Solution Brief

Next Generation Firewall – Optimizations with 4th Gen Intel® Xeon® Scalable Processor

Next Generation Firewall (NGFW) is a very dynamic, demanding, and compute-intensive network security workload. This paper illustrates how technologies in 4th Gen Intel® Xeon® Scalable processors can be used so that NGFW runs optimally.

Authors

Brian Keating

Subhiksha Ravisundar

Harshini Yerra

Pinkesh Shah

Nirint Shah

Executive Summary

Next Generation Firewall (NGFW) is a network security workload that requires the ability to perform packet processing and decryption/encryption of traffic at line rate, while also performing deep packet inspection (DPI) all the way up to layer 7 of the OSI protocol stack to identify threats in network traffic. This DPI can include pattern matching, as well as inference using classical Machine Learning (ML) and/or Deep Learning (DL) models to identify applications and/or detect malware in the traffic. Furthermore, as we will see, depending on the deployment, and the traffic profile (which can change dynamically), the amount of processing to be done for different packets can vary significantly. This combination of low-level layer 3-4 packet processing, higher layer protocol processing, cryptographic processing, pattern matching, ML/DL inference, and layer 7 applications makes the NGFW a very dynamic and demanding workload.

This paper demonstrates how platforms based on 4th Gen Intel® Xeon® Scalable processors are optimal for this demanding workload. We show how NGFW can use technologies in these processors, including Intel® QuickAssist Technology (Intel® QAT) for crypto acceleration, Intel® Advanced Matrix Extensions (Intel® AMX) for DL acceleration, Intel® Dynamic Load Balancer (Intel® DLB) for optimal dynamic load balancing of work across cores, as well as other features of the platform.

In this document, we also briefly consider the evolution of network security from appliance- or virtual appliance-based NGFW to cloud-delivered Firewall as a Service as part of the Secure (Access) Service Edge (SASE/SSE) architecture. Most of the features of processors that benefit NGFWs apply equally in this new manifestation.

The target audience for this document includes Chief Information Security Officers (CISOs), System Architects, and those defining the architecture and platform for network security applications and appliances.

This document is part of the Network and Edge Platform Experience Kits.

Introduction

According to Gartner [1], Next Generation Firewall (NGFWs) are "deep-packet inspection firewalls that move beyond port/protocol inspection and blocking to add application-level inspection, intrusion prevention, and bringing intelligence from outside the firewall." Most commercial NGFWs today include a traditional stateful firewall enhanced with application awareness, integrated content inspection features such as intrusion prevention and malware detection, a security gateway for IPsec and other VPN protocols, and potentially a TLS intercepting proxy to allow end-to-end TLS encrypted content to be decrypted, inspected, and re-encrypted. They need to be able to perform all this processing at line rate, including decryption and encryption and various types of content inspection: deep packet inspection (DPI) using pattern matching or machine learning (ML) and/or deep learning (DL) models to identify applications and/or detect malware in the traffic.

NGFWs are typically deployed as appliances or virtual appliances at the edge of the data center, on the boundary between networks of different trust levels (for example, the trusted LAN and the untrusted WAN).

Over time, several trends are leading to changes in how network security is deployed. As Gartner described in [2], "more users, devices, applications, services and data [are] located outside of an enterprise than inside." Gartner defined a new consolidated networking and security-as-a-service product category, delivered from the cloud, which they call Secure Access Service Edge (SASE). They subsequently noted [3] that some vendors' offerings focus on cloud-delivered security capabilities without the networking, leading to a new product category that they call Secure Service Edge (SSE).

This document is organized as follows:

- We first describe the key features of a NGFW
- We then describe the architecture of a typical NGFW, and of a sample NGFW that we built using open-source components for architectural analysis and benchmarking
- We describe the use cases and key performance indicators (KPIs) of a NGFW
- We describe some of the key system bottlenecks that limit the performance of the sample NGFW that we built
- We introduce the technologies in the 4th Gen Intel Xeon Scalable processors that help alleviate these key bottlenecks, and show how they can improve the KPIs
- We describe the evolution of network security from NGFW to SASE/SSE, including how these same technologies can be brought to bear on SASE to improve its KPIs as well
- Finally, we present a summary
- As an appendix, we provide the results of the benchmarking described earlier in this document

Key Features

As described in the original Gartner research note on NGFW [4], a NGFW should have the following attributes:

- Basic stateful firewall capabilities, such as packet filtering, NAT, VPN capabilities
- Integrated network intrusion prevention (IPS)
- Application awareness (the ability to identify applications and make policy decisions based on the application)

This definition explicitly excluded certain features such as data loss prevention (DLP), secure web gateways (SWGs) that do URL filtering, and antivirus. However, many NGFWs today do integrate some of these capabilities.

Another key feature not discussed in that document but that many vendors implement is the ability to detect intrusions "hidden" in end-to-end encrypted traffic such as TLS or QUIC by decrypting, inspecting, and then re-encrypting the traffic. This feature can typically be implemented only in an enterprise environment or equivalent (for example, an educational campus), where the network operator can be trusted to perform this sort of TLS or QUIC interception on behalf of the clients.

Architecture

Every network security vendor's NGFW has its own unique implementation, but there are some common aspects to how they are architected. Figure 1 shows a generic architecture.

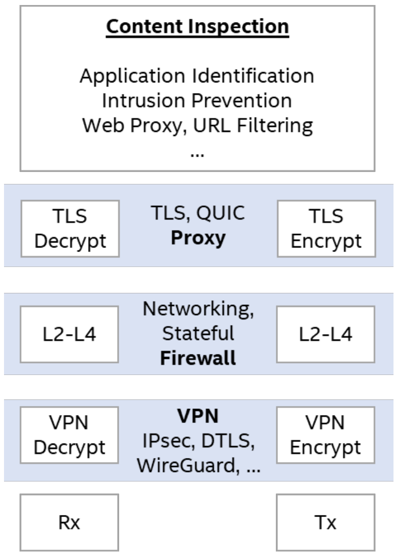


Figure 1. NGFW Generic Architecture

Within this architecture, we identify the following components:

- There is typically a **VPN** layer that implements a VPN protocol such as IPsec, DTLS, or WireGuard.
- There is typically a data plane that implements the **stateful firewall** features. This includes L2-L4 networking features including routing and network address translation (NAT). This typically also implements some form of access control lists (ACLs) to allow traffic to be filtered based on L3/L4 headers (IP addresses and TCP or UDP port numbers). These ACLs should be stateful, meaning that for example, connections initiated from the trusted network may be allowed, while those initiated from the untrusted network can be blocked.
- In those environments where the network operator is trusted to perform TLS or QUIC interception, there is typically a **TLS Proxy** (or QUIC proxy) that implements the proxying. This includes intercepting the handshakes to negotiate separate connections with the client and the server, including dynamically generating and presenting a certificate to the client for the server being impersonated, signed by the proxy using a private key for which the clients have installed the corresponding public key as a trusted CA; and then performing the decryption and re-encryption of the content before and after inspection, respectively.
- There is typically some component that performs **application identification** (App ID). There are many techniques used for App ID, including using:
  - Lookups based on packet metadata (for example, IP addresses, port numbers, and/or fields extracted from higher layers of the stack such as HTTP)
  - Pattern matching (including regular expression pattern matching) over the packet payload
  - Machine learning based on features extracted from the traffic, including the above but also such features as packet lengths and/or inter-arrival times
  - Deep learning over some or all of the packets themselves

  After the application has been identified (which is typically after seeing the first one or more packets of the flow), the NGFW can choose to apply different policies based on the identified application. For example, some applications may be considered "safe" or "trusted" and can bypass more detailed content inspection.
- There is typically an **Intrusion Prevention System** (IPS) that integrates with this data plane, through which traffic is sent by default. After the application has been identified using App ID, a policy may be applied that determines whether future packets of the flow also need to be inspected by the IPS.

For the purposes of benchmarking and analyzing the NGFW workload, we at Intel have instantiated a sample NGFW stack using open-source components. Figure 2 shows the logical view of this stack. Figure 3 shows the process view.

