Technology Guide

Accelerate Snort Performance with Hyperscan and Intel Xeon Processors on Public Clouds

Authors

Xiang Wang

Oisin O'Halloran

Subhiksha Ravisundar

Wojciech Andralojc

Declan Doherty

Heqing Zhu

Michihiro Koyama

1 Introduction

Snort is one of the most popular open-source Intrusion Detection/Prevention Systems (IDS/IPS). It continuously monitors network traffic to detect malicious activities and prevents vulnerability exploits by taking well-defined actions. Snort is a compute intensive workload requiring heavy string and regular expression (regex) matching based on pre-configured rulesets. This major performance overhead in Snort could be significantly reduced by Hyperscan software library optimized by Intel. Hyperscan is a high-performance string and regex matching library. It is optimized to use single instruction multiple data (SIMD) technology on Intel Processors to accelerate the matching performance. Hyperscan's integration with Snort could boost the performance by upto 3x[1] on an Intel processor-based server.

In enterprise networking, Snort is usually integrated with next-generation firewall and installed in network security appliances. However, there is a strong trend to migrate network security functions to cloud-based computing and network services. This brings the benefits of scalability and elasticity to prevent ever-increasing cybersecurity threats. It also frees IT operators from maintaining hardware equipment and adds the flexibility on usage or billing model. Most network security vendors now offer the option of delivering their network security software (such as IDS/IPS) as a service from public clouds. Given the offering of so many compute instance types in the cloud, it's not easy to choose proper ones to get best performance under budget. Since Hyperscan uses SIMD technology (such as Intel AVX-512 instruction-sets) and its performance is sensitive to the processor used by cloud instance types, find the right instance type is critical for Snort performance in cloud deployment.

This paper introduces Snort, optimized by Hyperscan, performance benchmarking on various compute instance types on major public cloud service provides. The benchmarking is fully automated with Intel Multi Cloud Networking Tool (MCNAT). This provides a solid reference on evaluating Snort performance and picking the optimal instance type for Snort on public cloud. Please contact authors to learn more about MCNAT.

This document is part of the Network Transformation Experience Kits.

[1] See Section 4 Performance Evaluation

Table of Contents

Figures

Tables

Document Revision History

Revision	Date	Description
001	March 2023	Initial release.

1.1 Terminology

Table 1. Terminology

Abbreviation	Description
DFA	Deterministic Finite Automaton
DPI	Deep Packet Inspection
HTTP	Hypertext Transfer Protocol
IDS/IPS	Intrusion Detection and Prevention System
ISA	Instruction Set Architecture
NFA	Non-deterministic Finite Automaton
PCAP	Packet Capture
PCRE	Perl Compatible Regular Expressions Library
Regex	Regular Expression
SASE	Secure Access Service Edge
SIMD	Single Instruction Multiple Data Technology
TCP	Transmission Control Protocol
URI	Uniform Resource Identifier
WAF	Web Application Firewall

1.2 Reference Documentation

Table 2. Reference Documents

Reference	Source
Intel® Xeon® Scalable Platform Built for Most Sensitive Workloads	https://www.intc.com/news-events/press-releases/detail/1423/intel-xeon-scalable-platform-built-for-most-sensitive
Snort	https://www.snort.org/
Snort Talos Rules	https://www.snort.org/downloads#rules
Hyperscan	https://www.hyperscan.io/
Hyperscan and Snort Integration	https://www.intel.com/content/www/us/en/developer/articles/technical/hyperscan-and-snort-integration.html
Hyperscan: A Fast Multi-pattern Regex Matcher for Modern CPUs	https://www.usenix.org/conference/nsdi19/presentation/wang-xiang
Teddy: An Efficient SIMD-based Literal Matching Engine for Scalable Deep Packet Inspection	https://dl.acm.org/doi/10.1145/3472456.3473512
Optimize Azure Cloud Security with Intel Hyperscan	https://www.hyperscan.io/2020/09/28/optimize-azure-cloud-security-with-intel-hyperscan/
Intel® 64 and IA-32 Architectures Software Developer Manuals	https://www.intel.com/content/www/us/en/developer/articles/technical/intel-sdm.html
Intel® Intrinsics Guide	https://www.intel.com/content/www/us/en/docs/intrinsics-guide/index.html
Accelerating Suricata Throughput Performance Using Hyperscan Pattern-Matching Software	https://www.intel.com/content/dam/www/public/us/en/documents/solution-briefs/hyperscan-scalability-solution-brief.pdf
Suricata	https://suricata.io/
Hyperscan in Suricata: State of the Union	https://suricon.net/wp-content/uploads/2016/11/SuriCon2016_GeoffLangdale.pdf
Hyperscan in Rspamd	https://www.slideshare.net/VsevolodStakhov/rspamdhyperscan
Regex Set Scanning with Hyperscan and RE2::Set	https://www.hyperscan.io/2017/06/20/regex-set-scanning-hyperscan-re2set/
Hyperscan integration with Github	https://github.blog/2018-10-17-behind-the-scenes-of-github-token-scanning/

2 Overview

Snort is a popular open-source IDS/IPS from Cisco. It is a rule-based system to detect malicious traffic according to user configured rulesets. Cisco provides Snort Talos as an official ruleset for Snort. Figure 1 shows the overall architecture and workflow of Snort:

1. Network traffic first passes through packet decoder which parses incoming packets.
2. Preprocessor conducts preprocessing, including HTTP URI normalization, packet defragmentation, TCP flow reassembly, etc.
3. Detection engine uses preloaded rule database to inspect content to discover threats.
4. Logging and alerting system could trigger actions defined in the rule upon successful match and save logs.
5. Output modules save outputs in formats including log, database, XML, etc.

