets may be dropped or left outstanding [20, 21, 22]. In this case, there is no security concern about the packets dropped; the packets are dropped outside the system. The outstanding packets that are waiting or have not been processed by the NIDPS node may affect the system.

However, packets can also be lost at the host. Most software tools use a computer program such as the kernel, which manages input/output (I/O) requests from software and decodes the requests into instructions to direct the CPU's data processing. When traffic moves from the interface (NIC) through the kernel's buffer to the processor space, where most of processing nodes are executed, the packets will be held in the kernel buffer before being processed by the CPU. When some nodes experience a high volume of data, the buffer will fill up and packets may be dropped. Configuring the kernel parameter in the New Application Programming Interface (NAPI) can enhance kernel performance by increasing the level of optimization and selecting multivariate features such as kernel complex quantitative near-infrared (K-NIR), kernel support vector regression (k-SVR), or kernel partial least squares (K-PLS) to improve the accuracy of packet processing [23, 24, 25, 26]. In order to hold and process packets quickly, these kernel performance enhancements pull a high value of packets from interfaces and bind them with obtainable CPU cycles, which limit packet speed and time and have no buffer memory. Furthermore, it requires a great deal of CPU to process a vast amount of data buffered in the kernel; the CPU cycles may run out of time. In this case, the packets that were dropped in the kernel and NIC might drop very early in the CPU cycles, which cannot buffer packets [27, 28]. In these two cases of host and processor packets loss, the NIDPS node is affected because packets are dropped before it is analysed. In this work, we are not focused on the network-based packets drop (NIC interfaces) as much as focusing on the host and processor based packets loss, because in the network-based, the packets are dropped from NIC and do not hit the system, but in the host based, the packets go through the system and then are dropped, which affects NIDPS detection performance. Our solution uses multi-layer switch technology, which supports QoS. The parallel NIDPS nodes are associated with queues each with a specific buffer and bandwidth thus increasing the queue buffer size automatically over its limit when needed. The NIDPS is faced with high speed and volume traffic. The appropriate number of NIDPS nodes is dependent in network speed limit. We therefore needed to operate with the class and quality of service technologies within the network switch.

By default, most of the Cisco switches work in Layer 2, the Data Link layer, and use the Class of Services (CoS) value [4, 29, 30] (see Figure 23). In this layer there is insufficient commands to support switch features such as QoS features, dynamic access control lists (ACLs), VLAN features, static IP routing, Routing Information Protocol (RIP), Policy-based routing (PBR) Cisco-default Smartports, etc. Other mechanisms operate at Layer 3 (see Figure 13). However, in the network switch operation, all the traffic should has equal priority and an equal chance of being delivered in a manner that is timely for the network. Some packets have an equal chance of being dropped when congestion network traffic occurs. DiffServ (Differentiated Service) allows different types of service to be offered depending on a code. For instance there can be a policy to give a certain type of package priority. QoS implementation is based on DiffServ architecture, which specifies that each packet be classified upon entry and has an equal priority and an equal chance of being delivered into network which is adjusted for different traffic speed in a good and timely manner. Furthermore, QoS makes network performance more predictable and bandwidth utilisation more effective [29, 30, 31, 32].

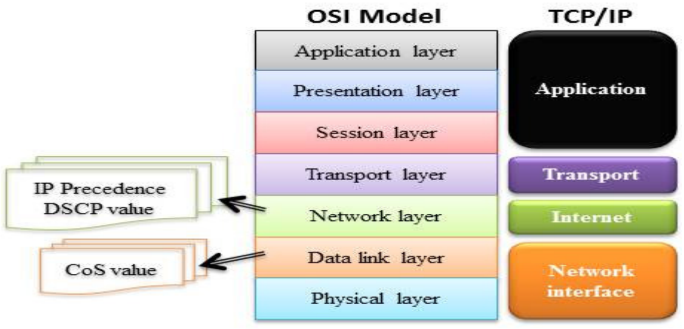


Figure 13. Place for CoS and DSCP values [4].

