TECH
mahindra

Whitepaper

Performance Optimization for Next-Generation Firewall (NGFW)

Contents

Abstract

Next generation firewalls (NGFW) provide advanced security functions, diving into the application layer and performing a thorough analysis to prevent malware. NGFWs decrypt traffic, performing deep packet inspection to compare against known patterns and artificial intelligence (AI) powered pattern matching. These functions are quite demanding, and if not properly deployed or configured, would overwhelm the compute resources and quickly become inefficient in delivering the optimal throughput while also possibly racking up costs to operate. In this paper, we demonstrate the effective deployment of NGFW on Arm Neoverse powered 80 core Ampere® Altra® processor to achieve greater than 200 Gbps throughput while utilizing only 48 cores of CPU, leaving the additional cores to be available for other tasks.

Key Takeaways

- We analyze NGFW functions running on Arm cores and provide a guide on how to optimize for high throughput while scaling to multiple cores.
- We show how to achieve greater than 200 Gbps throughput by scaling Vector Packet Processing (VPP) and Snort instances across multiple Arm Neoverse-N1-based cores.
- We recommend best practices and optimizations that are publicly available on Gitlab repositories.

While we demonstrate the effective deployment of NGFW, this is just one workload out of many software-defined networking workloads that can see similar benefits, be it performance, throughput, total cost of operation, or watt. With proper fine-tuning and configurations, one can realize these benefits across various workloads ranging from databases, CI/CD tools, web servers, ML, AI, video streaming, and many more.

Introduction

In this paper, we will walk you through the steps to support over 200 Gbps of throughput traffic with an elastic virtual NGFW by scaling VPP and Snort instances on an Arm Neoverse-N1-based processor. We use simulated real-time application data and follow the industry accepted TRex test framework. We will review the architecture and present the optimal system setup and configuration required to achieve the highest performance.

The Arm Neoverse-N1 CPU is part of the Arm Neoverse family of Infrastructure CPUs. It is optimized for handling a wide range of cloud-native workloads at world-class levels of performance, efficiency, and compute density. It adds up to a more flexible, scalable, competitive infrastructure for cloud providers, carriers, developers, and customers.

Software System Overview

Our test environment consists of 2 systems: a traffic generator (TG) and a device under test (DUT). DUT comprises the following software modules:

Ubuntu Linux OS	VPP routing SW	Data plane development kit (DPDK) NIC Poll mode driver	Snort intrusion prevention system (IPS)

We use the Trex application as a traffic generator.

Figure 1 displays the functional view of the NGFW. The internet traffic is generated from TRex and fed to the DUT (External.net). The DPDK poll mode driver pulls the traffic from the NIC, and the packets undergo Ethernet, IP, and stateful access control list (ACL) packet processing. The packet is further classified in the Snort plugin of VPP and is relatively load balanced across multiple Snort instances through the lockless data acquisition (DAQ) framework. Deep packet inspection occurs in the Snort application to detect and filter malicious packets. All valid packets are sent back through DAQ, DPDK, and eventually the internal-facing NIC and TRex.

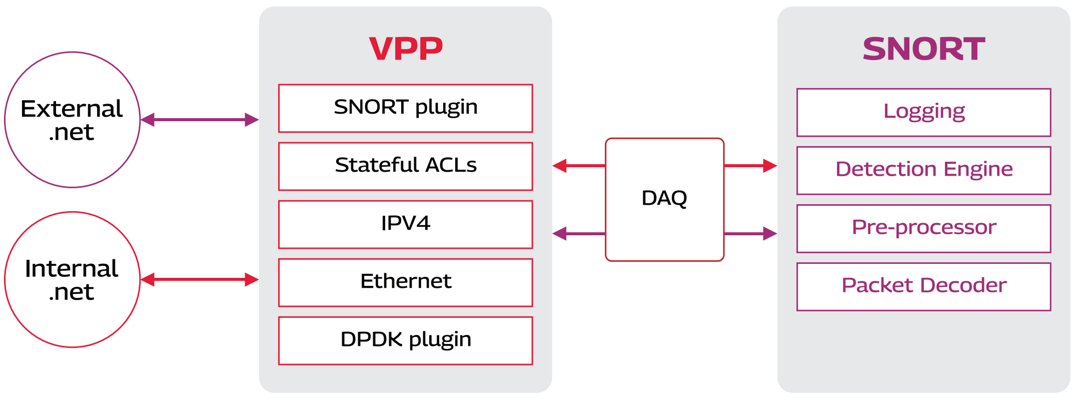

Figure 1: Functional overview of NGFW

All the metrics are compared with decode and detect-max IPS rules, where decode determines the underlying protocol like Ethernet, IP, or TCP in the packet, and detect-max module is related to deep packet inspection (DPI) of application layer data and its IPS mechanism.