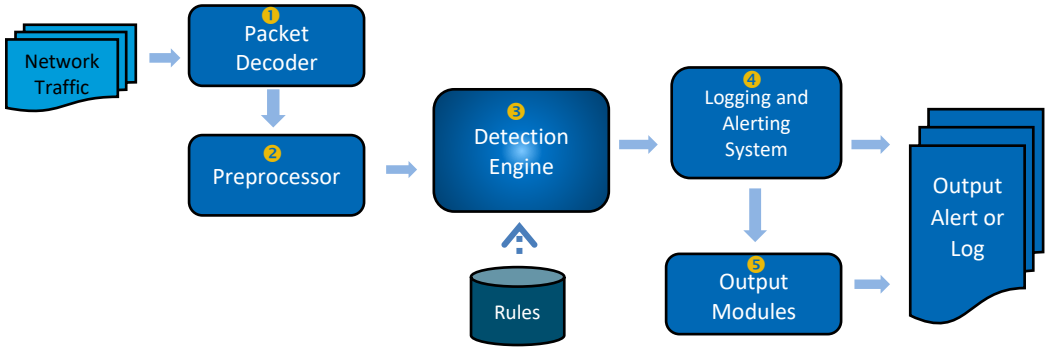

Figure 1. Snort Architecture and Workflow

Detection engine is the major performance bottleneck in the overall system as it is very compute intensive to match tens of thousands of rules with fixed string keywords and regex patterns. As a high-performance regex matching software library, Hyperscan can accelerate Snort matching performance significantly with Intel SIMD technologies, including Intel AVX-512 instruction-set, etc.

2.1 Intel SIMD & AVX-512 Instruction-sets

Intel SIMD technology boosts the data parallelism using advanced instruction set architecture. A SIMD instruction executes the same operation on multiple data in parallel. As shown in Figure 2, a scalar operation with general purpose register only computes up to 8 bytes in a time, while a SIMD operation is performed on multiple lanes of two SIMD registers independently, and the results are stored in the third register. Modern CPU supports a number of SIMD instructions that can work on specialized vector registers (SSE, AVX, etc.). The latest Intel AVX-512 instructions support up to 512-bit operations simultaneously.

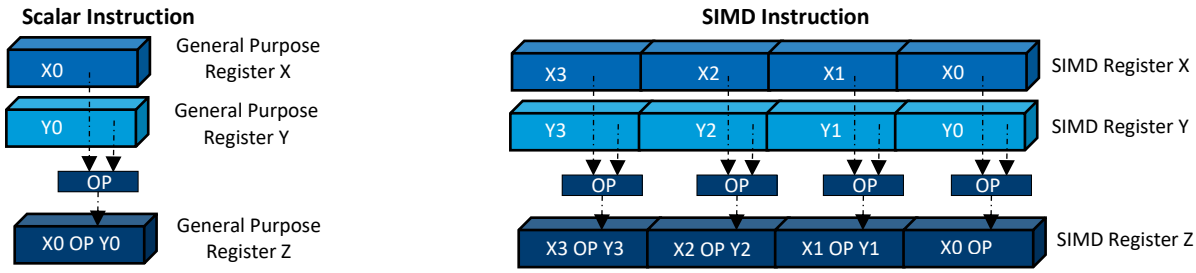

Figure 2. Scalar and SIMD Instructions

3rd Gen Intel® Xeon® Scalable processor adds a list of powerful AVX-512 instruction sets. For example, AVX-512 Vector Byte Manipulation Instructions (VBMI) allows byte-level permutes within 64-bytes which is useful to accelerate character class look up in regex matching. AVX-512 Vector AES instructions significantly boost symmetric crypto performance. Developers can refer to Intel SIMD intrinsic guide to embed SIMD intrinsic into their software to improve the performance. Figure 3 shows an example of using shuffle intrinsic (_mm512_permutexvar_epi8) for parallel table look up where src is the 64-byte table data and idx is the 64-byte index mask, each byte of dst contains the look up result on corresponding position specified by index mask in data mask.

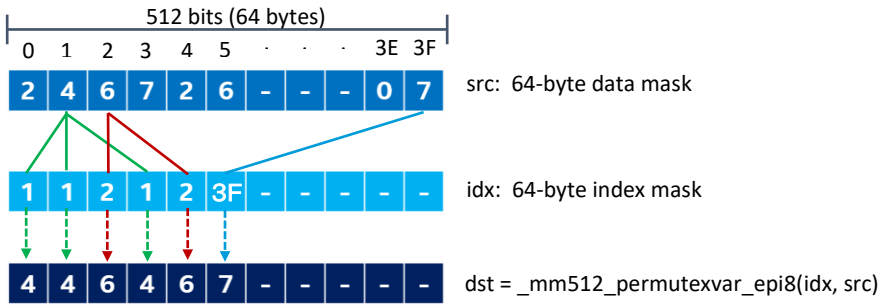

Figure 3. Byte Shuffle Intrinsic Usage for Table Look Up

2.2 Hyperscan

Hyperscan is a high-performance regular expression matching library optimized by Intel. It is a software implementation requiring Intel SSSE instruction-set support. Hyperscan delivers support from low-end Intel Atom® processors to Intel® Xeon® Scalable processors. It has a wide adoption with more than 40 customers and 37 open-source projects. It is a proven technology in network security use cases, including IDS/IPS, Deep Packet Inspection (DPI), Web Application Firewall (WAF), spam filtering, etc. The latest open-source version (BSD-3 license) of Hyperscan on Github is 5.4. Intel conducts continuous internal development and delivers new Hyperscan releases under Intel Proprietary License (IPL) beginning from 5.5 for interested customers. Please contact authors to learn more about getting new Hyperscan releases.

We show basic feature upgrades in below tables.

Table 3. Hyperscan 5.5 New Features

Feature	Description
Universal Database	It makes user care no more about platform differences in Hyperscan compilation stage when doing clustered deployment.
Multi-literal matcher ("Harry") for medium scale literals	It improves scanning performance by leveraging AVX-512 VBMI.
Character class matching model ("vshufti")	It improves scanning performance by leveraging AVX-512 VBMI.