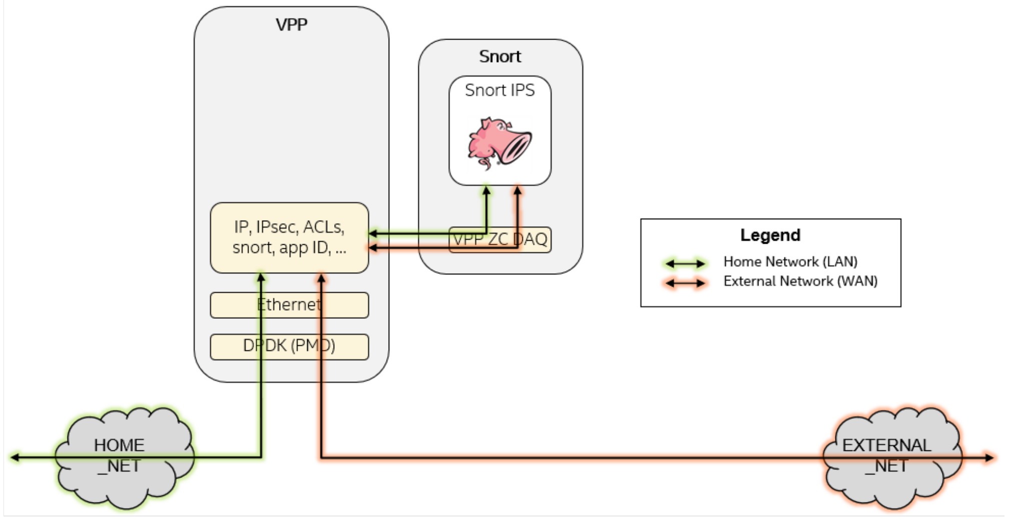


Figure 2. NGFW Sample Architecture - Logical View

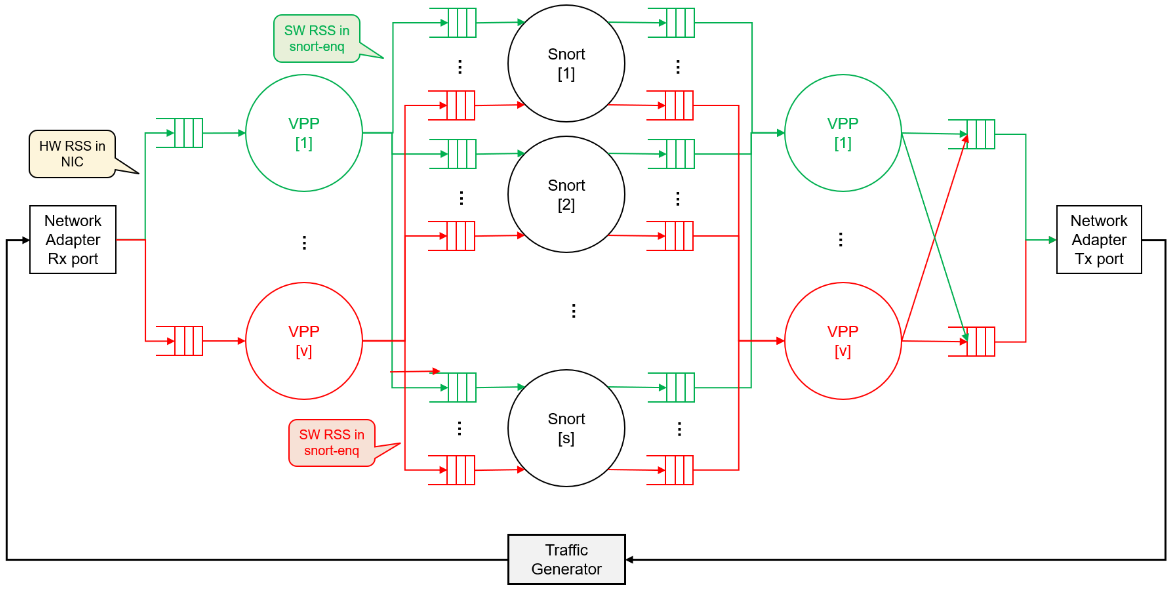


Figure 3. NGFW Sample Architecture - Process View

The sample NGFW stack instantiation uses the following open-source components:

- VPP (see [5]) acts as the data plane. This implements the basic stateful firewall including routing, NAT, stateful ACLs, and IPsec VPN. This is a multi-threaded process, with each VPP worker thread pinned to a dedicated CPU core or hardware thread (in the case of CPUs such as 4th Gen Intel Xeon Scalable processors that support SMT or hyper-threading).
- Snort (see [6]) acts as the IPS. We use Snort 3, which supports multi-threading. Here again, Snort worker threads are pinned to dedicated CPU cores or hardware threads.
- Snort and VPP are integrated using the Snort plugin to VPP. This uses a set of queue pairs for sending packets between VPP and Snort. The queue pairs, and the packets themselves, are stored in shared memory. The plugin includes the following subcomponents:
  - Two new VPP graph nodes:
    - snort-enq, which is implemented as a feature on the ip4-unicast feature arc. This makes a load-balancing decision about which Snort thread should process the packet and then enqueues the packet to the corresponding queue.
    - snort-deq, which is implemented as an input node that polls from multiple queues, one per Snort worker thread.
  - A new DAQ for Snort. This implements the Snort DAQ API functions to receive and transmit packets by reading from and writing to the relevant queues.

Not shown in the diagrams is the TLS Proxy. This can be implemented in one of several different ways:

- Within the data plane process, for example, VPP
- Within the IPS process, for example, Snort
- Within a separate "external" process using a different open-source implementation, for example, Squid (see [7]), or NGINX (see [8]). This can be integrated with VPP via the VCL layer that implements the sockets API.
- Using some combination of the above, for example using an external proxy to perform the TLS handshakes, after which the keys can be shared with VPP to "splice" or "cut through" the connections in the data plane.

Use Cases and Key Performance Indicators

We define three use cases within the NGFW workload. These are:

- **Cleartext Inspection:** Firewall plus IPS
- **VPN Inspection:** Firewall plus IPSec VPN plus IPS
- **TLS Inspection:** Firewall plus TLS Proxy plus IPS

Within each of these, we measure the following key performance indicators (KPIs), based on the recommendations in the IETF Benchmarking Working Group (BMWG) NGFW Benchmarking methodology document [9]. They include the following:

- **Inspected Throughput** and Application Transactions per Second
- **Latency**, as measured by TTFB and TTLB. For each, we report minimum, average, and maximum values.
- **TCP Connection Rate:** The rate at which new TCP connections can be established
- **TLS Handshake Rate:** The rate at which new TLS connections can be established (for TLS Inspection only)
- **Memory Bandwidth:** Although not specified in [9], we also consider memory bandwidth to be a KPI, as this can be a system bottleneck in some configurations.

[9] specifies that these are to be measured for the following traffic profiles:

- A relevant Application Traffic Mix. The document does not define this mix, but notes that the details of this mix should be supplied along with the results, including the name of applications and L7 protocols, the percentage of emulated traffic for each, the percentage of encrypted traffic and used cipher suites and keys, and the used object sizes for each application and L7 protocol. For benchmarking our sample NGFW, Intel chose an Application Traffic Profile Mix called Enterprise Mix, which we generated using Ixia's IxLoad AppLibrary (see [10]). The details of this traffic profile are described in Table 7 and Table 8 in Appendix A: Benchmarking Results.
- HTTP/TCP for various object sizes from 1 KB to 256 KB. For our benchmarking, we focus on a single object size of 64 KB.
- HTTPS/TLS or QUIC for various object sizes from 1 KB to 256 KB. For our benchmarking, we focus on a single object size of 64 KB.

System Bottlenecks

A selection of the results of our benchmarking of our sample NGFW can be found in Appendix A: Benchmarking Results.

Based on this, we have identified the following aspects of the NGFW workload that consume a significant proportion of the compute cycles or other system resources and can therefore potentially bottleneck the system performance. In this section, we describe these, before going on to discuss how these can benefit from the Intel technologies described in Relevant Intel Technologies.

Compute

Content Inspection

Content Inspection is typically far more compute-intensive than data plane processing as it requires inspection of the entire payload of the packets. In our benchmarking, we found Snort to be by far the most compute-intensive component of the NGFW, especially when running the Cleartext Inspection use case with Snort configured to use the registered ruleset (see [11]) and to use the Maximum Detection[1] base policy (see [12]). For a typical traffic profile and ruleset, Snort may only be able to process traffic at rates in the order of a few hundreds of Mbps per core. By comparison, the data plane (VPP) may be able to process traffic at something in the order of 10 Gbps per core. As a result, we typically need to allocate far more cores to Snort processing than to VPP. This is true when all packets are processed by Snort; there may be scenarios in which the policy is such that certain flows do not need inspection. For more details, see [13], [14].

Figure 4 illustrates a simple block diagram of the Snort processing stages, including packet decode, preprocessing, detection, and logging. Of these, the pattern matching mostly happens in the Detect stage.

[1] It should be noted that Maximum Detection base policy is not recommended for production deployment. We are investigating the impact on throughput of different base policies. Meanwhile, all benchmarks in this paper were carried out with this policy.

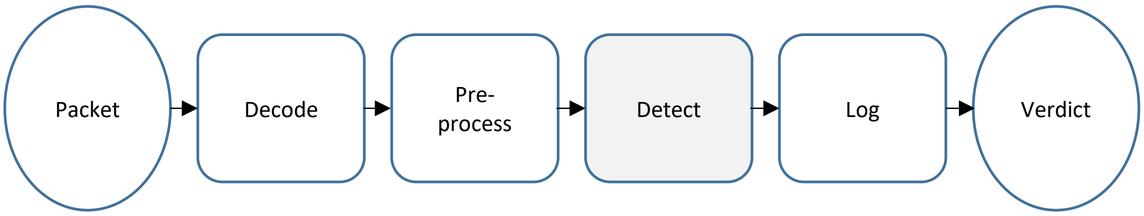


Figure 4. Snort Processing Stages

Figure 5 shows the output of Linux "perf top" when running our Snort in-memory benchmark on 4th Gen Intel Xeon Scalable processor, with the Enterprise Mix traffic profile and the Snort registered ruleset.

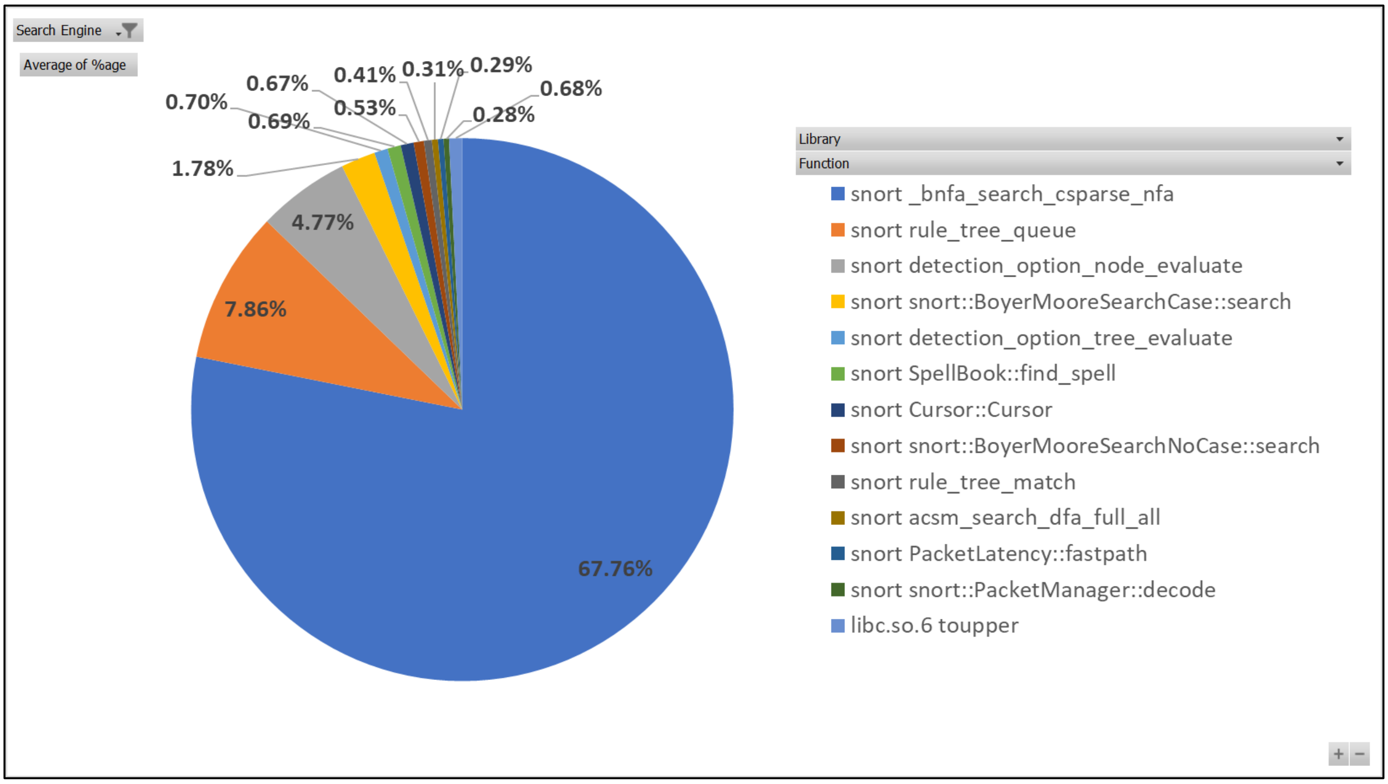


Figure 5. Performance Analysis of Snort using Default Search Engine ac-bnfa

As can be seen from the figure, the main bottleneck in Snort processing, consuming more than two thirds of the compute cycles, is the ac-bnfa component, which is the default detection engine used in the Detect stage of Snort. This component performs pattern matching, including fixed or literal pattern matching as well as regular expression pattern matching. As we see below, Intel's Hyperscan library has been integrated into Snort and provides a significant reduction in this aspect of the workload.