In our novel QoS architecture, the CoS values in Layer 2 were mapped to DSCP value (Differentiated Services Code Point) in Layer 3 so that appropriate categories could be matched at the network level. The static or dynamic classification methods involved Layer 3 header information matching, and a mechanism such as IP precedence or DSCP values was used to carry the IP packet header. For example, a dynamic classification access list can be used to identify IP traffic and place the traffic into a reserved queue [30, 31, 32]. Classification can also take place in the Layer 2 frame. Packet classification can be processor-intensive, so it should occur as far out toward the edge of the network as possible because every hop needs to make a determination on the treatment a packet should receive. A simpler classification is achieved through marking or setting the type of service (ToS) field in the IP header [29, 30, 31, 32]. In Layer 2, 802.1Q and 802.1p frames used 3 bits for IP type of service (ToS) field, and Layer 3 IPV4 packets used 6 bits for DSCP in (ToS) field to carry the classification (class) information (see Figure 14). The DSCP values allow for a higher degree of differentiation. CoS values range from 0 to 7 (8 values) and DSCP values range from 0 to 63 (64 values).

Regardless of the method by which the network is able to classify and identify IP traffic (either through port address information or through the ToS filed), those hops can then provide each IP packet with the required QoS. At that point, special techniques can be configured to provide priority queueing in order to ensure that large data packets do not interfere with packet data transmission. “If a node can set the IP Precedence or DSCP bits in the ToS field of the IP header as soon as it identifies traffic as being IP traffic, then all of the other nodes in the network can be classified based on these bits. However, in most IP networks, marking IP Precedence or DSCP should be sufficient to identify traffic as IP traffic” [4, 30, 31, 32].Differentiated services technology can be used such that each packet can be classified upon entry into the network and adjustments can be made for different traffic speeds and loads. In this work, the switch frame has been changed from Layer 2 to Layer 3. [4, 30, 31, 32].

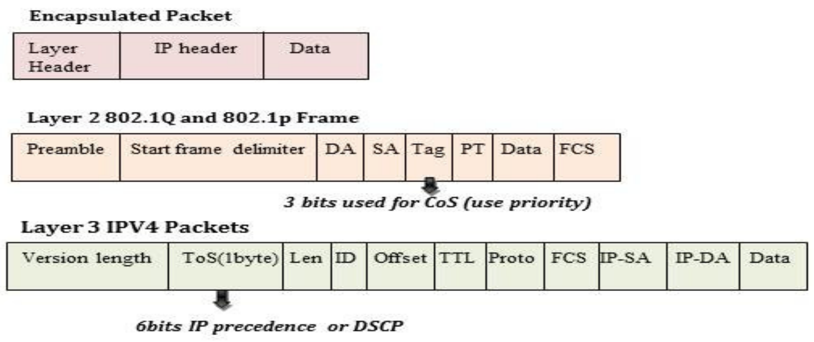


Figure 14. QoS classification Bits in Frames and Packets [30].

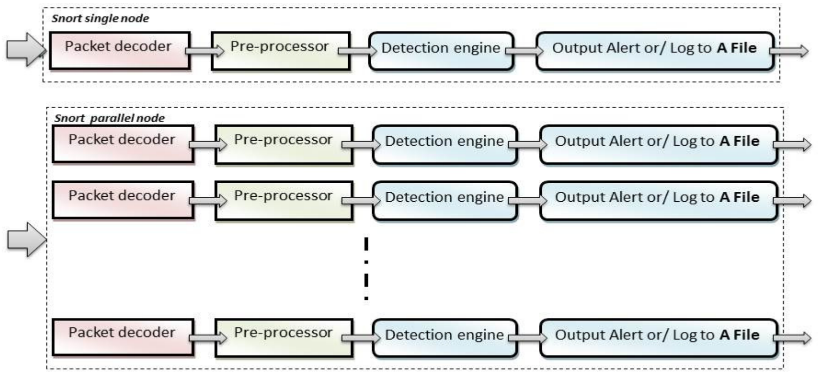


Figure 15: Snort NIDPS Parallel node.

Features of QoS, such as a policy map and class map, can be used to classify the traffic inside the switch with the same policy and class plan; different management can be given to packets with a different class and policy plan. Classification is process of identifying the data packets to a class or group in order to manage the packet appropriately [4, 30, 31, 32]. Network devices use several match criteria to place traffic into a certain number of classes. This can be processor-intensive if nodes must repeat classifications based on access list (ACL) matches. Therefore, nodes should mark packets as soon as they have identified and classified the IP traffic. Policing involves creating a policy that specifies the bandwidth limits for the traffic and applies it to the interface. Policing can be applied to a packet per direction and can occur on the ingress and egress interfaces [30]. Different types of traffic can be recognised in terms of, for instance, type and ports, and differentiated policies can be set.