Table 4. Hyperscan 5.6 New Features

Feature	Description
New API: redesign hs_compile_lit() and hs_compile_lit_multi() to process pure literal rule sets	It reunifies literal writing grammar with common regex.
New API: hs_scan_lit() for pure literal rule sets	It improves average performance over traditional runtime API for pure literal ruleset specifically.
Multi-literal matcher ("NeoHarry") for medium scale literals	It improves scanning performance by leveraging AVX-512 VBMI2.
Multi-literal matcher ("NeoTeddy") for small scale literals	It improves scanning performance by leveraging AVX-512 VBMI2.
64-state shuffle-based DFA engine ("Sheng64")	It Improves performance via associativity property.
Shuffle-based hybrid DFA engine("McSheng64")	It Improves performance via associativity property.
Shuffle-based hybrid DFA engine ("McSheng128")	It improves scanning performance by leveraging AVX-512 VBMI.
Improved coverage of "Universal Database" feature	It covers all different implementations of internal engines/matchers for all platforms. It also supports pure literal APIs.

2.3 Hyperscan Integration with Snort

Hyperscan has tight integration for both Snort 2.9 (patch provided from Intel) and Snort 3 (Cisco default integration). The main components to accelerate with Hyperscan includes:

- Single Literal Matching
  Users define specific literals to match in rules. Snort searches each of these literals in the packets using the Boyer-Moore algorithm. We replace this algorithm with Hyperscan to improve its matching performance.
- PCRE Matching
  Snort uses Perl Compatible Regular Expressions (PCRE) as its regular expression matching engine. Hyperscan is compatible with PCRE semantics, but it does not support a few backtracking and assertion syntaxes. To mitigate this

issue, Hyperscan includes a PCRE preprocessing function (PCRE prefiltering) to transform unsupported PCRE rules to supported ones. The matches produced by the original rules is a subset of the these generated by the transformed rules. This enables Hyperscan as a prefilter. If it doesn't produce matches, the actual rules will not generate any matches either. If there is a match, you can then use PCRE scan to confirm the match. Based on facts that Hyperscan gets better performance than PCRE and the matching rate in Snort is usually not very high, prefiltering with Hyperscan can avoid the excessive time cost of PCRE matching.

- Multiple Literal Matching
  Snort itself embraces a prefiltering-based matching design. By extracting keywords from each rule and using multiple literal matching to match them simultaneously, Snort can filter out rules that are not possible to match quickly. This saves the number of rules for deeper check and thus improves overall matching performance. Snort uses the Aho-Corasick algorithm for multiple literal matching. It is highly data dependent and does not scale well on a large number of rules due to space inefficiency and frequent cache misses. We use Hyperscan to replace Aho-Corasick algorithm to improve the performance significantly.

2.4 Multi Cloud Networking Automation Tool (MCNAT)

MCNAT is a software tool developed by Intel that provides automation for seamless networking workload deployment on public cloud and offers suggestions on selecting the best cloud instance based on performance and cost. It uses open-source software including Hashicorp Packer, Terraform and Ansible to build the automation pipeline.

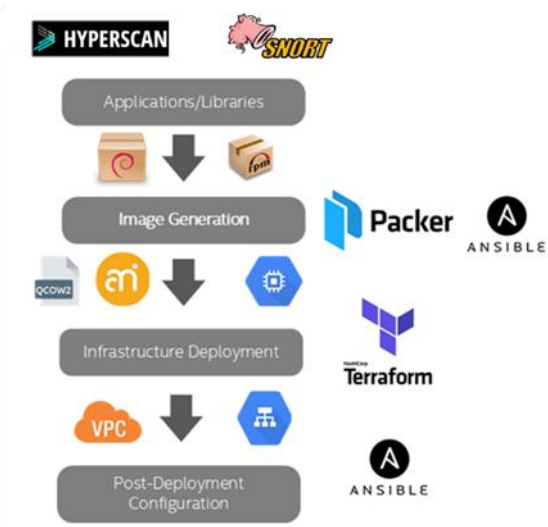

Figure 4. MCNAT Workflow

There are four stages in MCNAT as shown in Figure 4:

1) Applications/Libraries packaging
   Users need to package network applications/libraries with distribution package manager. For example, RPM could be used in many Linux distributions including Fedora, CentOS, etc.
2) Image generation
   Packer enables you to create identical VM images for multiple cloud platforms from a single HCL template. There are also Ansible playbooks with roles defining instructions on how to setup software inside image.
3) Infrastructure deployment
   Terraform launches completely provisioned and configured machine instances with Packer images in seconds on targeted public cloud. This will create cloud instances and all required networking and security resources for your application.
4) Post-deployment configuration
   It uses Ansible to configure software with data that is known after deployment (mac/ip addresses, etc.) and facilitates fully automated application deployment. This is followed by benchmarking and test data capturing.

MCNAT provides all automation setups for Snort benchmarking on major public clouds (AWS, Azure, Google Cloud, etc). We can easily use the captured performance data to pick most proper cloud instance types for Snort.

3 System Deployment

MCNAT uses the "1-node" module to deploy and test Snort, this module allows for the creation of a single cloud instance and the necessary networking setup. Once the Snort image is built and deployed. A full set of performance tests can run on the host machine and the results returned in CSV format.