Cryptography

For the VPN Inspection and TLS Inspection use cases, where most of the traffic must be decrypted or encrypted (or both, in the case of TLS Inspection), then a significant amount of compute is required to perform this encryption and decryption at line rate. This uses symmetric (secret key) cryptography. As we see below, 4th Gen Intel Xeon Scalable processors have multiple options for accelerating or offloading this processing.

In the case of TLS inspection, then every new connection also requires a TLS handshake to be negotiated, which uses asymmetric (public key) cryptography. While the connection rates are lower than the packet rate, asymmetric (public key) cryptography is significantly more compute intensive than symmetric (secret key) cryptography, so this can consume a significant proportion of the CPU as well. Again, Intel has multiple solutions to address this potential bottleneck.

ML/DL Inference

Some NGFWs use ML and/or DL to do application identification, malware detection, or "fuzzy" URL inspection. This aspect of the workload is also highly compute intensive. Again, 4th Gen Intel Xeon Scalable processors include multiple technologies to boost the performance of ML and DL operations.

Note that inference typically must be done once per flow, rather than once per packet. As a result, this tends to impact the Connection Rate KPI as well as the Inspected Throughput.

Memory Bandwidth

For the cleartext inspection use case, memory bandwidth on Intel® Xeon® Gold 6438N processor is less than 1% of the available memory bandwidth, so memory bandwidth is not a bottleneck for this use case.

Preliminary data from the TLS inspection use case suggests that the memory bandwidth is much higher than this. This is due, at least in part, to the large number of buffer copies that happen as the data traverses the stack, including multiple separate processes (VPP, Snort, and an external proxy) and multiple traversals up and down the TCP/IP stack.

4th Gen Intel Xeon processors have significantly higher memory bandwidth than prior generations due to higher memory frequencies of up to 4800 MT/s, which can help with such high memory bandwidth applications.

Note that we do not consider memory bandwidth further in this document.

Power

The power consumed by a processor running a workload depends on several factors: processor model, number of cores used, core frequency, fabric frequency, memory bandwidth, network bandwidth, workload power intensity, and temperature. Each processor model has a maximum power called TDP (thermal design power).

There are two modes for core frequency:

1) Fixed frequency: As described later, as part of Intel® Speed Select Technology – Performance Profile (Intel® SST-PP) technology, 4th Gen Intel Xeon Scalable processors offer three fixed frequency modes called performance profiles. Each mode has different fixed frequency, and a customer can choose the highest frequency through BIOS that gives the best performance for the application and still keeps processor power below the processor 's rated TDP.
2) Turbo frequency: In this configuration, the processor autonomously manages core frequency and keeps it at the highest value for the processor TDP. In the process of achieving the best frequency, the processor measures power and adjusts the frequency in real time to keep power at TDP.

We see below that when we run the NGFW workload using the default performance profile with fixed frequency, it consumes less power than TDP. This provides us opportunity to use either a performance profile with a higher fixed frequency, or turbo mode. In either case, we can achieve higher throughput.

Other Bottlenecks

Slow Packets

We have observed significant variation in the compute cycles required to process different packets in Snort. The ratio of cycles taken by the slowest vs. the fastest packets varies with traffic profile and ruleset, but for the "default" combination we have chosen (Enterprise Mix traffic profile plus Registered rules), with the Maximum Detection base policy, we see a ratio of 943:1, with the fastest packets consuming ~2.2K cycles and the slowest packets consuming ~2M cycles. The average (mean) is about 20K cycles.

Figure 6 is a plot showing the packet cycles consumed by each packet, as measured using timestamps in the DAQ layer of Snort. It shows that, for this traffic profile, the packet cycles are clustered well below 100 thousand cycles, and just below a million cycles, with a few outliers up around two million cycles.

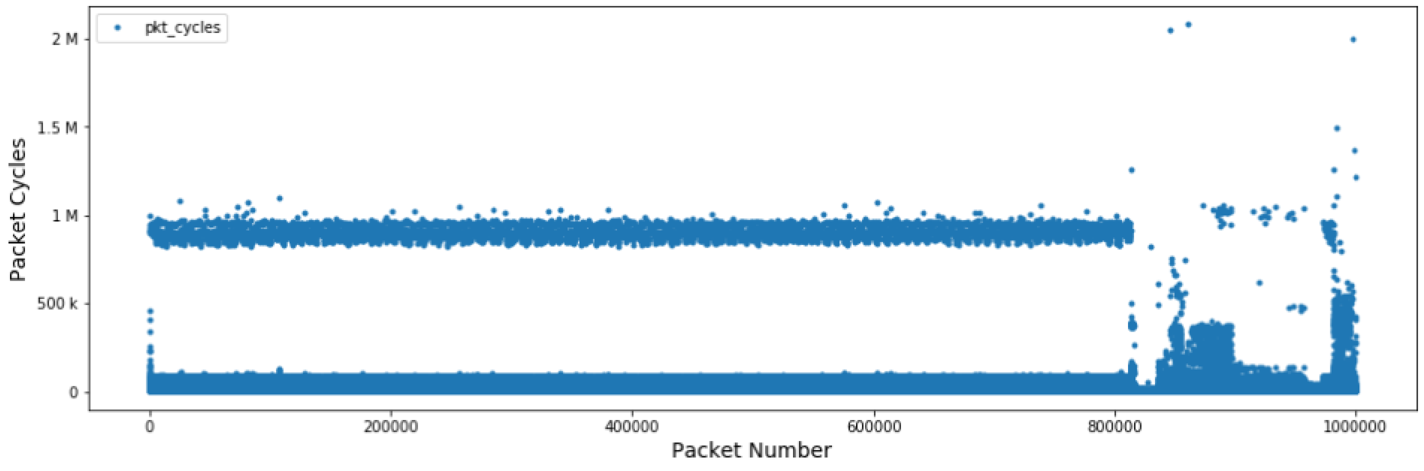


Figure 6. Packet Cycle Count for Enterprise Mix Traffic Profile

The details of **why** some Snort packets are so slow is the subject of a planned future paper, but the short explanation is that at least some of the slowest packets tend to be upstream TCP ACKs that acknowledge a large amount of data sent in the downstream direction. When combined with certain Snort features that are enabled in our default configuration, these packets end up consuming significant compute cycles. For example:

- **File signature generation:** This is enabled via the configuration parameter "file_id.enable_signature = true" in *max_detect.lua*. It is a malware detection feature that causes file signatures to be generated and logged on files being downloaded. This signature consists of a SHA256 hash. It is done incrementally; packets being downloaded are stored, and the hash is computed when the corresponding TCP ACK is seen going in the upstream direction. For the slowest packet in our Enterprise Mix traffic profile, we see 685 packets being acknowledged, resulting in a hash being computed over ~880 KB of data at a cost in software of ~2.9 cycles/Byte, or total of ~2. 5 M cycles.
- **JavaScript normalization:** This is enabled via the configuration parameter "http_inspect.normalize_javascript = true" in *max_detect.lua*. It is another malware detection feature, and specifically a countermeasure against JS obfuscation techniques. Like file signature generation, this is done incrementally on seeing the upstream ACK and can result in significant cycles being consumed for individual packets.
- **File decompression:** By default, Snort attempts to decompress (or inflate) compressed files that are downloaded. Like file signature generation, this is done incrementally on seeing the upstream ACK and can result in significant cycles being consumed for individual packets.
- **SSL certificate parsing:** This is on by default. As a result of lazy initialization in the OpenSSL library, the first certificate to be parsed results in some extremely slow initialization code being invoked. This can be addressed by patching OpenSSL to do this initialization up front, rather than on the parsing of the first certificate.

The **implication** is that, in the time taken to process one or more slow packets, thousands of packets can arrive, leading to a need for very deep queues, up to tens of thousands of packets in some cases.

These deeper queues lead to one or more issues:

- **Significant memory capacity:** We have chosen a DAQ queue depth of 8K packets. The current implementation of Snort/VPP integration uses separate queue pairs for each VPP thread and Snort thread, so there can be up to 2*v*s queue pairs, where:
  - v = number of VPP threads
  - s = number of Snort threads

  For CPUs with a large number of hyper-threaded cores (for example, Intel Xeon 4th Gen Scalable processors have SKUs with up to 60 cores and 120 threads), we may end up with, for example, v in the order of 10 and s in the order of 100, which means ~2 K queues, and up to ~16 M queue entries. Not only do these queues themselves consume memory, but this also means that we need to allow for up to that many packets in flight, leading to very large buffer pools.
- **Inefficient cache utilization:** Because of the high memory usage, most packets likely have been evicted from the L2 and L3 caches by the time they get processed.
- **Increased packet latency:** Obviously, the "slow" packets themselves incur high latency, but so too do other packets that are being processed on the same thread. This is unavoidable in the case of other packets in the same flow, which must be processed in order, but may be avoidable for packets in other flows that happen to arrive on the same thread as the slow packets.

- **Increased jitter:** There may be a large variation in the latency incurred by different packets.
- **Lower throughput:** If shallower queues are used, then throughput needs to be reduced to avoid overflowing these queues, which leads to packets being dropped.

Poor Entropy

By default, the VPP snort-enq graph node uses static load balancing to distribute traffic across the Snort threads. With this scheme, flows are "pinned" to specific queues (and hence specific cores or hardware threads) based on a hash calculated over the 5-tuple. Statistically, this gives good distribution, but only if certain criteria are met, including having a relatively large number of flows (realistically, at least 2000 flows per queue), and relatively "equal" flows, meaning that each flow should consume an approximately equal percentage of the core's available CPU cycles. Exceptions to this can be categorized as "elephant flows" and "baby elephants".