NGFW Software Architecture

We have considered VPP and Snort open-source applications because they allow high-speed packet processing methods. We have used multiple VPP threads, called worker VPP (WV) and multiple Snort instances (SI). Due to its flexibility in using different numbers of VPP and Snort cores, it allows users to set VPP and Snort instances according to their requirements. Since VPP and Snort operate with two separate cores, we have used DAQ as an interface for packet processing between VPP and Snort due to its lockless architecture. The load balancing across VPP and Snort is done using the 5-tuple hashing method, which allows the effective distribution of packets across different Snort instances. Packet flow across DUT is shown in Figure 2.

After receiving the traffic, it is loaded across different worker VPPs, and each worker VPP can send traffic across different Snort instances. In stateful case, all packets from the same flow are processed from specific worker VPP and Snort instance.

Several fine-tuning methods and system grub changes are done to achieve better performance. Details can be found in Appendix section (6 & 7).

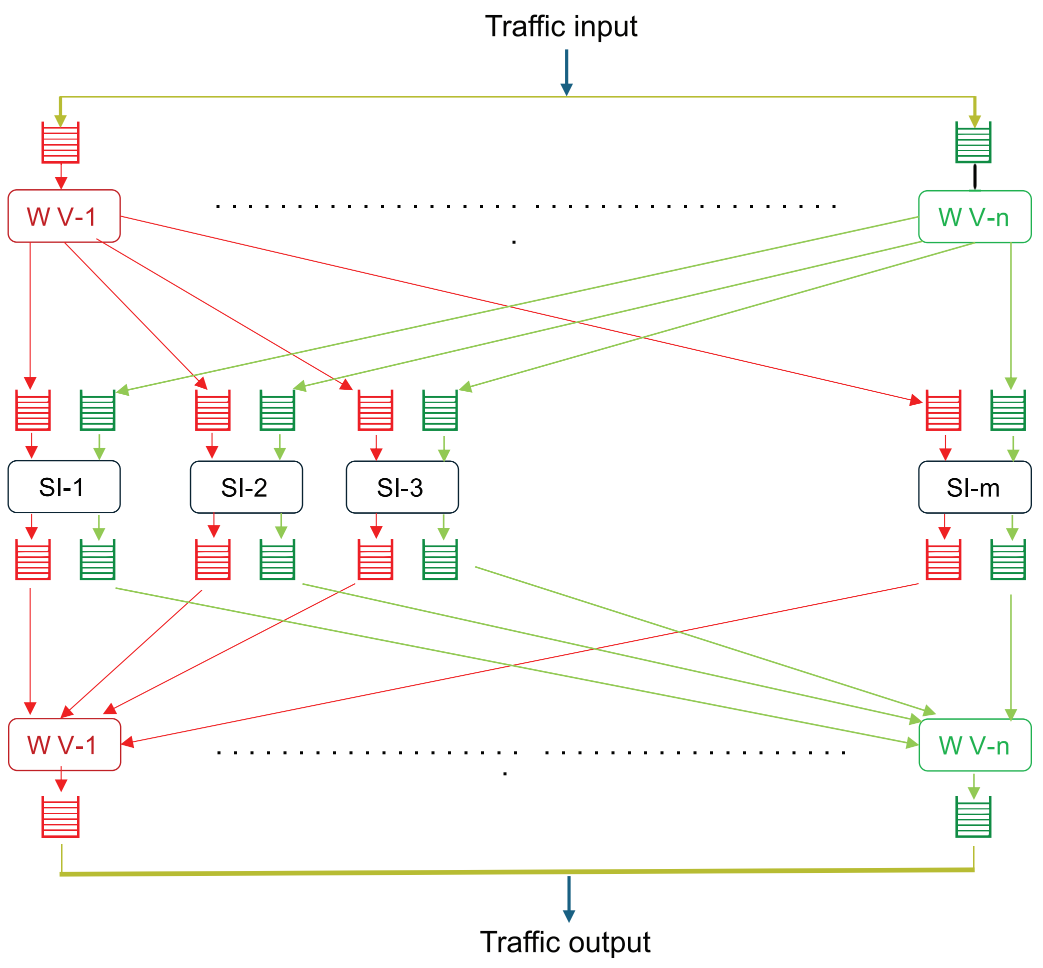

Figure 2: NGFW software architecture

Benchmarking Methodology

We consider the following parameters while benchmarking the test results:

Test Setup: Two-way topology consisting of TG and DUT connected via high-capacity Mellanox ConnectX-6 network cards. We have used Trex application as a traffic generator on TG, and VPP and Snort on DUT as open-source applications. The experiment has been