3.1 MCNAT Configuration

MCNAT is configured through a series of profiles, each defining the variables and settings required for each instance. Each instance type has its own profile which can then be passed to the MCNAT CLI tool to deploy that specific instance type on a given cloud service provider (CSP). Example command line usage is shown below and in Table 5.

```
# ./mcnat.py --deploy -u user -c aws -s snort -p c6i-4xlarge
```

Table 5. MCNAT Command Line Usage

Option	Description
--deploy	Instructs the tool to create a new deployment
-u	Defines which users credentials to use
-c	CSP to create deployment on (AWS, GCP, Azure, etc)
-s	Scenario to deploy
-p	Profile to use

3.2 Setup Steps

The MCNAT Command line tool is setup to Build and Deploy Instances in a single step, this is shown below. Once the instance is deployed the post configuration steps create the necessary SSH configuration to allow the instance to be accessed.

```
# ./mcnat.py --deploy -c aws -u <#user_name>-s snort -p c6i-4xlarge-5-5
[INFO] mcnat deploy for snort.c6i-4xlarge-5-5 on aws at 2023-01-31 16:45:31.270113
[INFO] loading config..
[INFO] User config loaded!
[INFO] CSP config loaded!
[INFO] Scenario config loaded!
[INFO] Deploying packer-env scenario with default profile on aws ...
[INFO] Successfully deployed!
[INFO] Building snort images on aws ...
[INFO] Successfully built!
[INFO] Destroying packer-env scenario with default profile ...
[INFO] subprocess.run ./tfws_deploy.py --destroy -c aws -u oohallor -s packer-env -p default
[INFO] Successfully destroyed!
[INFO] Deploying snort scenario with c6i-4xlarge-5-5 profile on aws ...
[INFO] Successfully deployed!
[INFO] Configuring snort scenario with c6i-4xlarge-5-5 profile on aws ...
[INFO] Successfully deployed!
```

Once MCNAT has deployed the instance, all performance tests can be run by using the "run_snort.sh" script, this script runs snort with a given set of rules on a PCAP, while pinning a range of core counts, to gather a full set of performance numbers for the instance under test.

When the tests are completed all the data is formatted as a csv and returned to the user.

```
Model name:                      Intel(R) Xeon(R) Platinum 8375C CPU @ 2.90GHz
TIME_STAMP,CSP,INSTANCE,CORES,TPUT,TPUT_HSCAN
31-01-2023 18:42:01,aws,c5n-18xlarge,1C1T,128,405
31-01-2023 18:42:01,aws,c5n-18xlarge,1C2T,187,540
31-01-2023 18:42:01,aws,c5n-18xlarge,2C2T,256,810
31-01-2023 18:42:01,aws,c5n-18xlarge,2C4T,375,1080
31-01-2023 18:42:01,aws,c5n-18xlarge,4C4T,513,1620
31-01-2023 18:42:01,aws,c5n-18xlarge,4C8T,751,2160
31-01-2023 18:42:01,aws,c5n-18xlarge,8C8T,1026,3240
31-01-2023 18:42:01,aws,c5n-18xlarge,8C16T,1514,4321
```

4 Performance Evaluation

In this section, we will compare Snort performance on different cloud instances in public cloud including AWS, Azure, and Google Cloud. This gives guidance on finding the most suitable cloud instance type for Snort based on performance and cost. Below results on all three CSPs will include:

- Generation to generation performance gain among 1st, 2nd, 3rd Gen Intel® Xeon® Scalable processors.
- Snort performance on smaller instance types that host 16 vCPUs
- Comparing Snort performance with enabling default search engine (ac_bnfa) versus Hyperscan.
- Performance scaling with the number of vCPUs.
- Showcasing how processor core frequency plays an important role for compute bound workloads like Snort
- Performance gain with Hyper-Threading Technology enabled.

4.1 AWS Deployment

4.1.1 Instance Type List

C5n-4xlarge – 1st Gen Intel® Xeon® Platinum processor instance type with 16 vCPUs

C6i-4xlarge – 3rd Gen Intel® Xeon® Platinum processor instance type with 16 vCPUs