- **Elephant flows** are flows where the packet rate and corresponding work exceed the capacity of a single core. Handling these requires the flow to be sprayed across multiple cores, which requires the packets to be reordered after processing and can be an issue if some or all of the processing needs to be done in order.
- **Baby elephants** are those flows that consume a significant percentage of the cycles of a core (for example, more than 5% but less than 100%). Having just a few of these assigned to a single core may cause the capacity of that core to be exceeded.

Poor entropy can be further exacerbated by the Slow Packets issue above. Even if the number of packets going to each Snort thread is about the same, the amount of work to be done on the different cores can vary more widely if some cores get more slow packets than others.

In general, poor entropy leads to some cores receiving more work than others. To avoid dropping packets on these busiest cores, the traffic rate must be reduced to the rate at which these slowest cores can keep up. This means that other cores are underutilized.

Of course, entropy can vary over time, and bursts of poor entropy can be addressed by deep queues to avoid packet drops, but this leads to the latency and other issues noted earlier.

Relevant Intel Technologies

The following Intel technologies can help alleviate some of the potential system bottlenecks identified earlier.

We summarize how these Intel technologies can be used to alleviate the various potential system bottlenecks in Table 1 in the Summary section.

Hyperscan

Hyperscan (see [15]) is a high-performance multiple regex matching library available as open source with a C API. Hyperscan uses hybrid automata techniques to allow simultaneous matching of large numbers of regular expressions across streams of data. Hyperscan also takes advantage of the latest Intel® Advanced Vector Extensions 512 (Intel® AVX-512) vectorized bit manipulation instructions (vBMI) available on both the 3rd and 4th Gen Intel Xeon Scalable processors.

Hyperscan can be used to help alleviate the following potential system bottlenecks in NGFW:

- **Compute/Snort/Pattern Matching:** Hyperscan has been integrated with Snort to provide a significant boost in performance compared to the default search engine.
- Figure 7 shows the output of the "perf top" utility for Snort using Hyperscan as the search engine. Compare this to Figure 5, which shows the output with the default engine (ac-bnfa). The ac-bnfa detection engine consumes a significantly higher percentage of the CPU cycles. By reducing the cycles consumed performing pattern matching, Snort achieves a significantly higher throughput.

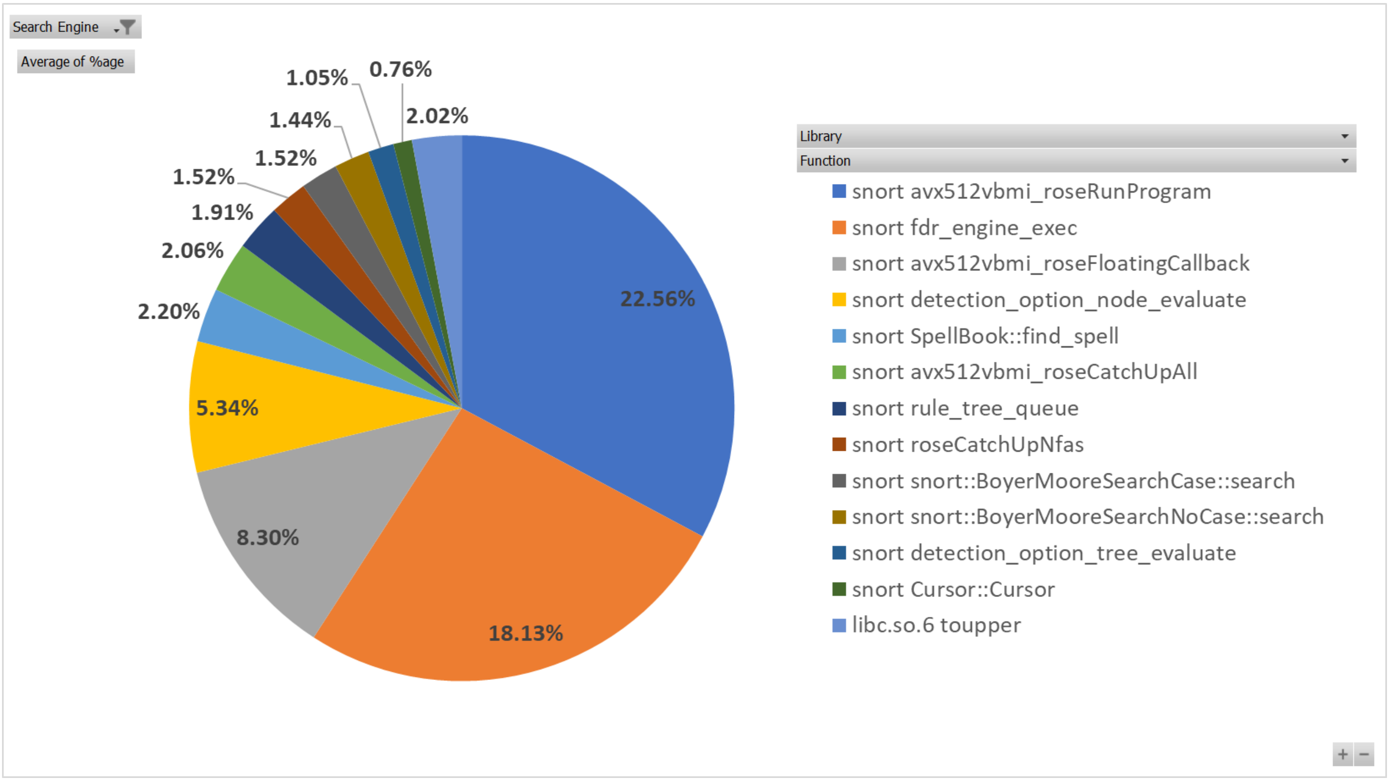


Figure 7. Perf Analysis of Snort using Search Engine Hyperscan

Intel® QuickAssist Technology (Intel® QAT) and Software Crypto

Intel QAT is a technology for accelerating data encryption/decryption, public key cryptography for key exchange, as well as compression/decompression. This acceleration technology is integrated into 4th Gen Intel Xeon Scalable processors, supporting rates of up to 400 Gbps for common cryptographic ciphers and up to 160 Gbps verified compression. You can learn more about Intel QAT at [16].

The 4th Gen Intel Xeon Scalable processor also inherits the Intel AVX-512 instruction accelerations. The relevant encryption instructions to boost cryptographic operation performance include VPMADD2 (vector instruction that does integer multiply accumulate and can optimize various public key cryptography operations), vAES (vector version of the AES-NI instructions), vPCLMUL (vectorized carry-less multiply, used to optimize AES-GCM), and SHA-NI (secure hash algorithm new instructions). Intel has previously published details of how it was possible to achieve over 1 Tbps of IPsec performance using these instructions in 3rd Gen Intel Xeon Scalable processors; see [17]. These instructions are further enhanced in 4th Gen Intel Xeon Scalable processors by the 19% Instructions Per Cycle (IPC) performance improvement of general-purpose instructions over the prior generation.

Intel also provides software libraries to simplify use of these instructions, including the Intel® QuickAssist Technology Engine for QAT OpenSSL* (Intel® QAT Engine for OpenSSL*) (see [18]) and the Intel® Multi-Buffer Crypto for IPsec library (see [19]).

Intel QAT can be used to help alleviate the following potential system bottlenecks in NGFW:

- **Compute/Cryptography/Symmetric:** Intel QAT has been integrated with VPP via the DPDK cryptodev API to improve the performance of IPsec by up to 5x for AES256-CBC+HMAC-SHA256, and up to 1.6x for AES256-GCM, according to internal measurements by Intel. We have also measured up to 5.9x increase in throughput for WireGuard using ChaCha20-Poly1305, as described in [20].
- **Compute/Cryptography/Asymmetric:** Intel QAT has been integrated with OpenSSL via the QAT engine. Using the Intel QAT hardware, TLS handshake performance for the cipher suite NGINX ECDHE-x25519-RSA2K improves by up to 6.3x vs. unoptimized software at 1c1t, and by up to 1.68x vs. optimized software at 12c24t, as described in [21].

As potential future work, we also plan to look at using Intel QAT to potentially accelerate the following:

- **Compute/Snort/File signature generation:** Intel QAT could also be used to accelerate the SHA256 hashing of larger data blocks in Snort.
- **Compute/Snort/File decompression:** Intel QAT could also be used to accelerate the decompression of compressed files in Snort.

Intel® Deep Learning Boost (Intel® DL Boost)

Intel DL Boost was first introduced in the 2nd Gen Intel® Xeon® Scalable processors with the addition of Intel AVX512 VNNI, an extension to the Intel AVX-512 instruction set. It brings accelerated performance to demanding AI workloads without discrete add-on accelerators. It can significantly improve performance for common AI inferencing and training workload. 4th Gen Intel Xeon Scalable processors add support for Intel® Advanced Matrix Extensions (Intel® AMX), a new component of Intel DL Boost technology. This built-in accelerator is dedicated to the matrix multiplication at the heart of deep learning workloads. Intel AMX combines a new instruction set that operates on large matrices in a single operation with two-dimensional register files that store larger chunks of data for each core. For more information on Intel DL Boost, including Intel AMX, see [22].

Intel DL Boost can be used to help alleviate the following potential system bottleneck in NGFW:

- **Compute/ML/DL Inference:** For network security applications that use Deep Learning inference, whether to detect malware, identify anomalies, or for other use cases, using Intel DL Boost technology including Intel AMX can significantly improve the performance of this component of the workload. For example, as described in [22], by applying IPEX static post-training quantization and using Intel AMX on 4th Gen Intel Xeon Scalable processors (specifically the Intel Xeon Gold 6428N processor), the performance of the BERT model – which can be trained for cybersecurity applications – can get 7.09 X ~ 8.36 X performance improvement compared to the 3rd Gen Intel Xeon Scalable processor (specifically, the Intel® Xeon® Gold 6338N processor).

Intel® Dynamic Load Balancer (Intel® DLB)

Intel DLB is a hardware managed system of queues and arbiters connecting producers and consumers. These producers and consumers are typically software threads running on different cores or threads. Intel DLB is a PCI device in the CPU package. It implements load balancing features including lock-free multi-producer/multi-consumer operation, multiple priorities for varying traffic types, and various distribution schemes including unordered, ordered, and atomic. For details, see [23].