In our work, Snort NIDPS is configured from single NIDPS detection node to multi NIDPS node (see Figure 15) and also configured CISCO QoS switch technology. Network QoS technology enables a new logical and throughput-traffic-forwarding plan to be implemented in the switch. A physical interface (port) was configured to two input queues (ingress queues) and four output queues (egress queues). A buffer was set for each queue in order to organise and hold more traffic by using a dynamic memory technology (buffer memory located). The buffer's memory space was divided between the switch common memory pool, the SVI, and the queue reserved pool (see Figure 7). We implemented a buffer allocation scheme to reserve a minimum amount for each egress buffer. Thus, all buffers cannot be consumed by one egress queue, and the system can control whether to grant buffer space to a requesting queue. Furthermore, a specific buffer memory space was defined for each queue, including ingress and egress thresholds. Packets were divided between two ingress and four egress queues via configured queue-sets. The remaining free common pool interfaces were set to reserve up to 50% of the available switch memory pool.

After all traffic has been placed into input queues and classes and policy based on their QoS requirements has been defined, appropriate services can be provided, for instance bandwidth guarantees, thresholds, memory buffering and priority servicing through an intelligent output queueing mechanism. The output queues were processed separately by implemented parallel NIDPS nodes in order to increase packet possessing speed. Other mechanisms that QoS offers is Shaped or Share Round Robin (SRR) technologies which can vary the bandwidth provided for the queues in the interface [17, 18, 19]. Shaped function (SRR) can guaranty each queue a bandwidth limit but queues cannot share bandwidth if queues reach their bandwidth limit. Share function (SRR) can guaranty a bandwidth limit for each queue and the other queues can share with each other if one of queues reached bandwidth limit. We utilized Share in the ingress queues but Shaped in the egress queues to ensure appropriate bandwidth for each egress queue.

Queue technology is placed at specific points in Cisco switches to help prevent congestion. The total inbound bandwidth of all interfaces may exceed a ring space of internal bandwidth [29, 30, 31]. After packets are processed through classification, policing, and marking, and before packets pass into the switch fabric, the system allocates them to input queues. Because multiple input queue interfaces can simultaneously send packets to output queue interfaces, outbound queues are allocated after the internal ring in order to avoid congestion. The SRR ingress queue sends packets to the internal ring, while the SRR egress queue sends the packets to the output queue. The novel configurable research architecture has a large limit of buffer space and a generous bandwidth allocation for each queue. Queue 1 was set as a priority queue for each ingress and egress queue, which allowed the system to prioritise packets with particular DSCP values and thereby allocate a large buffer. It also allows buffer space to be used more frequently, and then adjusts the thresholds for each queue and packets so that packets with lower priorities are dropped when queues are full. This allows the system to ensure that high priority traffic is not dropped.

Parallel NIDPS is a form of computation in which many NIDPS nodes work simultaneously, operating on the principle that the large incoming data can be divided into smaller sets, which are processed at the same time [4, 5]. Parallelism of NIDPS can occur at three general levels: the high-level processing node (entire system), the component level (part of the system with specific task), and the sub-component level parallelism (function within a specific task). In our novel QoS architecture, the entire Snort NIDPS was replicated on a number of machines, each of which processed a specific portion of the incoming traffic. Parallel queues (2 input queues and 4 output queues) were designed through QoS configuration on a switch virtual interface (SVI) where component level parallelism of NIDPS nodes were implemented with the aim of improving NIDPS throughput performance and reducing NIDPS processor time (see Figure 16).

The NIDPS node was configured from a single node NIDPS to a multi-node NIDPS. One group of rules were implemented for all nodes to perform a quick check for each packet with one group of rules to determine if the node contains the rule group associated with the given packet. Each node was configured to do a different type of task and therefore can be considered to be component level parallelism. Component level parallelism is defined as function parallelism of the NIDPS processing node. In component parallelism, individual components of NIDPS were isolated, and each output queue is given its own processing element. Furthermore, the component parallelism level could be created as a thread of a lightweight processing node and existing threads to schedule threads of NIDPS processing nodes.