conducted with two different types of Snort IPS rules (decode and detect-max), which aligns with the expectations of NGFWs. In the detect-max rule set, we have 33279 rules related to packet inspection for application layer protocols. No specific rules are set for decode, where all packets will be analyzed for underlying protocols like Ethernet, IP, and TCP. All the tests are done by varying VPP and Snort instance configurations and verifying different traffic patterns generated by the Trex application. All the metrics are compared with decode and detect-max rules. We used the following VPP and Snort instance numbers for each test case; these numbers are optimal pairs based on multiple tests. In decode, we used 12 VPP and 36 to 40 Snort instances, while in detect-max, we used 12 VPP and 12 to 13 Snort instances. Details of the lab setup and different test cases are added in the user guide; refer to Appendix section (13).

Results

Throughput

Throughput is the amount of data transferred in Gbps, which we measure and compare against different packet sizes. It is considered one of the important metrics in large data centers where vast amounts of data need to be handled in an effective way. Figure 3 provides throughput data as we scale the number of Snort instances, and Figure 4 provides throughput data for decode and detect-max rules for different types of traffic.

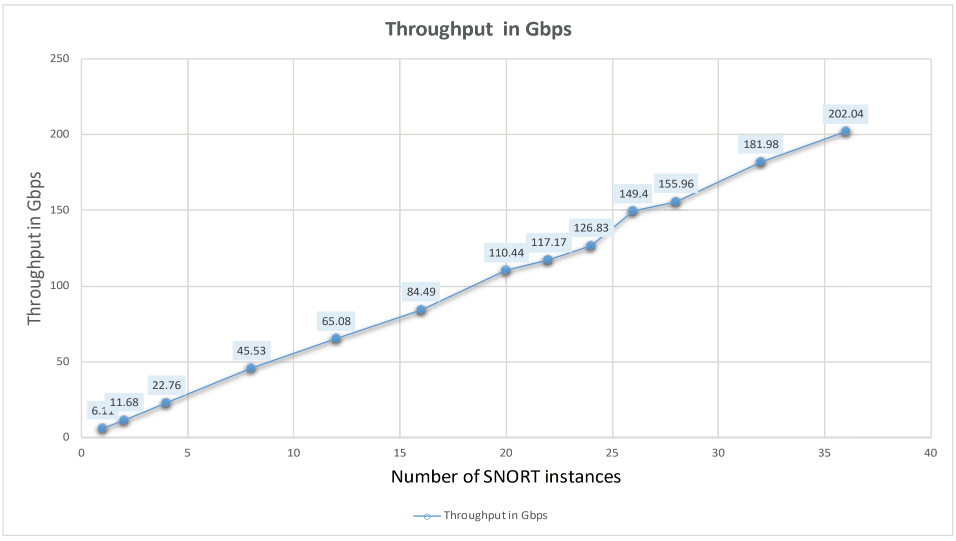

Figure 3: Throughput results across multiple SNORT instances

Here, we present the maximum throughput achieved in our system with different types of traffic and IPS rules.