Intel DLB can be used to help alleviate the following potential system bottlenecks in NGFW:

- **Queue Management:** In a CPU with a large number of threads, if the enqueue/dequeue rates are sufficiently high, the queue management itself may be a significant portion of processing cycles due to time wasted polling empty queues, lock contention, and cross core snoop latencies. Intel DLB can greatly reduce these costs by removing locks/multi-queues and eliminating cross core snoops.
- **Other Bottlenecks/Slow Packets:** With static load balancing, flows are "pinned" to specific cores. While slow packets are being processed, other packets from the same flow, and those from other flows assigned to that core, build up on the queue. With Intel DLB, packets from other flows can be dynamically load balanced to other cores, thereby reducing latency incurred by those other flows.
- **Other Bottlenecks/Poor Entropy:**
  - Elephant flows can be dealt with in one of the following ways:
    - For packets that can be processed out of order but need to be sent in-order, this can be achieved using ordered queues.
    - For packets that need to be processed in order, these can be handled by breaking the processing into multiple stages with only a subset of the processing – the "critical section" – needing to be done in-order. This part can then be handled using atomic queues, while the other parts can use ordered queues.
    - For an example of how this can be done for IPsec traffic, see [24].
  - Baby elephants can be dealt with similarly to how it is described for Slow Packets.

Intel Speed Select Technology – Performance Profiles (Intel SST-PP) and Intel Turbo Boost Technology

Intel SST-PP provides customers with multiple processor configurations within a single device, allowing the customer to select between an optimal balance of core count, base frequency, and thermal design point (TDP). Different profiles can be selected based on the workload.

Table 4 shows two different available profiles for the Intel® Xeon® Platinum 8470N. Meanwhile, Figure 12 compares the throughput and power for these two different modes. We can see that in the default Network mode, with a core frequency of 1.7 GHz, we have 30W of "headroom". By configuring the processor in Compute mode, with a core frequency of 2.1 GHz, the throughput increases by 21% while the power remains below the 300 W TDP.

As noted earlier, it is also possible to increase the operating frequency by enabling Intel Turbo Boost Technology. In this case, the processor dynamically selects the highest frequency for the cores while keeping the operating power at TDP. Figure 11 shows a 21% performance gain with turbo enabled due to the increase in base frequency from 1.8 GHz to 2.2 GHz for a different SKU, namely the Intel Xeon Gold 6428N, taking advantage of the power headroom at the fixed base frequency. This is the same performance benefit as we see with the SST-PP compute profile on the Platinum SKU above. However, configuring the Intel SST-PP compute profile sets the frequency to a fixed value, whereas with turbo the frequency can vary, which can lead to jitter.

Figure 9 shows 1.26x performance improvement for 4th Gen Intel Xeon Gold network processor versus 3rd Gen Intel Xeon Gold network processor using turbo mode for both generations.

The key take-away here is that the NGFW Cleartext Inspection use case has power headroom when running at default network mode fixed frequency. By using either a higher frequency profile or turbo frequency, it is possible to increase performance by up to 21%.[2]

Evolution to SASE/SSE

As noted earlier, network security deployments are evolving from appliance-based NGFWs to cloud-delivered SASE/SSE solutions. Some of the key differences include:

- **Maturity:** NGFW is a mature technology, having been deployed for over a decade. SASE/SSE is still an evolving product category.
- **Deployment:** NGFWs are typically deployed in an appliance or virtual appliance form factor, on the boundary between networks with a different level of trust. SASE/SSE is delivered as a service from the cloud and assumes zero trust.
- **Development:** NGFWs tend to be developed as monolithic applications, consisting of tightly coupled components. They are typically deployed on bare metal or in virtual machines (VMs) on dedicated hardware or general-purpose servers. SASE/SSE is intended to be developed using cloud native principles, "built on microservices with the ability to scale out as needed" (per [2]) and deployed in containers.
- **Functionality:** While both NGFW and SASE/SSE are, at some level, collections of network security functions, SASE includes additional functions not typically included in NGFW, including CASB, SWG, RBI, ZTNA, WAAP/WAF, DNS Protection.

Despite these differences, the same Intel technologies that improve the KPIs of NGFW in an appliance or virtual appliance deployment are expected to be mostly applicable to SASE/SSE cloud-based deployments, as well.

Summary

Table 1 maps the various potential system bottlenecks to the KPIs that they impact and the relevant Intel technologies that can alleviate them.

Table 1. Mapping Bottlenecks, KPIs, and Relevant Intel Technologies

Bottleneck		KPI	Relevant Intel Technologies
Compute	Snort/Pattern Matching	Inspected Throughput	Hyperscan
	Cryptography/Symmetric	Inspected Throughput	Intel QAT and Software Crypto
	Cryptography/Asymmetric	TLS Handshake Rate	Intel QAT and Software Crypto
	Snort/File Signature Generation	Inspected Throughput	Intel QAT and Software Crypto
	Snort/File Decompression	Inspected Throughput	Intel QAT and Software Crypto
	ML/DL Inference	Inspected Throughput, Connection Rate, Latency	Intel DL Boost
Power			Intel SST-PP and Intel Turbo Boost Technology
Other Bottlenecks	Slow Packets		Intel DLB
	Poor Entropy/Elephant Flows		Intel DLB
	Poor Entropy/Baby Elephants		Intel DLB

Terminology

Table 2. Terminology

Abbreviation	Description
1C2T	One Core 2 Threads (HT enabled)

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

Abbreviation	Description
ACL	Access Control List
App ID	Application Identification
BERT	Bidirectional Encoder Representations from Transformers
BW	Bandwidth
CA	Certificate Authority
DAQ	Data AcQuisition, a library for packet input/output in Snort
DL	Deep Learning
DLP	Data Loss Prevention
DPDK	Data Plane Development Kit
DPI	Deep Packet Inspection
DTLS	Datagram Transport Layer Security
HT	Hyper-Threading; also known as Symmetric Multi-Threading (SMT)
IPC	Instructions Per Cycle
IPS	Intrusion Prevention System
JS	JavaScript
KPI	Key Performance Indicator
ML	Machine Learning
MSR	Model Specific Register
NAT	Network Address Translation
NGFW	Next Generation Firewall
OSI	Open Systems Interconnection
PDR	Partial Drop Rate
QUIC	A secure, general purpose, connection-oriented transport protocol as defined by IETF RFC 9000
SASE/SSE	Secure (Access) Service Edge
SMT	Symmetric Multi-Threading; also known as Hyper-Threading (HT)
SWG	Secure Web Gateway
TCP	Transmission Control Protocol
TDP	Thermal Design Power/Thermal Design Point
TLS	Transport Layer Security
TTFB, TTLB	Time to First Byte, Time to Last Byte
UDP	User Datagram Protocol
vBMI	Vectorized Bit Manipulation Instructions
VCL	VPP Comms Library
VPP	Vector Packet Processing

Document Revision History

Revision	Date	Description
001	February 2023	Initial release.

References

[1] Gartner, "Next-generation Firewalls (NGFWs)," [Online]. Available: https://www.gartner.com/en/information-technology/glossary/next-generation-firewalls-ngfws.

[2] Gartner, "The Future of Network Security Is in the Cloud," 2019.

[3] Gartner, "2022 Strategic Roadmap for SASE Convergence," 2022.

[4] Gartner, "Defining the Next Generation Firewall," 2009.

[5] "VPP Technology," [Online]. Available: https://fd.io/gettingstarted/technology/.

[6] "Snort," [Online]. Available: https://www.snort.org/snort3.

[7] "Squid," [Online]. Available: http://www.squid-cache.org/.

[8] F5, "Nginx," [Online]. Available: https://www.nginx.com/.

[9] B. Balarajah, C. Rossenhoevel and B. Monkman, "Benchmarking Methodology for Network Security Device Performance," 2022. [Online].

[10] Ixia, "AppLibrary," [Online]. Available: https://www.keysight.com/ie/en/products/network-test/protocol-load-test/applibrary.html.

[11] Cisco, "Snort Rules," [Online]. Available: https://www.snort.org/downloads#rules.

[12] Cisco, "Maximum Detection Base Policy," [Online]. Available: https://www.cisco.com/c/en/us/support/docs/security/firepower-ngfw/214405-what-are-the-metrics-used-to-determine-t.html#anc6.

[13] "Snort Reference," [Online]. Available: https://github.com/snort3/snort3/releases/download/3.1.43.0/snort_reference.html.

[14] "Snort User Manual," [Online]. Available: https://github.com/snort3/snort3/releases/download/3.1.43.0/snort_user.html.

[15] Intel, "Hyperscan.io," [Online]. Available: https://www.hyperscan.io/.

[16] Intel, "Intel® QuickAssist Technology (Intel® QAT) Overview," [Online]. Available: https://www.intel.com/quickassist.

[17] Intel, "3rd Generation Intel® Xeon® Scalable Processor - Achieving 1 Tbps IPsec with Intel® Advanced Vector Extensions 512 (Intel® AVX-512) Technology Guide," [Online]. Available: https://networkbuilders.intel.com/solutionslibrary/3rd-generation-intel-xeon-scalable-processor-achieving-1-tbps-ipsec-with-intel-advanced-vector-extensions-512-technology-guide.

[18] Intel, "Intel® QuickAssist Technology (Intel® QAT) OpenSSL* Engine," [Online]. Available: https://github.com/intel/QAT_Engine.

[19] Intel, "Intel® Multi-Buffer Crypto for IPsec Library," [Online]. Available: https://github.com/intel/intel-ipsec-mb.

[20] Intel, "Intel® QAT - Accelerate WireGuard* Processing with 4th Gen Intel® Xeon® Scalable Processor," 2022. [Online]. Available: https://networkbuilders.intel.com/solutionslibrary/intel-qat-accelerate-wireguard-processing-with-4th-gen-intel-xeon-scalable-processor-technology-guide.

[21] Intel, "4th Gen Intel® Xeon® Scalable Processor & Intel® QuickAssist Technology: NGINX Performance," [Online].

[22] Intel, "Intel® Deep Learning Boost - Improve Inference Performance of BERT Base Model from Hugging Face for Network Security," [Online].

[23] N. McDonnell, "Queue Management and Load Balancing on Intel Architecture," *Network Builders,* 2018.

[24] Intel, "Load Balancing & Scaling of IPSEC Workloads".

[25] Intel, "SPR VPN Performance (doc from Georgii - need details!)," [Online].

Appendix A: Benchmarking Results

Note that at this time, we have not completed the VPN Inspection or TLS Inspection benchmarks. We present the results of our Cleartext Inspection benchmarking only.

NGFW: Cleartext Inspection[3]

Block Diagram and Test Setup

Figure 8 shows what the test setup looks like. The details of the platforms, including the processors, Ethernet Adapters, and configuration, are available in Table 16 and Table 17.