4.1.2 Results

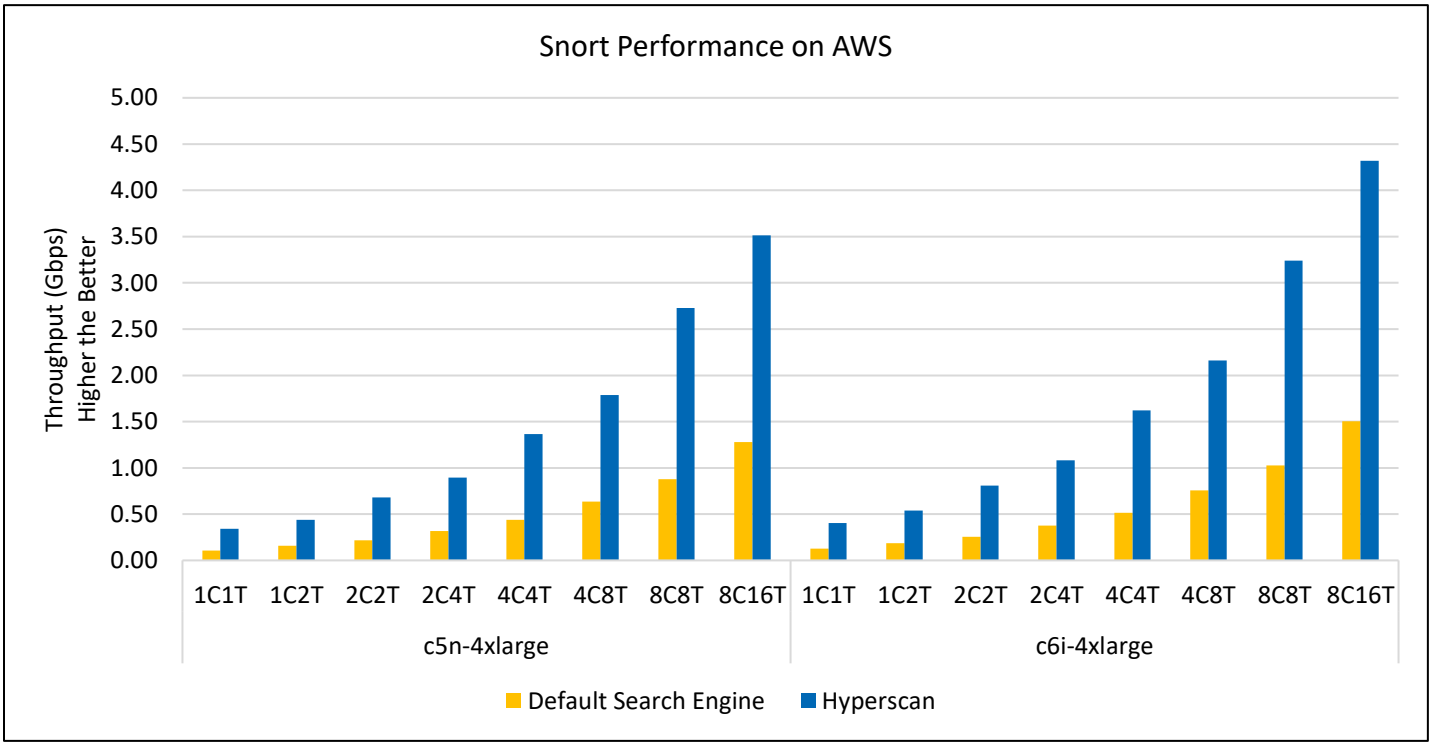

Figure 5. Snort Performance on AWS

- Comparison from 1st Gen to 3rd Gen Intel Xeon Scalable processor servers show a performance gain of up to 23%.
- Enabling Snort with Hyperscan in place of using the default search engine gives an additional gain of up to 3.1x on c6i instance types.
- Both instance types show very linear performance scaling with number of cores.
- Intel Xeon processor-based servers can use Hyper Threading Technology that gives an additional 33% performance gain.
- The operating frequency of c5n is 3.4Ghz and c6i is 3.5Ghz. For this workload core frequency is very linear to the performance.

4.2 Microsoft Azure Deployment

4.2.1 Instance Type List

Std-d16-v4 – 2nd Gen Intel Xeon Platinum processor instance type with 16 vCPUs

Std-d16-v5 – 3rd Gen Intel Xeon Platinum processor instance type with 16 vCPUs

4.2.2 Results

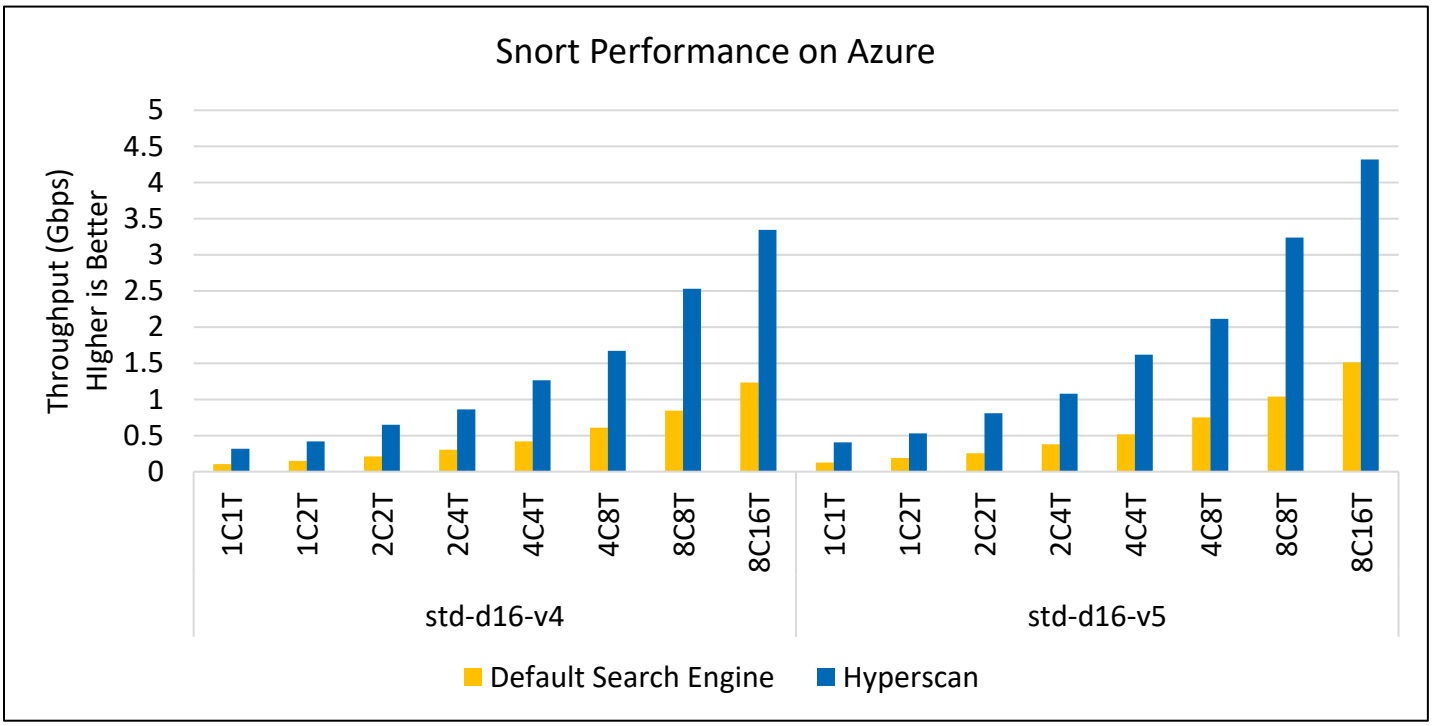

Figure 6. Snort Performance on Azure

- Comparison from 2nd Gen to 3rd Gen Intel Xeon Scalable processor-based servers show a performance gain of up to 30%.
- Enabling Snort with Hyperscan in place of using the default search engine gives an additional gain of up to 3.3x on std-v5 instances.
- Both instance types show very linear performance scaling with number of cores.
- Intel Xeon Scalable processor-based servers can use Hyper-Threading Technology that gives an additional 30% performance gain.