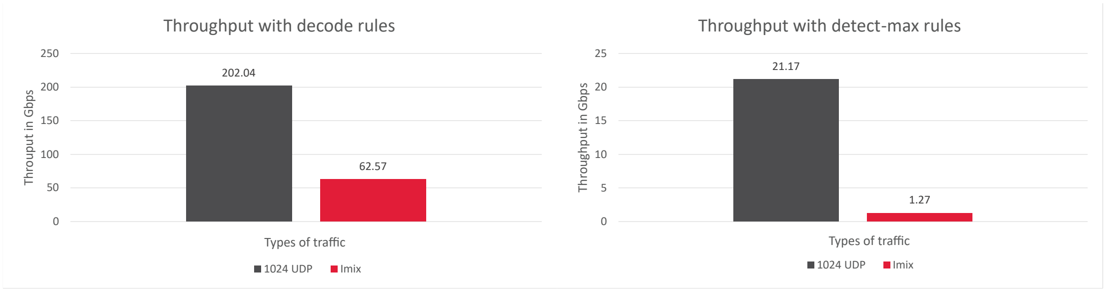

Figure 4: Throughput (in Gbps) with different packet types

Packets Per Second (PPS)

Packet rate is the number of packets per second for different packet sizes. Packet size and rate are two crucial parameters that must be considered when evaluating and analyzing the performance of networks. Figure 5 highlights the maximum packets per second achieved in our system.

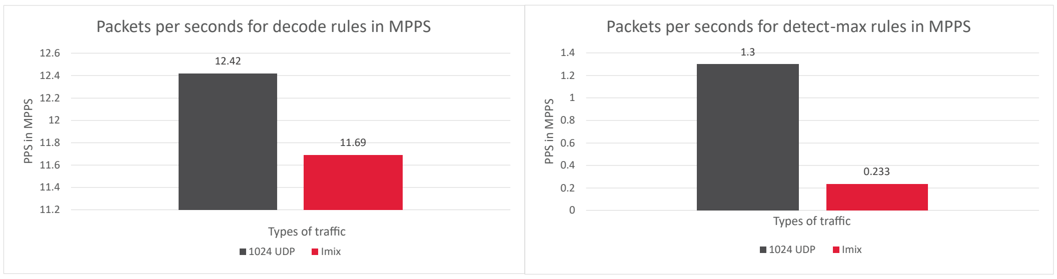

Figure 5: PPS (in MPPS) comparing with different packet types

Concurrent Connections

A concurrent connection is a session that happens simultaneously with another event that is already running. We can have multiple events running at the same time, each with a different user. This parameter also tells how many sessions/requests can be handled per second when handling real-time applications. Figure 6 showcases the maximum concurrent connections achieved in our system.

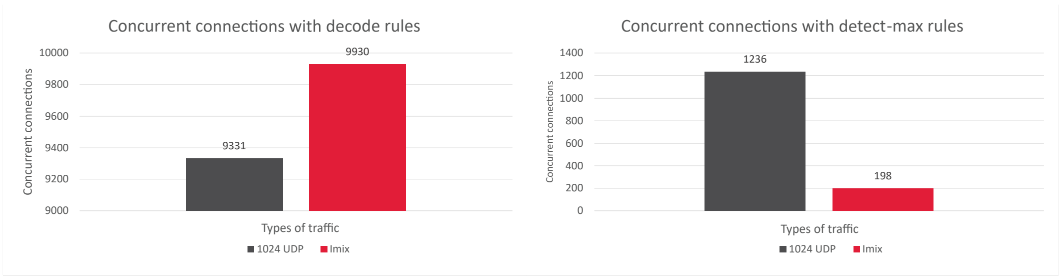

Figure 6: Concurrent connections with different packet types

Memory Utilization

Memory utilization is amount of the memory used versus memory available at a given moment. This is an important parameter for monitoring physical memory usage and identifying memory overflow conditions. Memory utilization is always proportional to the amount of traffic the system processes. Hence, while designing any system, this data helps in making decisions on required memory. Figure 7 is a view of the memory utilization with different types of traffic through our system.