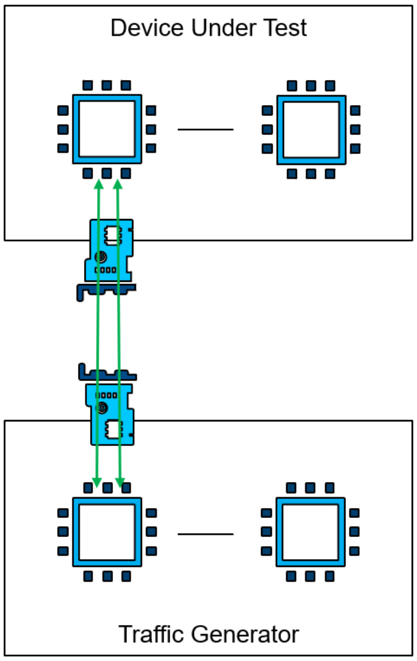


Figure 8: NGFW Test Setup

Summary of Generation to Generation Performance

In this section, we compare the performance of some specific SKUs of the 4th Gen Intel Xeon Scalable processor versus the prior generation. We first compare the performance of some Network Gold SKUs shown in Table 3. We then look at the impact of different profiles/modes using a Network Platinum SKU, as shown in Table 4, before going on to compare the "top-of-stack" SKUs shown in Table 5. Note that some network security appliance vendors tend to use these "top of stack" SKUs for their high-end appliances.

Table 3: Network Gold SKUs

Processor SKU	3rd Gen	4th Gen
SKU Name	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N
TDP (W)	185	185
Core Count	32	32
Base Core Frequency (GHz)	2.2	1.8
All-Core Turbo Frequency (GHz) for NGFW workload	2.4	2.2
Uncore Frequency (GHz)[4]	1.6	1.6

[3] For workloads and configurations visit www.Intel.com/PerformanceIndex. Results may vary.
[4] The uncore frequency was explicitly set using MSRs.

Table 4: Network Platinum SKU Performance Profiles

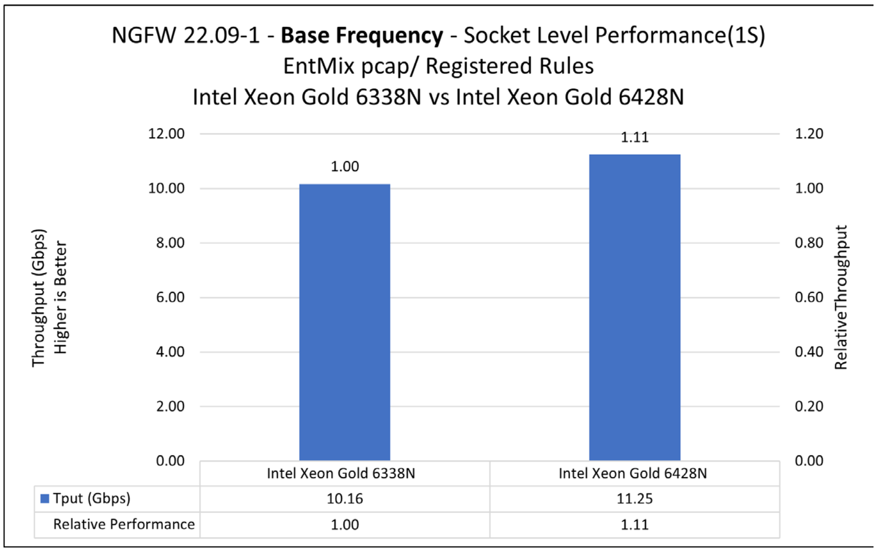
Processor SKU	Intel Xeon Gold 8470N (4th Gen)	
Profile/Mode	Network (Default)	Compute
TDP (W)	300	300
Core Count	52	52
Base Core Frequency (GHz)	1.7	2.1
Uncore Frequency (GHz)[5]	1.6	1.4

Table 5. "Top of Stack" SKUs

Processor SKU	3rd Gen	4th Gen
SKU Name	Intel Xeon Platinum 8380	Intel Xeon Platinum 8490H
Profile (aka Mode)	**n/a**	**Network**
TDP (W)	270	350
Core Count	40	60
Base Core Frequency (GHz)	2.3	1.7

The first three charts that follow compare the socket-level performance of the network SKUs at base core frequency and at turbo core frequency.

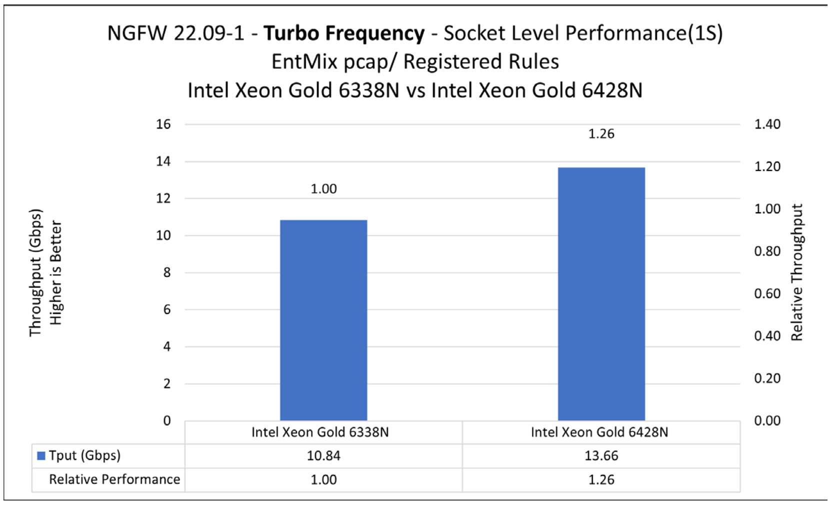
- Figure 9 shows a performance gain of 11%.
- Figure 10 shows a performance gain of 26%.
- Figure 11 shows a turbo boost of 21% for the Intel Xeon Gold 6428N.



	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N
■ Tput (Gbps)	10.16	11.25
Relative Performance	1.00	1.11

Figure 9. NGFW Gen to Gen Performance with Base Frequency (Results may vary. See Table 19 for configuration details)

[5] The uncore frequency was explicitly set using MSRs



	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N
■ Tput (Gbps)	10.84	13.66
Relative Performance	1.00	1.26

Figure 10. NGFW Gen to Gen Performance with Turbo Frequency (Results may vary. See Table 19 for configuration details)

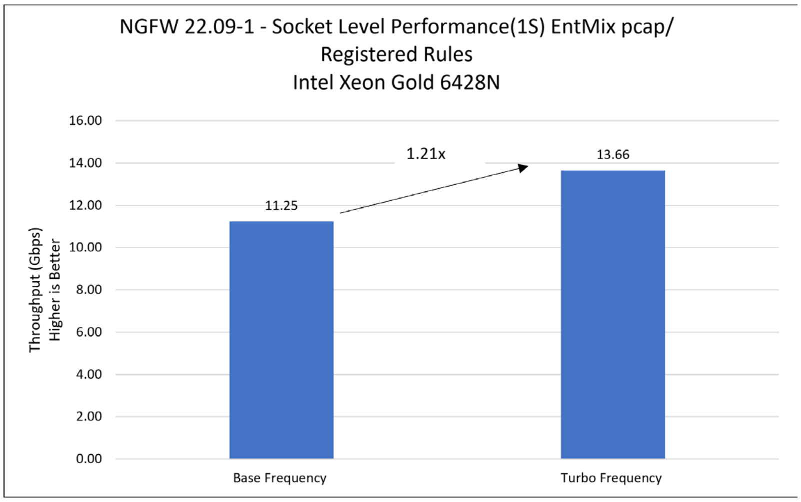
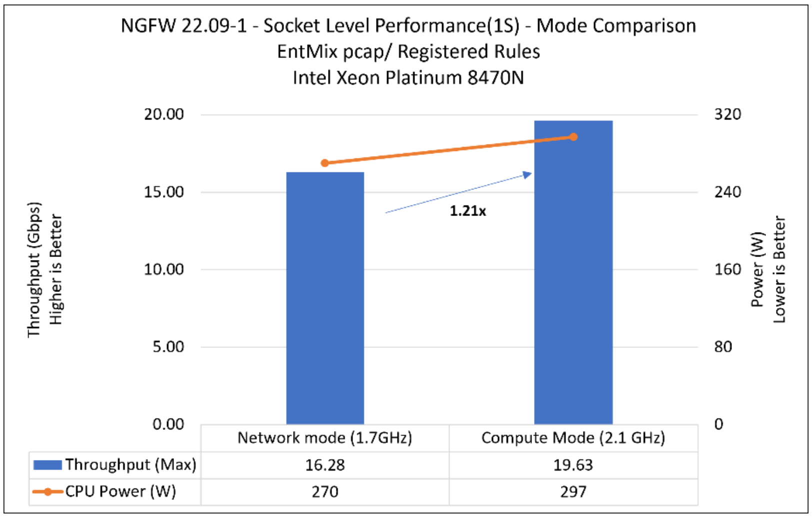


Figure 11. Performance at Base Frequency vs. Turbo Frequency on Intel Xeon Gold 6428N (Results may vary. See Table 19 for configuration details)

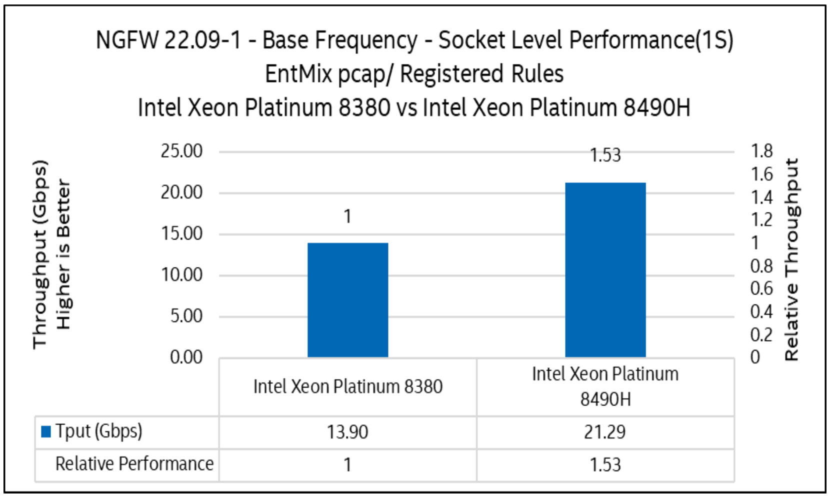
Next, we compare the socket-level performance of two different profiles of the Intel Xeon 8470N processor SKU. Figure 12 shows that we can achieve an additional 21% performance boost by configuring the processor with the "compute" mode/profile while remaining under the 300W TDP of the processor SKU.