4.3 Google Cloud Deployment

4.3.1 Instance Type List

N1-std-16 – Intel® Xeon® E5 v3 processor instance type with 16 vCPUs

N2-std-16 – 3rd Gen Intel Xeon Platinum processor instance type with 16 vCPUs

4.3.2 Results

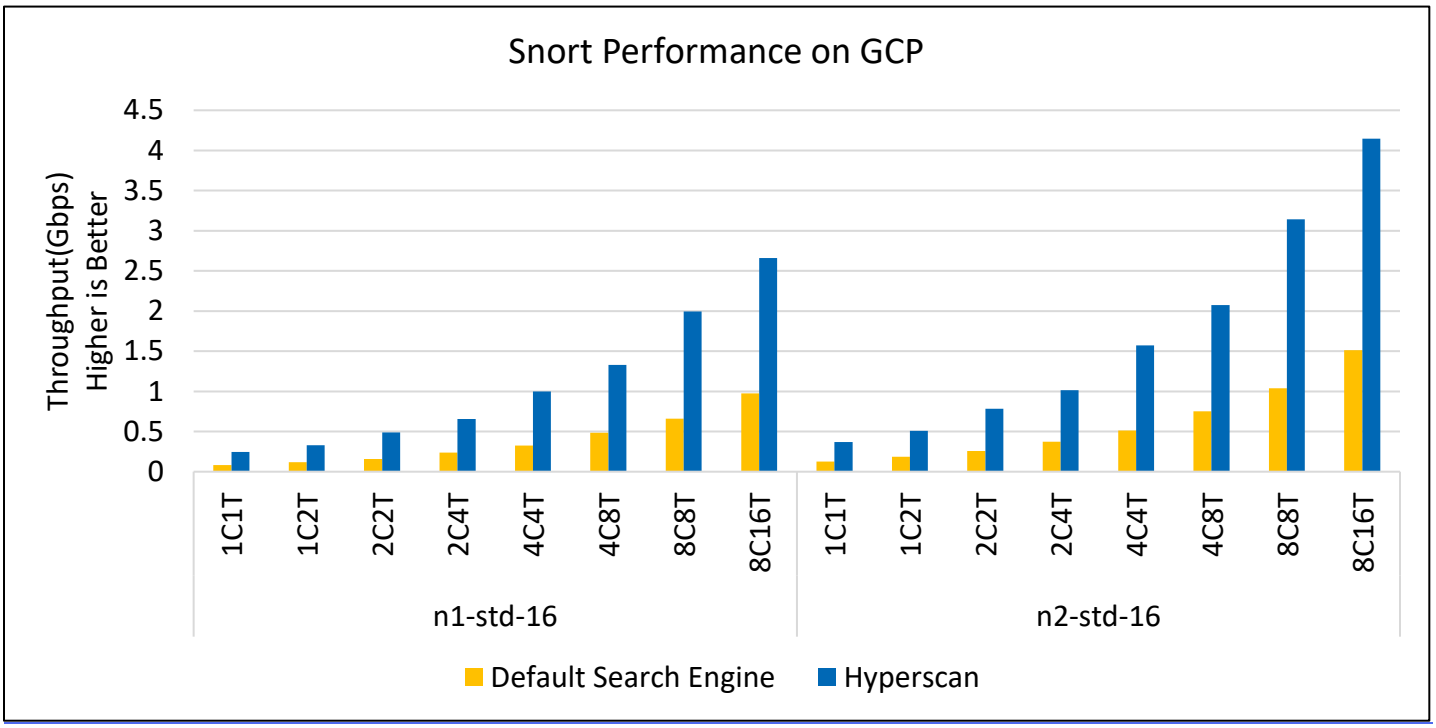

Figure 7. Snort Performance on GCP

- Comparison from Intel Xeon E5 v3 processor to 3rd Gen Intel Xeon Scalable processor-based servers shows a performance gain of up to 50%.
- Enabling Snort with Hyperscan versus using the default search engine gives an additional gain of up to 3x on n2-std-16 instances.
- Both instance types show very linear performance with core scaling.
- Intel Xeon Scalable processor-based servers can use Hyper-Threading Technology that gives an additional 32% performance gain on n2-std-16 instance types.

4.4 Throughput per Dollar Comparison

Table 6. Instance On-demand Hour Rates

Instance Type	On-demand hourly rate ($)
AWS - c5n-4xlarge	0.86
AWS - c6i-4xlarge	0.68
GCP - n1-std-16	0.76
GCP - n2-std-16	0.78
Azure - std-d16-v4	0.77
Azure - std-d16-v5	0.77

Table 6 details the on demand hourly rate for all the instances mentioned in this paper. The above was the on-demand rate at the time of publishing this paper and focuses on the US west coast. The On-demand hourly rate might vary with the region, availability, corporate accounts, and other factors.

Sites used to get the ON-demand Pricing for each instance type:

- https://aws.amazon.com/ec2/pricing/on-demand/
- https://azureprice.net/
- https://cloud.google.com/compute/vm-instance-pricing#general-purpose_machine_type_family