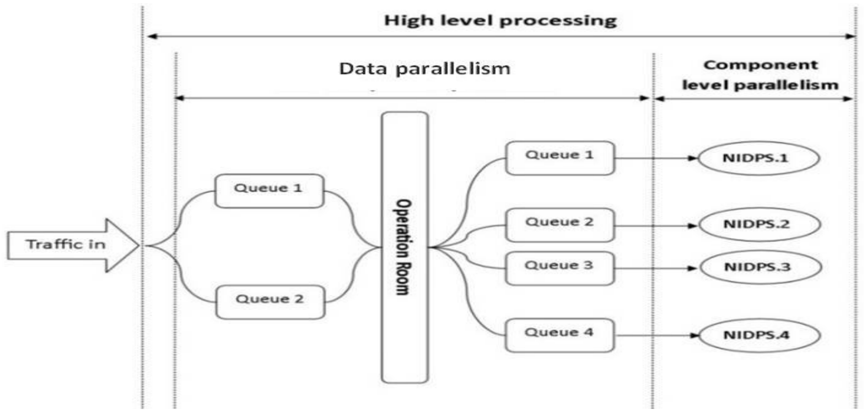


Figure 16: High level parallel processing.

8. CONCLUSIONS, RECOMMENDATIONS AND FURTHER WORK

8.1 Conclusion

NIDPS have become important components in securing today's computer networks. To be highly effective, an NIDPS must perform packet examination of incoming traffic at or near network speed. Failing to do so will permit malicious packets to infiltrate through the network undetected, and thus threaten network security. Our paper proposes and evaluates a new design of NIDPS architecture which uses a novel and unique infrastructure of QoS configuration and parallel technology in Layer 3 Cisco Catalyst switches to improve the NIDPS performance. The novel architecture reduces difficulties in maintaining security due to multiple characteristics of advanced computer networks, such as processing in real time, high speeds and high loads, which increase difficulties for defenders and reduce difficulties for attackers. Our experimental results show vast improvement in packet detection in such environments and therefore give better protection against attacks.

8.2 Recommendations and Further Work

Intruders usually use signatures that behave similarly to viruses used in computers. They also analyses data packets related to IP, which contains known anomalies, as either a single signature or a set of signatures. The detection system is capable of detecting suspicious activity in logs and generating alerts, based on these signatures and rules. NIDPSs are used to capture data and detect malicious packets that travel on the network media (cables, wireless) and match them to a database of signatures. Signature-based NIDPS are able to detect known attacks, but the major problem of the signature-based approach is that every signature should have an entry in a database in order to compare with the incoming packets. New signatures arise constantly and an issue is how to keep track up with new signatures. Another problem is processing time required to check all signatures. Knowledge sharing may provide a solution. Cloud computing which provides for massive processing distribution and sharing is a possible future direction [34, 35] but this also raises issues of trust. Our future work will investigate the use of specialized and trustworthy security clouds.

REFERENCES

[1] Nazer, G. M. & Selvakumar, A. A. L, (2011) "Current intrusion detection techniques in information technology—A detailed analysis", European Journal of Scientific Research, 65, 4, pp 611-24.

[2] Radhakishan, V. & Selvakumar, S, (2011) "Prevention of man-in-the-middle attacks using ID based signatures", In Proc. 2nd Int, Conf. Networking and Distributed Computing, (Sept. 2011). IEEE Press, pp165-169, DOI= http://dx.doi.org/10.1109/ICNDC.2011.40.

[3] Beg, S., Naru, U., Ashraf, M., & Moshin, S, (2010) "Feasibility of intrusion detection system with high performance computing: a survey", Int. J. Advances in Computer Science, 1 (Dec. 2010), pp26-35.

[4] Bul'ajoul, W., James, A., & Pannu, M. 2015 "Improving network intrusion detection system performance through quality of service configuration and parallel technology", Journal of Computer and System Sciences, 81, 6 (Sep. 2015), pp981–999, DOI= http://dx.doi.org/ 10.1016/j.jcss.2014.12.012.

[5] Bul'ajoul, W., James, A., & Pannu, M. (2013), Network intrusion detection systems in high-speed traffic in computer networks", In e-Business Engineering (ICEBE), 2013 IEEE 10th International Conference (Sept. 2013), IEEE, pp168-175, DOI= http://dx.doi.org/10.1109/ICEBE.2013.26.

[6] Sangeetha, S., Ramya, R., Dharani, M. K., & Sathya, P (2015) "Signature based semantic intrusion detection system on Cloud", In Information Systems Design and Intelligent Applications, Springer India, 339 (Jan. 2015), pp657-666, DOI= http://dx.doi.org/ 10.1007/978-81-322-2250-7_66.

[7] Rehman, R U (2003). Intrusion detection systems with Snort: advanced IDS techniques using Snort, Apache, MySQL, PHP, and ACID, Prentice Hall Professional.