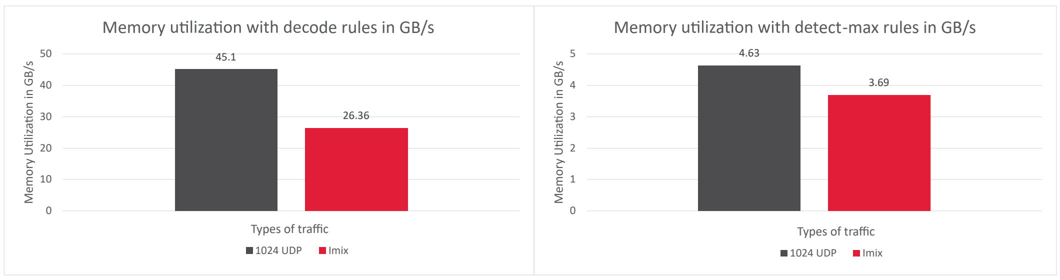

Figure 7: Memory utilization

Conclusion

The traffic levels generated in real-world critical applications and services are increasing day by day. A high-performance system that can handle more traffic is needed. This can lead to high investment and maintenance costs. In this project, we enhance the Arm Neoverse system's capabilities to handle more traffic. We provide clear guidance on how to scale VPP and Snort instances to achieve over 200Gbps traffic with 48 Arm Neoverse-N1 cores. We provide KPIs as well as the code needed to reproduce these results.

About the Author(s)

Manjunatha I

Module Lead, Tech Mahindra

Manjunatha holds a Master's degree in systems analysis from NITK Surathkal. He is currently the module lead at Tech Mahindra and has 16+ years of experience in different domains like networking, data plane and control plane stacks, system architect, data science and others.

E Lakshmi Prabha

Tech Lead, Tech Mahindra

Lakshmi Prabha holds an Engineering degree in electronics and works as a test lead at Tech Mahindra. She has nine years of professional experience in testing in the networking domain with extensive knowledge in automation.

Geetha Lakha

Practice Head, Optical, Access and Networking Division, Tech Mahindra

Geetha Lakha has an Engineering degree in electronics and communication and a management certification from IIT, Bombay. She has clocked over 20 years in the corporate world. She heads the networking and optical domain practice in Tech Mahindra and has delivered projects for TEM/OEM customers like Arm, Intel, Cisco, Infinera, Nokia, Juniper, Arris, F5, Netscout, and others.

Matthew Dirba

Principal Performance Engineer, Arm

Matthew Dirba graduated in 2002 from Texas A&M University with a B.S. in Computer Engineering and currently works as a principal performance engineer in the infrastructure line of business at Arm. Expertise in network/server performance and software development.

Appendix

1. Block Diagram and Test Setup

The complete setup consists of TG and DUT. Both devices are connected via 100 Gig Mellanox ConnectX-6 network cards. Traffic is generated from Trex and sent to DUT, where packets are processed over VPP and Snort applications.

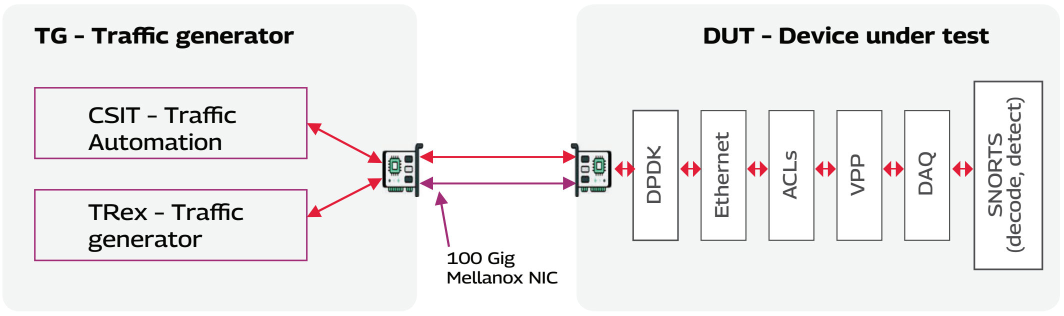

Figure 8: Block diagram

The lab setup details, application installation steps, system configuration steps, TG-DUT configuration steps, running and testing traffic data between TG and DUT, and KPI measurements are discussed in the NGFW user guide listed in Appendix section (13).