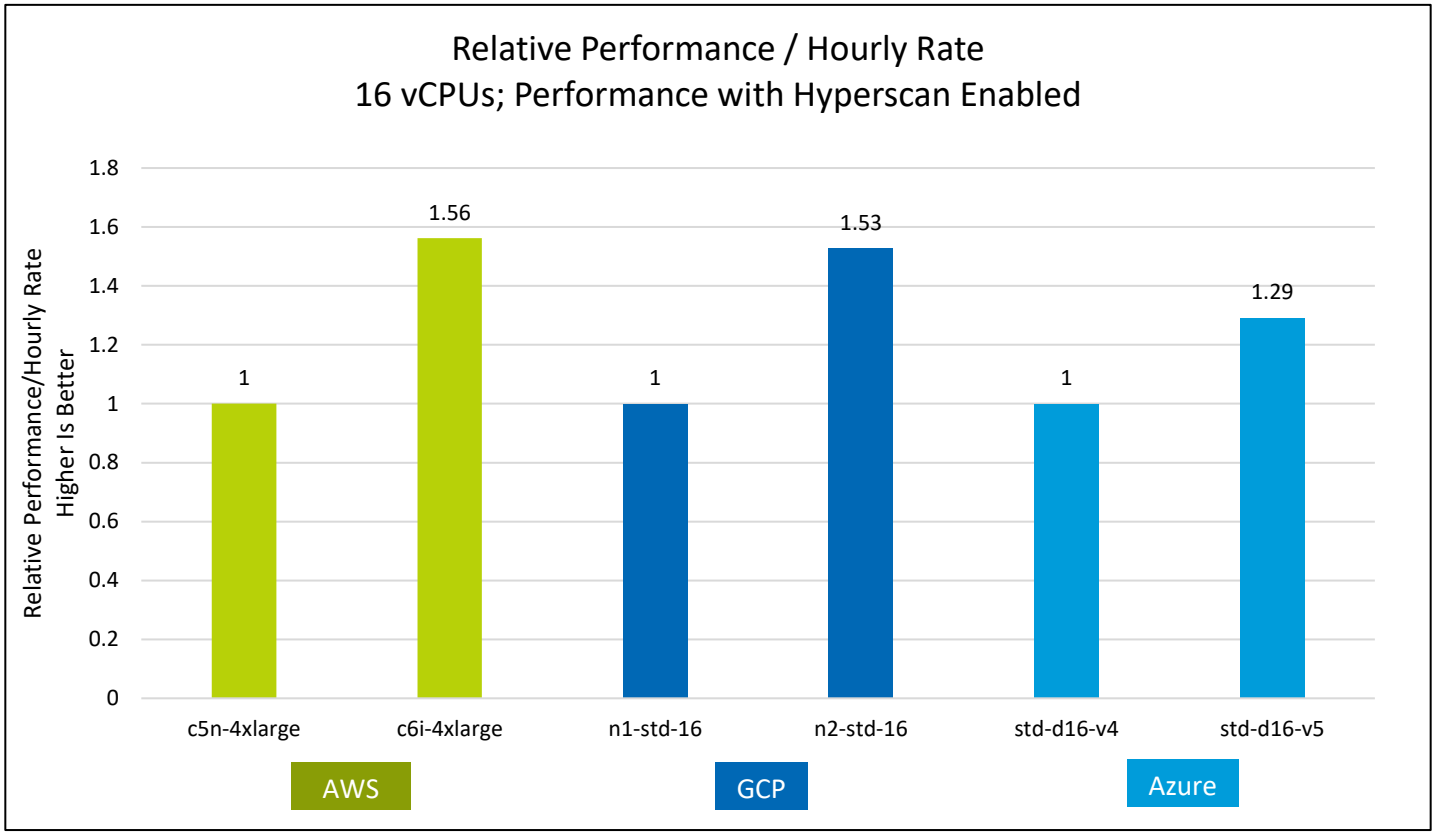

Figure 8. Relative Performance per Hourly Rate

Figure 8 compares relative performance per hour rate on all the instance types mentioned thus far. This data is also derived for the throughput achieved with 16 vCPUs on all the instance types. We compare instance types within the same public cloud and use the previous generation instance as the baseline. In summary, newer generation of instances powered by 3rd Gen Intel Xeon Scalable processor always achieve better performance/hour rate.

5 Summary

The trend of offering network security solutions (such as IDS/IPS) as services on public cloud delivers advantage including scalability, flexibility, ease of maintenance, etc. As more network security vendors are embracing this model, one main challenge for such network security vendors is to find the most suitable cloud instance type in terms of performance and cost given so many choices provided by public service providers. Different cloud instance types differ in computation power, network bandwidth, security features, cost, etc. These are key factors that impact the performance of network security workloads. We use Snort as the representative IDS/IPS solution and fully automate its deployment on multiple cloud instance types on different public clouds. By comparing performance numbers, we can conclude the latest processor compute capabilities (such as SIMD instructions) are critical to Snort as a compute intensive solution. Hyperscan boosts Snort performance by about 3x[2] on public clouds. This benchmarking gives solid references on cloud instance type selection for Snort and other similar network security workloads.

[2] See Section 4 Performance Evaluation

Appendix A Platform Configuration

Name	c5n-4x-large	c6i-4x-large
Time	Mon Dec 12 12:51:43 UTC 2022	Mon Dec 12 12:31:36 UTC 2022
System	Amazon EC2 c5n.4xlarge	Amazon EC2 c6i.4xlarge
Baseboard	Amazon EC2 Not Specified	Amazon EC2 Not Specified
Chassis	Amazon EC2 Other	Amazon EC2 Other
CPU Model	Intel® Xeon® Platinum 8124M CPU @ 3.00GHz	Intel® Xeon® Platinum 8375C CPU @ 2.90GHz
Microarchitecture	Sky Lake	Ice Lake
Sockets	1	1
Cores per Socket	8	8
Hyperthreading	Enabled	Enabled
CPUs	16	16
Intel Turbo Boost	Enabled	Enabled
Base Frequency	3.0GHz	2.9GHz
All-core Maximum Frequency	3.4GHz	3.5GHz
Maximum Frequency	3.5	3.5
NUMA Nodes	1	1
Prefetchers	L2 HW, L2 Adj., DCU HW, DCU IP	L2 HW, L2 Adj., DCU HW, DCU IP
Accelerators	QAT:0, DSA:0, IAA:0, DLB:0	QAT:0, DSA:0, IAA:0, DLB:0
Installed Memory	42GB (1x42GB DDR4 2666 MT/s [Unknown])	32GB (1x32GB DDR4 3200 MT/s [Unknown])
Hugepagesize	2048 kB	2048 kB
Transparent Huge Pages	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled
NIC	1x Elastic Network Adapter (ENA)	1x Elastic Network Adapter (ENA)
Disk	1x 50G Amazon Elastic Block Store	1x 50G Amazon Elastic Block Store
BIOS	1	1
Microcode	0x2006c0a	0xd000331
OS	Ubuntu 20.04.5 LTS	Ubuntu 20.04.5 LTS
Kernel	5.15.0-1026-aws	5.15.0-1026-aws
TDP		
Frequency Governor		
Frequency Driver		
Max C-State	9	9