[8] The Snort Documents (2014) [Online]. Available: https://www.snort.org/documents. [Accessed: 11-March-2015].

[9] Friedberg, I., Skopik, F., & Fiedler, R (2015) "Cyber situational awareness through network anomaly detection: state of the art and new approaches", E & i Elektrotechnik und Informationstechnik, 132,2 (March. 2015), pp101-105, DOI= http://dx.doi.org/ 10.1007/s00502-015-0287-4.

[10] Tesfahun, A., & Bhaskari, D. L (2015) "Effective hybrid intrusion detection system: a layered approach", International Journal of Computer Network and Information Security (IJCNIS), 7, 3 (Feb 2015), pp35, DOI= http://dx.doi.org/ 10.5815/ijcnis.2015.03.05.

[11] Jiang, W., Song, H., & Dai, Y (2005) "Real-time intrusion detection for high-speed networks", Computers and Security, 24, 4 (Jun. 2005), pp287–294, DOI= http://dx.doi.org/ 10.1016/j.cose.2004.07.005.

[12] Kenkre, P. S., Pai, A., & Colaco, L (2015) "Real time intrusion detection and prevention system", In Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014 (Jan. 2015), Springer International Publishing, 405-411, DOI= http://dx.doi.org/ 10.1007/978-3-319-11933-5_44.

[13] Khamphakdee, N., Benjamas, N., & Saiyod, S (2015) "Improving Intrusion Detection System Based on Snort Rules for Network Probe Attacks Detection with Association Rules Technique of Data

Mining", Journal of ICT Research and Applications, 8.3 (Mar. 2015), pp234-250, DOI= 10.5614/itbj.ict.res.appl.2015.8.3.4.

[14] Khamphakdee, N., Benjamas, N., & Saiyod, S (2014) "Improving Intrusion Detection System based on Snort rules for network probe attack detection", In Information and Communication Technology (ICoICT), 2014 2nd International Conference on, IEEE, (May. 2014), pp69-74, DOI= http://doi.org/10.1109/ICoICT.2014.6914042.

[15] Saboor, A., Akhlaq, M., & Aslam, B. (2013) "Experimental evaluation of Snort against DDoS attacks under different hardware configurations", In Information Assurance (NCIA), 2013 2nd National Conference on, IEEE, (Dec. 2013), pp31-37, DOI= http://doi.org/10.1109/NCIA.2013.6725321.

[16] Buchanan, W. J., Flandrin, F., Macfarlane, R., & Graves, J (2011) "A methodology to evaluate rate-based intrusion prevention system against distributed denial-of-service (DDoS)", In: Cyberforensics, (Jun. 2011).

[17] Chen, M. J., Hsiao, Y. M., Su, H. K., & Chu, Y. S (2015), "High-throughput ASIC design for e-mail and web intrusion detection", IEICE Electronics Express, 12 (Jan. 2015), DOI=http://doi.org/10.1587/elex.12.20140854.

[18] Jiang, H., Zhang, G., Xie, G., Salamatian, K., & Mathy, L (2013) "Scalable high-performance parallel design for network intrusion detection systems on many-core processors", In Proceedings of the Ninth ACM/IEEE Symposium on Architectures for Networking and Communications Systems (Oct. 2013), IEEE Press, pp137-146, DOI= http://dx.doi.org/10.1109/ANCS.2013.6665196.

[19] Tilera, TILE-Gx8036 tm Processor specification brief [Online]. Available: http://www.meganovo.cn/uploadfile/web/product/tilera/processor/TILE-Gx8036_PB033-02_web.pdf. [Accessed: 20-May-2015].

[20] Naouri, Y., & Perlman, R (2015) "Network congestion management by packet circulation", Washington, DC: U.S, Patent and Trademark Office , 8,989,017(Mar. 2015).

[21] Kishore, K. R., Hendel, A., & Kalkunte, M. V (2015) "System, Method and Apparatus for Network Congestion Management and Network Resource Isolation", Washington, DC: U.S. Patent and Trademark Office, 20,150,146,527 (May. 2015).

[22] Zhu, Y., et al (2015) "Packet-Level Telemetry in Large Datacenter Networks", In Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication, ACM, 15(Aug. 2015), pp479-491 ,DOI= http://doi.org/10.1145/2785956.2787483.