	Network mode (1.7GHz)	Compute Mode (2.1 GHz)
Throughput (Max)	16.28	19.63
CPU Power (W)	270	297

Figure 12. Performance of Intel Xeon Platinum 8470 with Different Performance Profiles (Results may vary. See Table 19 for configuration details)

Finally, we compare the socket-level performance of top-of-stack SKUs at base frequency and at turbo frequency. Figure 13 shows that we can achieve approximately 50% more performance from the 4th Gen part, largely due to the 50% increase in the core count (from 40c to 60c). The relative IPC improvements of the cores are negated by the lower frequency of the 4th Gen part.



	Intel Xeon Platinum 8380	Intel Xeon Platinum 8490H
Tput (Gbps)	13.90	21.29
Relative Performance	1	1.53

Figure 13. Performance at Base Frequency for "Top of Stack" SKUs (Results may vary. See Table 20 for configuration details)

Raw Results

Table 6, Table 7, and Table 8 show the raw performance data behind the charts above. For each platform, it shows the throughput, the corresponding number of cores and threads running VPP and Snort, the operating core and uncore frequencies, the memory bandwidth, the CPU power, and some other derived numbers. From Table 6, the CPU power results show that there is a higher TDP headroom on the 4th Gen products compared to the 3rd Gen, which allows the frequency to be increased above the base frequency while staying within the TDP envelope. The memory bandwidth and Memory BW per core is very small for the Clear text inspection case.

Note that all benchmarks were run with the Enterprise Mix traffic profile and the Snort registered ruleset. The details of these can be found in Traffic Profile and Snort Ruleset Overview.

Table 6. NGFW Cleartext Inspection Socket Level Performance Raw Data for N SKUs

Metric	Units	Base Frequency		Turbo Frequency	
SKU		Intel Xeon Gold 6338N	Intel Xeon Gold 6428N	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N

Metric	Units	Base Frequency		Turbo Frequency	
Throughput (Max)	Gbps	10.16	11.25	10.84	13.66
Relative Throughput		1	1.11	1	1.26
No. of VPP cores		4C8T	3C6T	4C8T	3C6T
No. of Snort cores		26C52T	27C54T	26C52T	27C54T
Throughput per Core	Gbps	0.34	0.38	0.36	0.46
Throughput per Snort Core	Gbps	0.39	0.42	0.42	0.51
CPU Power	Watt	179	156	184	184
Core Freq	GHz	2.2	1.8	2.4	2.2
Uncore freq	GHz	1.6	1.6	1.6	1.6
Memory BW	GB/s	5.9	5.9	6.6	7.5
Memory BW per Core	GB/s	0.20	0.20	0.22	0.25

Table 7. NGFW Cleartext Inspection Socket Level Performance Raw Data Comparing Intel SST-PP Profiles on 8470N

Metric	Units	Intel Xeon Platinum 8470N	
Profile		NFV profile (1.7GHz)	Compute Profile (2.1 GHz)
Throughput (Max)	Gbps	16.28	19.63
Relative Throughput		1.00	1.21
No. of VPP cores		4C8T	4C8T
No. of Snort cores		46C92T	46C92T
CPU Power	Watt	270	297
Core Freq	GHz	1.7	2.1
Uncore freq	GHz	1.6	1.4

Table 8. NGFW Cleartext Inspection Socket Level Performance Raw Data Comparing Top-End SKUs

Metric	Units	Base Frequency	
SKU		Intel Xeon Platinum 8380	Intel Xeon Platinum 8490H
Throughput (Max)	Gbps	13.90	21.29
Relative Throughput		1.00	1.53
No. of VPP cores		4C8T	4C8T
No. of Snort cores		34C68T	54C108T
CPU Power	Watt	227	305
Core Freq	GHz	2.3	1.9
Uncore freq	GHz	1.6	1.6

Configuration

Traffic Profile and Snort Ruleset Overview

Table 9. Traffic Profile Details

Attribute	Units	Value
Name		Enterprise Mix
Source		Ixia IxLoad (v9.00) AppLibrary
Summary		

Attribute	Units	Value
Number of Flows		25,594
Number of Packets		999,835
Average Packet Size	Bytes	349
PCAP file size	MB	364

Table 10. Traffic Profile Details: Applications

Protocol	Port	Application	Num Flows	Flows (%)	Packets (%)
TCP	1521	Oracle Database	5027	19.6%	29.5%
	22	Secure Shell (SSH)	4678	18.3%	50.0%
	80	HTTP	241	0.9%	1.3%
	139	NetBIOS Session Service	125	0.5%	1.3%
	6881	BitTorrent	94	0.4%	2.9%
	443	HTTPS	81	0.3%	2.3%
	1494	Citrix Independent Computing Architecture (ICA)	79	0.3%	3.9%
	8426	(Not Specified)	77	0.3%	1.6%
	21	File Transfer Protocol (FTP)	61	0.2%	1.0%
	25	Simple Mail Transfer Protocol (SMTP)	27	0.1%	0.2%
	Others	Various (one flow each)	248	1.0%	3.1%
UDP	9875	Session Announcement Protocol (SAP)	7464	29.2%	1.5%
	17771	LogMeIn Hamachi VPN	7392	28.9%	1.5%
Total			**25594**	**100.0%**	**100.0%**

Table 11. Snort Rules Details

Attribute	Value
Name	Registered Rules
Source	https://www.snort.org/downloads#rules
Version	snortrules-snapshot-31110.tar.gz
Number of Rules	44040
Number of PCREs	11193

Core and Thread Mapping

The mapping of software threads to hardware cores and threads for each platform is found in the following tables.

Table 12. Legend Used for Core Pinning

⬜	OS core (isolated)
🟧	VPP main and Snort main threads
🟦	VPP worker threads
🟩	Snort worker threads

Table 13. Core Mapping for Intel Xeon Gold 6338N (32C64T)

Logical Core 0				Logical Core 1			
⬜ 0	🟩 8	🟩 16	🟩 24	⬜ 0	🟩 8	🟩 16	🟩 24
🟧 1	🟩 9	🟩 17	🟩 25	🟧 1	🟩 9	🟩 17	🟩 25
🟦 2	🟩 10	🟩 18	🟩 26	🟦 2	🟩 10	🟩 18	🟩 26

3	11	19	27	3	11	19	27
4	12	20	28	4	12	20	28
5	13	21	29	5	13	21	29
6	14	22	30	6	14	22	30
7	15	23	31	7	15	23	31

Table 14. Core Mapping for Intel Xeon Gold 6428N (32C64T)

Logical Core 0				Logical Core 1			
0	8	16	24	0	8	16	24
1	9	17	25	1	9	17	25
2	10	18	26	2	10	18	26
3	11	19	27	3	11	19	27
4	12	20	28	4	12	20	28
5	13	21	29	5	13	21	29
6	14	22	30	6	14	22	30
7	15	23	31	7	15	23	31

Table 15. Core Mapping for Intel Xeon Platinum 8470N (52C104T)

Logical Core 0				Logical Core 1			
0	13	26	39	0	13	26	39
1	14	27	40	1	14	27	40
2	15	28	41	2	15	28	41
3	16	29	42	3	16	29	42
4	17	30	43	4	17	30	43
5	18	31	44	5	18	31	44
6	19	32	45	6	19	32	45
7	20	33	46	7	20	33	46
8	21	34	47	8	21	34	47
9	22	35	48	9	22	35	48
10	23	36	49	10	23	36	49
11	24	37	50	11	24	37	50
12	25	38	51	12	25	38	51

Table 16. Core Mapping for Intel Xeon Platinum 8380 (40C80T)

Logical Core 0				Logical Core 1			
0	10	20	30	0	10	20	30
1	11	21	31	1	11	21	31
2	12	22	32	2	12	22	32
3	13	23	33	3	13	23	33
4	14	24	34	4	14	24	34
5	15	25	35	5	15	25	35
6	16	26	36	6	16	26	36
7	17	27	37	7	17	27	37
8	18	28	38	8	18	28	38
9	19	29	39	9	19	29	39

Table 17. Core Mapping for Intel Xeon Platinum 8490H (60C120T)

Logical Core 0				Logical Core 1			
0	15	30	45	0	15	30	45
1	16	31	46	1	16	31	46
2	17	32	47	2	17	32	47
3	18	33	48	3	18	33	48
4	19	34	49	4	19	34	49
5	20	35	50	5	20	35	50
6	21	36	51	6	21	36	51
7	22	37	52	7	22	37	52
8	23	38	53	8	23	38	53
9	24	39	54	9	24	39	54
10	25	40	55	10	25	40	55
11	26	41	56	11	26	41	56
12	27	42	57	12	27	42	57
13	28	43	58	13	28	43	58
14	29	44	59	14	29	44	59

VPP Command Line and 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 /var/log/vpp.log
  exec vpp_setup
    cli-listen /run/vpp/vpp.sock
}
cpu {
  main-core 1
  corelist-workers 2,66
}
buffers { buffers-per-numa 1048576 }
dpdk {
  dev default {
    num-rx-queues 2
    num-tx-queues 2
    num-rx-desc 1024
    num-tx-desc 1024
  }
  # DPDK interfaces represent Home and External interfaces
  dev 0000:3d:00.0 { name eth1 }
  dev 0000:3d:00.1 { name eth0 }
  no-multi-seg
}
```

Figure 14. VPP Startup Config

```
set int state eth0 up
set int state eth1 up
set int ip address eth0 100.100.100.1/24
set int ip address eth1 200.200.200.1/24
set int promiscuous on eth0

ip route add 0.0.0.0/0 via 200.200.200.2 eth1
```

```
set ip neighbor eth0 100.100.100.2 b4:96:91:ad:8d:61
set ip neighbor eth1 200.200.200.2 40:a6:b7:19:06:f0

set dpdk rss key port 0 key
6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6d5a6
d5a6d5a
exec snort_setup
```

Figure 15. VPP Startup Config

Snort Command Line Parameters

Snort was invoked with the following command line:

```
% $SNORT_PATH/snort -A none --daq-dir $DAQ_DIR --daq vpp --daq-var input_mode=polling --daq-var
socket_path=/run/vpp -i vpp1:daq0 -c $LUA_DIR/snort.lua -R $RULES_DIR/snort-registered.rules --
tweaks max_detect -Q --max-packet-threads=1
```

Some of these parameters are described in more detail in Table 18.

Table 18. Snort Command Line Parameters

Parameter	Comments
-Q	Runs Snort in "inline" mode (intrusion prevention), as distinct from passive mode (intrusion detection)
--tweaks max_detect	Uses Maximum Detection base policy. As noted earlier in this paper, Maximum Detection base policy is not recommended for production deployment. We are investigating the impact on throughput of different base policies. Meanwhile, all benchmarks in this paper were carried out with this policy. Note that the *snort.lua* file has been updated to "include max_detect.lua" so this parameter does not need to be specified.
<none>	Hyperscan is enabled as the default detection engine. This is implemented via a patch to *snort.lua*.

Grub Command Line

