Solution Brief

Intel® QuickAssist Technology - Envoy TLS Acceleration with Intel® QAT

Author

Ismo Puustinen

Executive Summary

Envoy (https://www.envoyproxy.io/) is a popular L7 proxy. When Envoy is used as an edge proxy, it often must terminate lots of TLS (Transport Layer Security) connections. The RSA asymmetric cryptography operations needed for this can be accelerated using Intel® QuickAssist Technology (Intel® QAT).

This document is part of the Network Transformation Experience Kits.

Introduction

Envoy uses BoringSSL as the default TLS library. By default, the handshakes in Envoy are synchronous, meaning that the handshake function blocks the Envoy worker thread execution until the handshake has been completed. This will not work in a scheme such as Intel® QAT acceleration, because the Intel® QAT performance benefit comes from the fact that Envoy is ready to do more processing while the Intel® QAT device handles the cryptographic operations in parallel. If the Intel® QAT calls were synchronous, the performance benefit would not be there.

To facilitate asynchronous processing, Envoy has an extension type called "private key provider", which performs the following two main functions:

- Allows running custom code for private key signing and decrypting operations
- Allows asynchronous handshakes

The above are done using BoringSSL private key methods. BoringSSL private key methods are hooks in the TLS handshake processing, which allow external functions to be set for handling ECDSA sign operations and RSA sign and decrypt operations.

When the private key provider is used in Envoy, the server-side handshake function call can be configured to return immediately, and a callback is evoked when the handshake is ready to be completed, meaning that the cryptographic operation is ready.

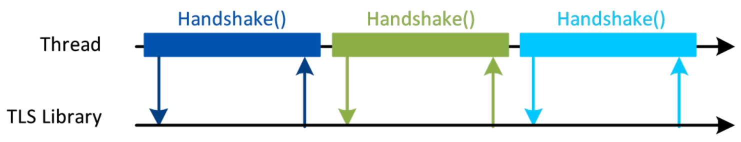

Figure 1. BoringSSL/TLS handshake in synchronous mode

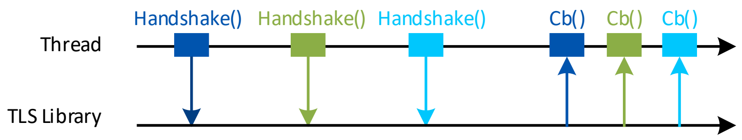

Figure 2. BoringSSL/TLS handshake in asynchronous mode

Private Key Providers

Envoy extensions in the main code tree are either core extensions or contrib extensions. The core extensions are part of the Envoy main container images, while the contrib extensions (along with the core extensions) are part of Envoy contrib container images. Now, Envoy code base has two contrib extensions for accelerating TLS handshakes using Intel® technologies: Intel® QAT (for Intel® QAT acceleration) and CryptoMB (for Intel® Advanced Vector Extensions 512 (Intel® AVX-512) acceleration).