[23] Wu, H., Cadambi, S., & Chakradhar, S. T (2015) "Optimizing data warehousing applications for GPUs using dynamic stream scheduling and dispatch of fused and split kernels", Washington, DC: U.S. Patent and Trademark Office, 8,990,827 (Mar. 2015).

[24] Lutz, T., Fensch, C., & Cole, M (2015) "Helium: a transparent inter-kernel optimizer for OpenCL", In Proceedings of the 8th Workshop on General Purpose Processing using GPUs , ACM, (Feb. 2015), pp70-80, DOI= http://doi.org/10.1145/2716282.2716284.

[25] Fraser, N. J., Moss, D. J., Lee, J., Tridgell, S., Jin, C. T., & Leong, P. H (2015) "A Fully Pipelined Kernel Normalised Least Mean Squares Processor For Accelerated Parameter Optimisation", In Proc. International Conference on Field Programmable Logic and Applications (FPL), page to appear.

[26] Lee, J., Chang, K., Jun, C. H., Cho, R. K., Chung, H., & Lee, H. (2015) "Kernel-based calibration methods combined with multivariate feature selection to improve accuracy of near-infrared spectroscopic analysis", Chemometrics and Intelligent Laboratory Systems.

[27] Emmerich, P., et al (2015) "Optimizing Latency and CPU Load in Packet Processing Systems", In International Symposium on Performance Evaluation of Computer and Telecommunication Systems (SPECTS).

[28] Smith, S. C., Hammell, R. J., Parker, T. W., & Marvel, L. M (2014) "A theoretical exploration of the impact of packet loss on network intrusion detection, In Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), 2014 15th IEEE/ACIS International Conference on. IEEE, (Jun/Jul. 2014), pp1-6, DOI=http://doi.org/10.1109/SNPD.2014.6888699.

[29] Cisco (2015) Catalyst 3560 Switch Software Configuration Guide, Cisco IOS Release 15.0(2)SE and Later,[Online], Available: http://www.cisco.com/c/en/us/td/docs/switches/lan/catalyst3560/software/release/15-0_2_se/configuration/guide/scg3560.html. [Accessed: 11-April-2015].

[30] Szigeti, T., Hattingh, C., Barton, R., & "Briley, K (2013) "End-To-End QoS Network Design: Quality of Service for Rich-Media and Cloud Networks", Pearson Education.

[31] Wallace, K (2011) Implementing Cisco Unified Communications Voice Over IP and QoS (CVOICE) Foundation Learning Guide: (CCNP Voice CVoice 642-437), Cisco Press.
[32] Americas Headquarters, C (2010) Catalyst 3560 Switch Software Configuration Guide, Cisco IOS Release 12.2(55)E.
[33] Parker, D. B (1981) Computer Security Management, Reston Publishing Company, Reston, VA.
[34] Agrawal, D., Das, S., & El Abbadi, A (2011) "Big data and cloud computing: current state and future opportunities", In Proceedings of the 14th International Conference on Extending Database Technology (March. 2011), ACM, pp530-533, DOI= http://dx.doi.org/ 10.1145/1951365.1951432.
[35] Patel, A., Taghavi, M., Bakhtiyari, K., & Júnior, J. C (2013) "An intrusion detection and prevention system in cloud computing: a systematic review", Journal of Network and Computer Applications, 36, 1 (Jan. 2013), pp25-41,DOI= http://dx.doi.org/ 10.1016/j.jnca.2012.08.007.

AUTHORS

Waleed Bul'ajoul, a doctoral research and lecturer assistant,at Coventry University, UK, with background in networking and security especially in NIDPS, Network configuration, Quality of services and parallel technologies.

Anne James is Professor of Data Systems Architecture at Coventry University, UK. She obtained her BSc degree at Aston University, UK in 1980 and her PhD at the University of Wolverhampton UK in 1986. The research interests of Professor James are in the general area of creating distributed systems to meet new and unusual data and information challenges.

Dr Siraj Shaikh, is a Reader in Cyber Security at the Centre for Mobility and Transport at Coventry University, UK .He has been involved in research, development and evaluation of large-scale distributed secure systems for over fifteen years. His main research interest lies in systems security, essentially at the intersection of cyber security, systems engineering and traditional computer science. He has addressed a range of problem domains including monitoring of insider and stealthy attacks, automotive cybersecurity, rail safety, and software assurance.

Dr Mandeep Pannu is an academic at Kwantlen Polytechnic University Canada, She obtained her MSc and PhD degrees at Coventry University, UK. Her research interests are in information retrieval and computer security