```
rw hugepagesz=1G hugepages=16 isolcpus=1-59,61-119,121-179,181-239 default_hugepagesz=1G
rcu_nocbs=1-59,61-119,121-179,181-239 nohz_full=1-59,61-119,121-179,181-239 panic=30
init=/sbin/init net.ifnames=0 image_name=/images/pma-seed-20220510-082122.cpio.gz nmi_watchdog=0
audit=0 nosoftlockup hpet=disable mce=off tsc=reliable numa_balancing=disable
memory_corruption_check=0 workqueue.power_efficient=false module_blacklist=ast
modprobe.blacklist=ice,qat_4xxx,intel_qat init_on_alloc=0 initrd=/kernel/initrd.img-5.15.0-27-
generic
```

System Configuration

Table 19. DUT System Configuration – Network SKUs

Name	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N	Intel Xeon Platinum 8470N
Time	Wed Nov 9 12:21:01 PM UTC 2022	Thu Nov 17 09:44:04 AM UTC 2022	Mon Oct 31 04:38:38 AM UTC 2022
System	Supermicro SuperServer	Intel Corporation ArcherCity	Intel Corporation M50FCP
Baseboard	Supermicro X12DPG-QT6	Intel Corporation ArcherCity	Intel Corporation M50FCP
Chassis	Supermicro Main Server Chassis	Rack Mount Chassis	Rack Mount Chassis
CPU Model	Intel(R) Xeon(R) Gold 6338N CPU @ 2.20GHz	Intel(R) Xeon(R) Gold 6428N	Intel(R) Xeon(R) Platinum 8470N
Microarchitecture	ICX	SPR	SPR
Sockets	2	2	2
Cores per Socket	32	32	52
Hyper-threading	Enabled	Enabled	Enabled
CPUs	128	128	208
Intel Turbo Boost	Enabled	Disabled	Disabled

Name	Intel Xeon Gold 6338N	Intel Xeon Gold 6428N	Intel Xeon Platinum 8470N
Base Frequency	2.2GHz	1.8GHz	1.7GHz
All-core Maximum Frequency	2.7GHz	2.5GHz	2.7GHz
Maximum Frequency	3.5GHz	3.8GHz	1.7GHz
NUMA Nodes	2	2	2
Prefetchers	L2 HW, L2 Adj., DCU HW, DCU IP	L2 HW, L2 Adj., DCU HW, DCU IP	L2 HW, L2 Adj., DCU HW, DCU IP
Accelerators	QAT:0, DSA:0, IAA:0	QAT:0, DSA:2, IAA:0	QAT:8, DSA:8, IAA:0
Installed Memory	512GB (16x32GB DDR4 3200 MT/s [2666 MT/s])	512GB (16x32GB 4800 MT/s [4000 MT/s])	512GB (16x32GB 4800 MT/s [4800 MT/s])
Hugepagesize	1048576 KB	1048576 KB	1048576 KB
Transparent Huge Pages	madvise	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled	Disabled
Ethernet Adapter	1x Intel Ethernet Network Adapter E810-CQDA2	1x Intel Ethernet Network Adapter E810-CQDA2	1x Intel Ethernet Network Adapter E810-CQDA2
Disk	1x 223.6G INTEL SSDSC2BW240H6, 1x 240M Disk	1x 223.6G INTEL SSDSC2KB240G8	1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk
BIOS	1.4	EGSDREL1.SYS.9207.P03.2211041113	SE5C7411.86B.8805.D02.2209220021
Microcode	0xd000375	0x2b000111	0x2b000081
OS	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS	Ubuntu 22.04.1 LTS
Kernel	5.15.0-27-generic	5.15.0-27-generic	5.15.0-27-generic
TDP	185 watts	185 watts	300 watts
Power & Perf Policy	Performance	Performance	Performance
Frequency Governor	Performance	Performance	Performance
Frequency Driver	acpi-cpufreq	acpi-cpufreq	intel_pstate
Max C-state	1	1	1
SST-PP Profile	N/A	Auto (NFV)	Auto (NFV)/Compute

Table 20. DUT System Configuration - Top of Stack SKUs

Name	Intel Xeon Platinum 8380	Intel Xeon Platinum 8490H
Time	Wed Nov 23 12:25:22 PM UTC 2022	Tue Nov 22 06:07:53 AM UTC 2022
System	Supermicro SuperServer	Intel Corporation ArcherCity
Baseboard	Supermicro X12DPG-QT6	Intel Corporation ArcherCity
Chassis	Supermicro Main Server Chassis	Rack Mount Chassis
CPU Model	Intel(R) Xeon(R) Platinum 8380 CPU @ 2.30GHz	Intel(R) Xeon(R) Platinum 8490H
Stepping	D2	E5
Microarchitecture	ICX	SPR
Sockets	2	2
Cores per Socket	40	60

Name	Intel Xeon Platinum 8380	Intel Xeon Platinum 8490H
Hyper-threading	Enabled	Enabled
CPUs	160	240
Intel Turbo Boost	Disabled/Enabled	Disabled/Enabled
Uncore RAPL	Enabled/Disabled	Enabled/Disabled
Base Frequency	2.3GHz	1.9GHz
All-core Maximum Frequency	3.0GHz	2.9GHz
Maximum Frequency	2.3GHz	1.9GHz
NUMA Nodes	2	2
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:8, DSA:8, IAA:8, DLB:8
Installed Memory	256GB (16x16GB DDR4 3200 MT/s [3200 MT/s])	512GB (16x32GB DDR5 4800 MT/s [4800 MT/s])
Hugepagesize	1048576 KB	1048576 KB
Transparent Huge Pages	madvise	madvise
Automatic NUMA Balancing	Disabled	Disabled
Ethernet Adapter	1x Ethernet Network Adapter E810-C for QSFP	1x Ethernet Network Adapter E810-C for QSFP
Ethernet Adapter Firmware	4	4
Disk	1x 223.6G INTEL SSDSC2BW240H6	1x 223.6G INTEL SSDSC2KB240G8
BIOS	1.4	EGSDREL1.SYS.9207.P03.2211041113
Microcode	0xd000375	0x2b000111
OS	Ubuntu 22.04 LTS	Ubuntu 22.04 LTS
Kernel	5.15.0-27-generic	5.15.0-27-generic
TDP	270 watts	350 watts
Power & Perf Policy	Performance	Performance
Frequency Governor	Performance	Performance
Frequency Driver	intel_pstate	acpi-cpufreq
Max C-state	1	1

Baseline1: Test by Intel as of 11/9/22. 1-node, 2x Intel(R) Xeon(R) Platinum 6338N, 32 cores, HT On, Turbo On/Off, Total Memory 512GB (16x32GB 3200 MT/s [2667 MT/s]), BIOS 1.4, microcode 0xd000375, 1x Intel Ethernet Controller E810-CQDA2 , 1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk, Ubuntu 22.04 LTS, 5.15.0-27-generic, GCC 11.3, NGFW 21.09-1, score=10.84Gb/s

Baseline2: Test by Intel as of 11/23/22. 1-node, 2x Intel(R) Xeon(R) Platinum 8380, 40 cores, HT On, Turbo Off, Total Memory 256GB (16x16GB 3200 MT/s [3200 MT/s]), BIOS 1.4, microcode 0xd000375, 1x Intel Ethernet Controller E810-CQDA2 , 1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk, Ubuntu 22.04 LTS, 5.15.0-27-generic, GCC 11.3, NGFW 21.09-1, score=13.90Gb/s

New1: Test by Intel as of 10/31/22. 1-node, 2x Intel(R) Xeon(R) Platinum 8470N, 52 cores, HT On, Turbo Off, Total Memory 512GB (16x32GB 4800 MT/s [4800 MT/s]), BIOS SE5C7411.86B.8805.D02.2209220021, microcode 0x2b000081, 1x Intel Ethernet Controller E810-CQDA2 , 1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk, Ubuntu 22.04 LTS, 5.15.0-27-generic, GCC 11.3, NGFW 21.09-1, score=19.3Gb/s

New2: Test by Intel as of 11/17/22. 1-node, 2x Intel(R) Xeon(R) Platinum 6428N, 32 cores, HT On, Turbo On/Off, Total Memory 512GB (16x32GB 4800 MT/s [4000 MT/s]), BIOS EGSDREL1.SYS.9207.P03.2211041113, microcode 0x2b000111, 1x Intel Ethernet Controller E810-CQDA2 , 1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk, Ubuntu 22.04 LTS, 5.15.0-27-generic, GCC 11.3, NGFW 21.09-1, score=13.66Gb/s

New3: Test by Intel as of 11/22/22. 1-node, 2x Intel(R) Xeon(R) Platinum 8490H, 60 cores, HT On, Turbo Off, Total Memory 512GB (16x32GB 4800 MT/s [4800 MT/s]), BIOS EGSDREL1.SYS.9207.P03.2211041113, microcode 0x2b000111, 1x Intel Ethernet Controller E810-CQDA2

```
, 1x 223.6G INTEL SSDSC2KB240G8, 1x 240M Disk, Ubuntu 22.04 LTS, 5.15.0-27-generic, GCC 11.3,
NGFW 21.09-1, score=21.29Gb/s
```

Test Case Configuration

Table 21. Device Under Test Software Configuration

Software	Version
NGFW	22.09-01
VPP	v22.06.0-16
Snort	3.1.36.0
DAQ	3.0.9
LuaJIT	2.1.0-beta3
OpenSSL	1.1.1f 31 Mar 2020
libpcap	1.10.1 (with TPACKET_V3)
PCRE	8.45 2021-06-15
ZLIB	1.2.11
Hyperscan	5.4.0 2021-01-26
LZMA	5.2.5
Snort Rules	Snort Registered Rules
Pcap	EntmIx Pcap (captured from IxLoad)
Measurement Methodology	PDR (0.00001%)
DAQ mode	Polling
num-rx-queues	1 queue per VPP thread
num-tx-queues	1 queue per VPP thread
num-rx-descs	1024
num-tx-descs	1024
buffers-per-numa	1048576
DAQ queue-size	8192

Packet Generation Platform Configuration

Table 22. Packet Generation Platform Configuration

Software	Version
Compiler	gcc 9.4.0
pktgen	22.07.0

0223/DN/WIT/PDF 765277-001US