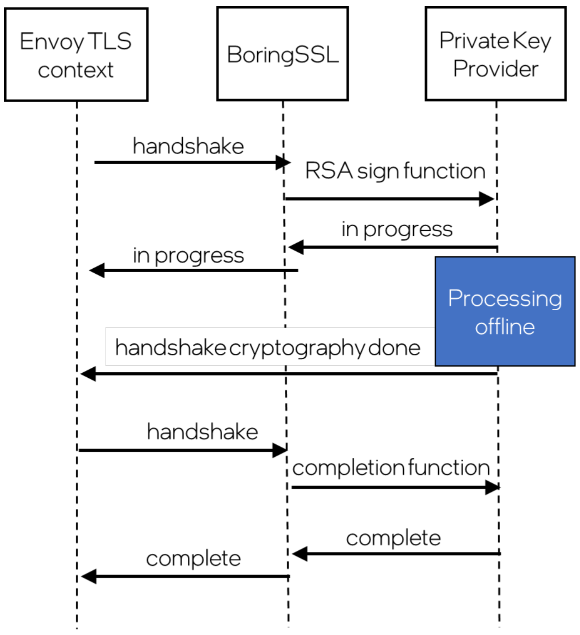

Figure 3. Envoy TLS private key provider handshake flow

CryptoMb private key provider uses Intel® AVX-512 multi-buffer instructions for accelerating handshakes. The Intel® AVX-512 instructions are present starting with the 3rd Gen Intel® Xeon® Scalable processors, and they do not require any special hardware enabling. Just running Envoy on a suitable platform and enabling the CryptoMb private key provider in the Envoy configuration is enough. The multi-buffer instructions gather several RSA operations into a shared buffer. When the 8-slot buffer is full or when a timer expires, the RSA operations are processed using SIMD (single instruction, multiple data) instructions, which provide greater throughput than processing the RSA operations separately. The downside of this approach is the potentially increased latency because operations may need to wait in the buffer before the processing can be done.

Envoy Intel® QAT Acceleration

Envoy TLS configuration can be done by two methods: either using direct configuration from a configuration file or using SDS (Secret Discovery Service) protocol for remotely configuring Envoy from an external control plane. Intel® QAT TLS acceleration can be enabled in both ways.

When using the direct configuration file method, the regular way for setting the private key is by adding it as a private_key field to Envoy's common_tls_context:

```
common_tls_context:
  tls_certificates:
  - certificate_chain:
      filename: ”/tmp/rsa-cert.pem”
    private_key:
      filename: ”/tmp/rsa-key.pem”
```

However, when Intel® QAT acceleration is required, the private_key field should be replaced with a suitably configured private_key_provider field:

```
common_tls_context:
  tls_certificates:
  - certificate_chain:
      filename: ”/tmp/rsa-cert.pem”
    private_key_provider:
      provider_name: qat
      typed_config:
        "@type":
"type.googleapis.com/envoy.extensions.private_k
ey_providers.qat.v3alpha.QatPrivateKeyMethodCon
fig"
         poll_delay: 0.002s
         private_key:
           filename: ”/tmp/rsa-key.pem”
```

The Intel® QAT private key provider configuration has two fields: poll_delay and private_key. The private_key field works as a regular Envoy DataSource type. The poll_delay field is a Duration type and specifies how often the Intel® QAT instance should be polled when waiting for an answer to Intel® QAT request. The right value depends on the tradeoff between CPU consumption and latency requirements and might require experimentation depending on the workload setup. A value of 0.002s (2 milliseconds) is a good starting point.

Kubernetes* Deployment of Intel® QAT-Accelerated Envoy

If Envoy is used in a Kubernetes environment, the Intel® QAT device plugin needs to be installed to the cluster. The device plugin finds which nodes have Intel® QAT support enabled and schedules Intel® QAT workloads on them (based on the Kubernetes deployment extended resource request). When the Intel® QAT-enabled Envoy is then scheduled on the node, the device plugin exposes one or more Intel® QAT VF devices to the Envoy container.