Name	std-d16-v4	std-d16-v5
Time	Mon Dec 12 16:33:27 UTC 2022	Mon Dec 12 16:56:13 UTC 2022
System	Microsoft Corporation Virtual Machine	Microsoft Corporation Virtual Machine
Baseboard	Microsoft Corporation Virtual Machine	Microsoft Corporation Virtual Machine
Chassis	Microsoft Corporation Desktop	Microsoft Corporation Desktop
CPU Model	Intel® Xeon® Platinum 8272CL CPU @ 2.60GHz	Intel® Xeon® Platinum 8370C CPU @ 2.80GHz
Microarchitecture	Cascade Lake	Ice Lake
Sockets	1	1
Cores per Socket	8	8
Hyperthreading	Enabled	Enabled
CPUs	16	16
Intel Turbo Boost	Enabled	Enabled
Base Frequency	2.6GHz	2.8GHz
All-core Maximum Frequency	0.0GHz	0.0GHz
Maximum Frequency	0	2.8GHz
NUMA Nodes	1	1
Prefetchers	L2 HW, L2 Adj., DCU HW, DCU IP	L2 HW, L2 Adj., DCU HW, DCU IP
PPINs	0	0
Accelerators	QAT:0, DSA:0, IAA:0, DLB:0	QAT:0, DSA:0, IAA:0, DLB:0
Installed Memory	1GB (1x1GB Other Unknown []); 32767MB (1x32767MB Other Unknown []); 31745MB (1x31745MB Other Unknown [])	1GB (1x1GB Other Unknown []); 32767MB (1x32767MB Other Unknown []); 31745MB (1x31745MB Other Unknown [])
Hugepagesize	2048 kB	2048 kB
Transparent Huge Pages	always	always
Automatic NUMA Balancing	Disabled	Disabled
NIC	No data found.	1x MT27800 Family [ConnectX-5 Virtual Function]
Disk	1x 30G Virtual_Disk, 1x 628K Virtual_CD	1x 30G Virtual_Disk, 1x 628K Virtual_CD
BIOS	90008	90008
Microcode	0xffffffff	0xffffffff
OS	Ubuntu 20.04.5 LTS	Ubuntu 20.04.5 LTS
Kernel	5.15.0-1021-azure	5.15.0-1021-azure
TDP		
Frequency Governor	Performance	Performance
Frequency Driver		intel_cpufreq
Max C-State	9	9

Name	n1-std-16	n2-std-16
Time	Fri Feb 10 17:24:05 UTC 2023	Fri Dec 9 15:54:01 UTC 2022
System	Google Google Compute Engine	Google Google Compute Engine
Baseboard	Google Google Compute Engine	Google Google Compute Engine
Chassis	Google Other	Google Other
CPU Model	Intel® Xeon® CPU @ 2.30GHz	Intel® Xeon® CPU @ 2.60GHz
Microarchitecture	Haswell	Ice Lake
Sockets	1	1
Cores per Socket	8	8
Hyperthreading	Enabled	Enabled
CPUs	16	16
Intel Turbo Boost	Enabled	Enabled
Base Frequency	2.0GHz	2.0GHz
All-core Maximum Frequency	3.4GHz	3.5GHz
Maximum Frequency	2000 MHz	2000 MHz
NUMA Nodes	1	1
Accelerators	QAT:0, DSA:0, IAA:0, DLB:0	QAT:0, DSA:0, IAA:0, DLB:0
Installed Memory	48GB (3x16GB RAM []); 12GB (1x12GB RAM [])	64GB (4x16GB RAM [])
Hugepagesize	2048 kB	2048 kB
Transparent Huge Pages	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled
NIC	1x device	1x device
Disk	1x 20G PersistentDisk	1x 20G PersistentDisk
BIOS	Google	Google
Microcode	0xffffffff	0xffffffff
OS	Ubuntu 20.04.5 LTS	Ubuntu 20.04.5 LTS
Kernel	5.15.0-1027-gcp	5.15.0-1025-gcp
TDP		
Power & Perf Policy		
Frequency Governor		
Frequency Driver		
Max C-State	9	9

Appendix B Snort Software Configuration and Command Line

Software Configuration	Software version	Location
Host OS	Ubuntu 20.04.1	https://ubuntu.com/
Kernel	5.15.0	https://www.kernel.org/
Hyperscan	5.5.0	https://github.com/intel/hyperscan
DAQ	3.0.9	https://github.com/snort3/libdaq
Glibc	2.31	https://www.gnu.org/software/libc/

Software Configuration	Details
Pcap	Alexa_200
Rules	snortrules-snapshot-31210
Hyperscan CMD	snort -c {{ snort_install_path }}/etc/snort/snort.lua --tweaks max_detect -r "$pcaps" -R $rules -z $cores --lua "search_engine = { search_method = 'hyperscan'}" --daq dump --daq-var load-mode=read-file --daq-var output=none -H -Q --warn-all -A none --pcap-loop 3
AC_BNFA CMD	snort -c {{ snort_install_path }}/etc/snort/snort.lua --tweaks max_detect -r "$pcaps" -R $rules -z $cores --daq dump --daq-var load-mode=read-file --daq-var output=none --pcap-loop 3 -H -Q --warn-all -A none

Performance varies by use, configuration and other factors. Learn more at www.Intel.com/PerformanceIndex.

Performance results are based on testing as of dates shown in configurations and may not reflect all publicly available updates. See backup for configuration details. No product or component can be absolutely secure.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

Intel technologies may require enabled hardware, software or service activation.

Intel does not control or audit third-party data. You should consult other sources to evaluate accuracy.

The products described may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

1122/DN/WIPRO/PDF 751287-001US