2. Traffic Profile Overview

- Traffic Generation Source: Trex (as Wireshark capture in ".pcap" format)
- Number of Flows: Total number of flows are based on the client and server, IP ranges are configured in the Trex test suite
- Number of Packets: Total number of flows generated by Trex * number of packets in Wireshark capture
- Average Packet Size: Tested with different packet sizes (64B, 512B, 1024B, 1518B, and mixed)

3. VPP Startup Configuration

Following is the VPP startup file and the corresponding command lines needed to bring up VPP. The command lines are used to define addresses and neighbors for the Home and External Network.

```
unix
{
  log /tmp/vpe.log
  cli-listen /run/vpp/cli.sock
  cli-no-pager
  nodaemon
  full-coredump
}
memory
{
  main-heap-size 2G
  main-heap-page-size 1G
  default-hugepage-size 1G
}
statseg
{
  size 2G
  page-size 1G
  per-node-counters on
}
```

```
plugins
{
  plugin default
  {
    disable
  }
  plugin dpdk_plugin.so
  {
    enable
  }
  plugin perfmon_plugin.so
  {
    enable
  }
  plugin snort_plugin.so
  {
    enable
  }
}
cpu
```

```
{
  main-core 1
  corelist-workers 2,3,4,5,6,7,8,9,10,11,12,13
}
buffers { buffers-per-numa 1048576 }
dpdk
{
  dev 0001:01:00.0 { name eth0}
  dev 0000:01:00.0 { name eth1}
  log-level debug
```

```
  uio-driver vfio-pci
  dev default
  {
    num-rx-queues 12
    num-tx-queues 12
    num-rx-desc 1024
    num-tx-desc 1024
  }
  no-multi-seg
}
```

Figure 9: VPP startup.conf

4. VPP Cli Configuration

Following is the VPP cli configuration to route traffic from the interface and create Snort instances.

```
set interface ip address eth0
10.10.1.1/24
set interface ip address eth1
10.10.2.1/24
set interface state eth0 up
set interface state eth1 up
ip route add 16.0.0.0/8 via
10.10.1.2
ip route add 48.0.0.0/8 via
10.10.2.2

snort create-instance name snort1
queue-size 8192
```

```
snort attach instance snort1 inter-
face eth0
snort attach instance snort1 inter-
face eth1
. . .
snort create-instance name snort(n)
queue-size 8192
snort attach instance snort(n)
interface eth0
snort attach instance snort(n)
interface eth1

#where n = number of snort instances
```

Figure 10: Configuring VPP interfaces and starting SNORT instances

5. SNORT Commands

Snort invoked with the following command line:

Decode.lua:

```
snort --c $SNORT_LUA_DIR/decode.lua
--lua "search_engine.search_method =
'hyperscan'" --snaplen 9000
--daq-dir=$DAQ_DIR --daq vpp
--daq-var debug -i snort(n) -k none
-Q --warn-conf-strict
```

Detect_Max.lua:

```
snort --c $SNORT_LUA_DIR/detect-max-
.lua --lua "search_engine.search_-
method = 'hyperscan'" --snaplen 9000
--daq-dir=$DAQ_DIR --daq vpp
--daq-var debug -i snort(n) -k none
-Q --warn-conf-strict
```

Figure 11: SNORT commands for different types of IPS rules

6. GRUB Commands

System grub provides essential parameters. By updating these, we can improve hardware performance. Here are a few of these parameters:

CPU Isolation: It is one of the kernel boot parameters; it helps isolate certain cores/CPUs for specific tasks. From Figure-12, the parameter – “isolcpus” is used to set CPU isolation.

HugePage Memory: This parameter is used as one of the memory management techniques in modern computer systems to improve performance by using larger memory blocks than the default page size. It helps reduce the pressure on the Translation Lookaside Buffer (TLB) and lowers the overhead of managing memory in systems with large amounts of RAM. From Figure-12, parameters – “hugepagesz” and “hugepages” are used to set hugepage memory.