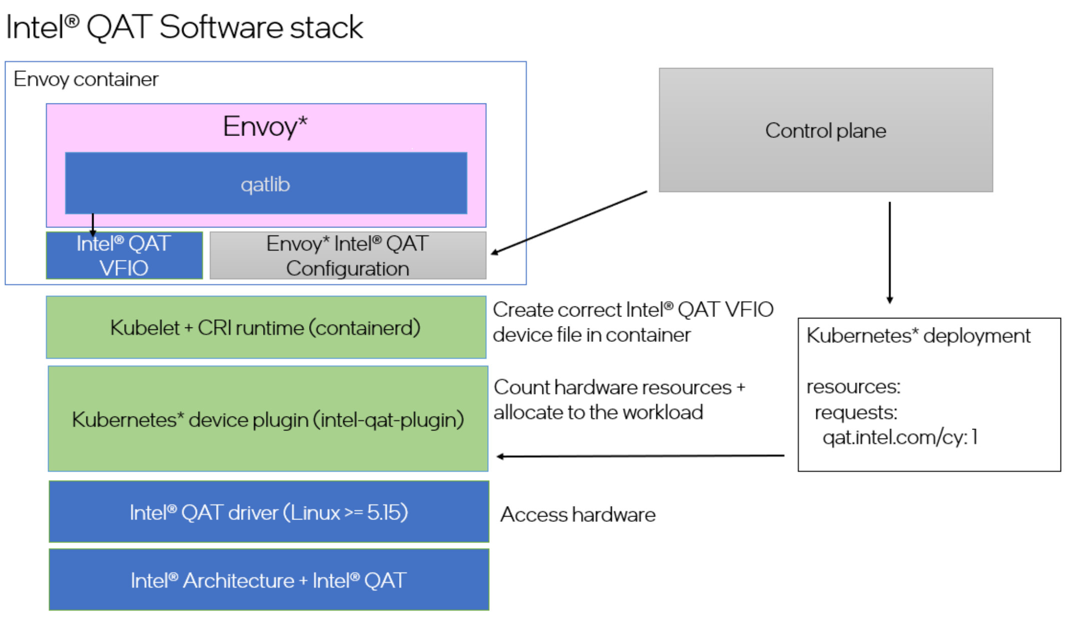
Intel® QAT Software stack
Envoy container
Envoy*
qatlib
Intel® QAT VFIO
Envoy* Intel® QAT Configuration
Control plane
Kubelet + CRI runtime (containerd)
Create correct Intel® QAT VFIO device file in container
Kubernetes* deployment
resources:
requests:
qat.intel.com/cy: 1
Kubernetes* device plugin (intel-qat-plugin)
Count hardware resources + allocate to the workload
Intel® QAT driver (Linux >= 5.15)
Access hardware
Intel® Architecture + Intel® QAT

Figure 4. Envoy SW stack enablement for Kubernetes

Note that if the number of Intel® QAT resources, which are allocated to the container is increased, there may be increased performance because Intel® QAT private key provider will automatically set up load balancing between the available Intel® QAT instances. If the Intel® QAT VF devices are pointing to different physical Intel® QAT endpoints, there is a speedup potential. However, if the Intel® QAT VF devices come from the same physical QAT endpoint, there will be no performance increase.

Intel® QAT Performance

The performance benefit from using Intel® QAT for TLS handshakes depends on many factors. Most important is simply the amount of asymmetric cryptography that needs to be done because that makes the cryptography acceleration have more effect in the overall performance. For example, if there are only a few new TLS connections per second or if the selected RSA key size is small, the acceleration possibilities are smaller. Conversely, if the RSA key size is large and there are many incoming RSA connections, the possibility for performance increase is bigger. Another thing to consider is the number of CPU threads Envoy is running on. On a smaller number of CPU cores, the performance benefit is easier to see since the acceleration leaves the CPU cores free to do other useful work needed for connection processing.

The performance impact has several components: the change in maximum throughput (requests / second), the change in latency (time required to complete a single operation), the change in CPU utilization, and the change in server power requirements.

Test Setup

The following performance numbers were measured on a pre-production 4th Gen Intel® Xeon® Scalable processor. An HTTPs load generator (K6) running on another node was creating a specific number of new TLS connections per second to the Envoy proxy. Every connection was created with a new TLS handshake–session reuse was not happening in this setup. The Envoy proxy was running pinned on a limited number of CPU hyper threads. All CPU hyper threads selected where sibling threads. For example, 8 hyper threads would come from 4 physical cores.

Envoy was configured to test three configurations: Intel® QAT private key provider with a single Intel® QAT VF device, CryptoMb private key provider utilizing Intel® AVX-512 multi-buffer TLS acceleration, and default configuration with no TLS acceleration. For each test, Envoy was using a 2048-bit RSA key with X25519 key exchange protocol. Envoy was configured to return an HTTP 200 response code with an empty body for each HTTPs response. The throughput numbers show that when processing only new TLS connections, Intel QAT can increase the number of connections that can be processed. For example, with Envoy pinned to four CPUs, the throughput is 3.6x better when using Intel® QAT than with using non-accelerated Envoy.