There are no specific numbers that VPP or Snort provides to set the parameter: hugepage memory. This number is derived based on different test trials.

```
GRUB_CMDLINE_LINUX="iommu.passthrough=1 nohz_-
full=2-13,18-21,26-29,34-37,42-45 rcu_nocbs=2-13,18-21,26-29,34-37,42-45
isolcpus=nohz,domain,managed_irq,2-13,18-21,26-29,34-37,42-45 irqaffini-
ty=0-1,14-17,22-25,30-33,38-41,46-79 default_hugepagesz=1G hugepagesz=1G
hugepages=64 isolcpus=nohz,domain,2-13,18-21,26-29,34-37,42-45 transpar-
ent_hugepage=madvise cpufreq.off=1"
```

Figure 12: GRUB changes

7. Fine-Tuning Methods

Hardware-side/system-side configuration

- Increased Hugepages and GRUB in DUT to support huge memory
- Cache stashing, CPU clock frequency tuning, CPU isolation

Software improvements

- Enhancement to SNORT plugin within VPP to support multiple SNORT instances
- Enhancement to SNORT plugin support load balancing using the HASH method to increase traffic sharing across multiple snorts from multiple worker VPPs
- Tested with different VPP and SNORT instances and achieved different numbers for different types of traffic and IPS to achieve better throughput. Note: All the tests are based on the Trex test framework as a traffic generator.
- VPP startup.conf tuning: Updated different parameters like dpdk and other parameters to achieve better results.

8. DUT System Configuration

Metric	Units	DUT
Vendor ID		Arm
System		Supermicro
Chassis		Supermicro Main Server Chassis
Architecture		aarch64
CPU Model		Arm Neoverse-N1 (Ampere® Altra® Processor)
CPU Frequency	MHZ	3000
Sockets		1
Numa Nodes		1
Threads per Core		1
Cores per socket		80
Core Count		80
DRAM - Memory	GB	512 (16x32GB
Transparent Huge		DDR4 3200 MT/s)
Pages	kb	madvise
Hugepagesize		1048576
Ethernet Adapter		Mellanox-MT28908 Family [ConnectX-6]
OS		Ubuntu - 20.04 LTS
kernel		5.17.1

9. UDP Traffic Results

Metrics	Units	Values				
Average Packet Size	Bytes	64	512	1024	1518	Imix Traffic
Throughput	Gbps	12.62	100.9	202.04	225	62.5
Packet Per second	MPPs	12.33	12.48	12.42	9.2	11.62
Concurrent Connections		11709	11708	9331	8634	9930
Number of VPP worker threads		12	12	12	12	12
Number of Snort Instances		36	36	36	36	36

10. SNORT Rule Sets

Attribute	Value
Name	Registered Rules
Source	https://www.snort.org/downloads#rules
Version	snortrules-snapshot-31110.tar.gz
Number of rules (detect-max)	33279
Number of rules (decode)	There are no specific rules for decode where all the packets will be analyzed for underlying protocols like Ethernet, IP, and TCP

11. DUT Software Configuration

Software	Version
VPP	v23.10
Snort	3.1.25.0
DAQ	3.0.9
libpcap	1.9.1-3
vectorscan	5.4.7
Snort Rules	Snort Registered Rules
Pcap	Captured from CSIT execution
Measurement Methodology	NDR (0%)
DAQ mode	Interupt
num-rx-queues	1 queue per VPP thread
num-tx-queues	1 queue per VPP thread
num-rx-desc	4096
num-tx-desc	4096
buffers-per-numa	1048576
DAQ queue-size	8192

12. Gitlab Reference for VPP/CSIT/Trex Changes

As part of this exercise, multiple changes were made to open-source applications like Trex, CSIT, and VPP to achieve the required target. Following are the private gitlab repos for Trex, CSIT, and VPP. Users can find details of the changes within these repos.

Trex
techmarm1 / techmarm_trex-core · GitLab

CSIT
techmarm1 / techmarm_csit · GitLab

VPP
techmarm1 / techmarm_vpp · GitLab

13. Arm Performance Measurement with VPP and SNORT

To measure Arm performance with VPP and Snort, follow the user guide

https://gitlab.com/techmarm1/techm_ngfw_user_guide

References:

PMU study reference link:

https://armkeil.blob.core.windows.net/developer/Files/pdf/white-paper/neoverse-n1-core-performance-v2.pdf

TECH mahindra