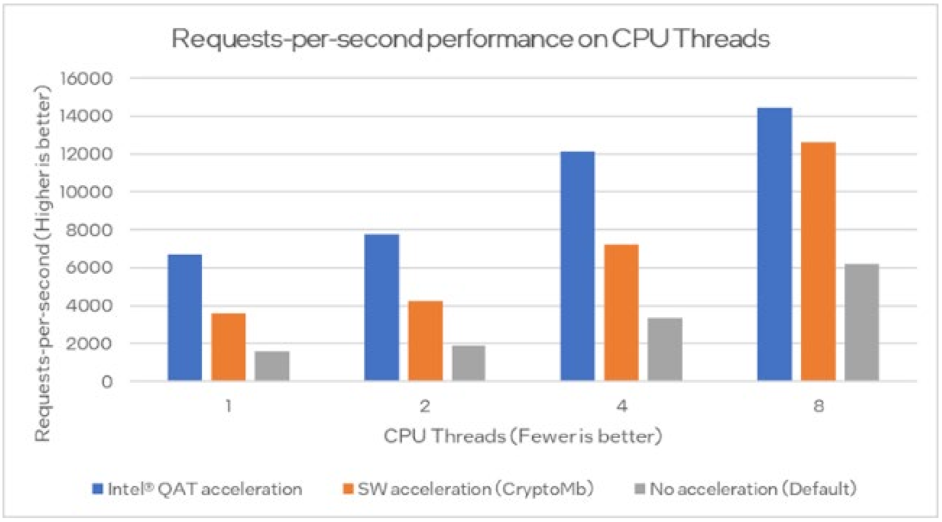

Figure 5. Requests-per-second performance on CPU threads

The latency measurements and CPU usage charts are from the benchmark where Envoy is pinned to four CPU cores. The very last value of each latency graph (where latency rises sharply) is removed from the graph, because latency measurement is not meaningful on a system with maximum load.

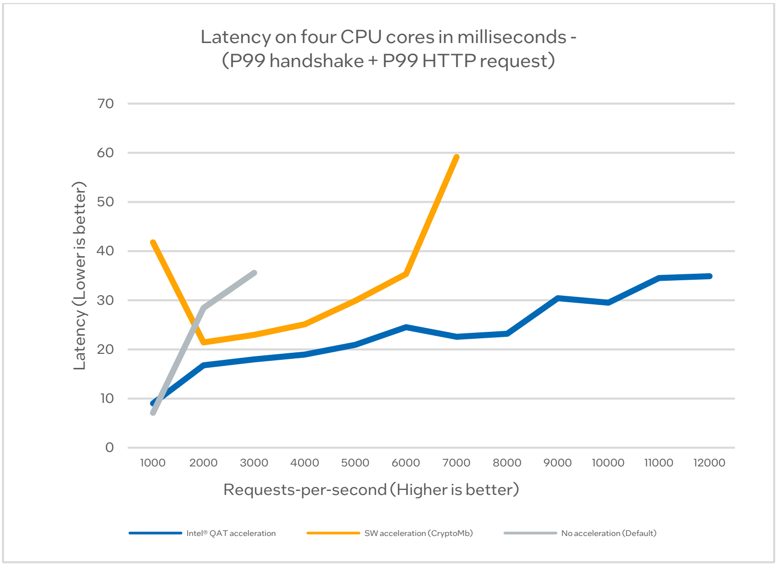

Figure 6. Latency on four CPU cores in milliseconds (P99 handshake + P99 HTTP request)

However, when the values below the knee point are considered, Intel® QAT has in most cases the lowest latency, and the difference is improved when the number of incoming new TLS connections increases.

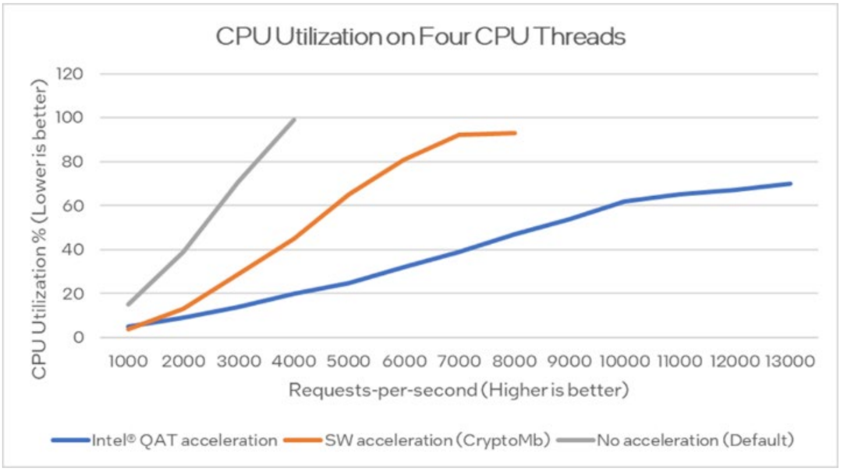

Figure 7. CPU Utilization on four CPU threads

The CPU utilization on Figure 7 shows that when Envoy is not accelerated, the CPU usage scales in a linear way until it reaches the saturation point at almost one hundred percent CPU load. However, when Intel® QAT acceleration is used, the CPU load stays much lower. For example, at 2,000 requests / second, the difference in CPU load on the four cores is 30 percentage points. The saved CPU cycles can be used for other Envoy processing, such as running HTTP filters or other traffic processing.

Test Deployment

Table 1. Software and hardware used for benchmarking

CPU	2S Pre-production Intel® Xeon® Scalable Processor/ Intel® Xeon® Platinum 8480+
Platform	Pre-production Intel reference platform
OS	Ubuntu* 20.04.4 LTS
Kernel	5.19.1-051901-generic
Microcode	0x89000060
Base Frequency	2.0GHz
Hyper-threading	On
Cores per Socket	56
Turbo	Enabled
Ethernet Adapter Summary	Ethernet Controller XXV710 for 25GbE SFP28
Drive Summary	INTEL® SSDPELKX020T8 (2TB), 2 * INTEL SSDPF21Q800GB (800 GB)
Installed memory	1 Ti total, 32*32GB DIMM, 4800 MT/s configured as 4400 MT/s
Intel QAT driver	Linux 5.19.1 upstream driver
Envoy	Envoy contrib binary built from upstream git, SHA 3e2f1507851b511689cdc07fd3aecf714912ba8a, (main branch, Aug 31, 2022)
Test by	Intel
Test date	Sep 2022

Summary

Envoy supports Intel® QAT for accelerating TLS handshakes. The performance benefit varies depending on the use case, but Intel® QAT can help in reducing CPU usage, reducing individual request latency, and increasing throughput. The Intel® QAT to Envoy support needs to be enabled by a configuration file change or by a dynamic Envoy listener configuration over the xDS protocol. In addition to that, the Envoy container must have Intel® QAT resources added to it by configuring Kubernetes cluster accordingly.

Terminology

Table 2. Terminology

Abbreviation	Description
CPU	Central processing unit
ECDSA	Elliptic Curve Digital Signature Algorithm
Intel® AVX-512	Intel® Advanced Vector Extensions 512
Intel® QAT	Intel® QuickAssist Technology
RSA	Rivest–Shamir–Adleman – A public-key cryptosystem
SDS	Secret Discovery Service
SIMD	Single Instruction Multiple Data
TCP/IP	Transmission Control Protocol/Internet Protocol
TLS	Transport Layer Security

References

Table 3. References

Reference	Source
Envoy Intel QAT API	https://www.envoyproxy.io/docs/envoy/latest/api-v3/extensions/private_key_providers/qat/v3alpha/qat.proto
Intel QAT landing page	https://www.intel.com/content/www/us/en/architecture-and-technology/intel-quick-assist-technology-overview.html
Intel QAT installation guide	https://github.com/intel/qatlib/blob/main/INSTALL
Intel QAT device plugin	https://intel.github.io/intel-device-plugins-for-kubernetes/cmd/qat_plugin/README.html
Envoy benchmarking guide	https://www.envoyproxy.io/docs/envoy/latest/faq/performance/how_to_benchmark_envoy

Document Revision History

Revision	Date	Description
001	January 2023	Initial release.

0123/DN/WIT/PDF 767057-001US