Intel® QuickAssist Technology

Performance Optimization Guide

Revision 008

December 2021

Document Number: 330687-008

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

All information provided here is subject to change without notice. Contact your Intel representative to obtain the latest Intel product specifications and roadmaps.

All product plans and roadmaps are subject to change without notice.

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.

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software, or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at intel.com.

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.

Contents

1 Introduction 6

1.1 Terminology 6

1.2 Where to Find Current Software and Documentation 7

1.2.1 Product Documentation 7

1.2.2 Documentation Conventions 7

2 Intel® QuickAssist Technology Software Overview 8

3 Software Design Guidelines 9

3.1 Polling vs. Interrupts (If Supported) 9

3.1.1 Interrupt Mode 9

3.1.2 Polling Mode 10

3.1.3 Epoll Mode 11

3.1.4 Recommendations 11

3.2 Use of Data Plane (DP) API vs. Traditional API 11

3.2.1 Batch Submission of Requests Using the Data Plane API 12

3.3 Synchronous (sync) vs. Asynchronous (async) 12

3.4 Buffer Lists 13

3.5 Maximum Number of Concurrent Requests 13

3.6 Symmetric Crypto Partial Operations 14

3.7 Reusing Session in QAT Environment 14

3.8 Backpressure Mechanism 14

3.9 Load Balancing Within a QAT Endpoint 14

3.10 Best Known Method (BKM) for Avoiding PCI Performance Bottlenecks 15

4 Application Tuning 16

4.1 Platform-Level Optimizations 16

4.1.1 BIOS Configuration 16

4.1.2 Core Selection 16

4.1.3 Memory Configuration 16

4.1.4 Payload Alignment 16

4.1.5 NUMA Awareness 18

4.2 Intel® QuickAssist Technology Optimization 18

4.2.1 Disable Services Not Used 18

4.2.2 Using Embedded SRAM (if supported) 18

4.2.3 Disable Parameter Checking 18

4.2.4 Adjusting the Polling Interval 18

4.2.5 Reducing Asymmetric Service Memory Usage 19

Figures

Figure 1. Packet Decrypt and Encrypt 17

Tables

Table 1. Terminology 6
Table 2. Reference Documents 7

Revision History

Document Number	Revision Number	Description	Revision Date
330687	008	• Updated the document with New Intel Logo • Updated the content with Intel One text font	December 2021
330687	007	• Updated Section 3.1.3, Epoll Mode • Updated Section 3.1.4, Recommendations Updated Section 3.2, Use of Data Plane (DP) vs. Traditional API. • Updated note. • Updated Section 3.3, Synchronous (sync) vs. Asynchronous (async) • Added Section 3.7, Reusing Session in QAT API • Added Section 3.8, Backpressure Mechanism • Updated Section 4.2.5, Reducing Asymmetric Service Memory Usage. • Added note	August 2019
330687	006	• Deleted Figure 1, Performance Impact of Multiple Buffers	December 2018
330687	005	• Removed references to using coalescing timer and to Intel® Communications Chipset 8900 to 8920 series.	September 2018
330687	004	• Updates to interrupt and epoll modes, other minor technical changes	January 2017
330687	003	• Minor updates throughout. Added Section 3.2.3, Epoll Mode and Section 3.2.4, Recommendations • Updated Section 4.2.6, Reducing Asymmetric Service Memory Usage	October 2015
330687	002	• Updated Section 4.1.3, Payload Alignment	May 2015
330687	0.1	• Initial release	September 2014

§

1 Introduction

This performance optimization guide for Intel® QuickAssist Technology can be used both during the architecture/design phases and the implementation/integration phases of a project that involves the integration of the Intel® QuickAssist Technology software with an application stack.
Accordingly, the guide is divided into two main sections:

- **Software Design Guidelines** – Architecture and design guidelines on how best to integrate the Intel® QuickAssist Technology software into the application software stack. Trade-offs between various design choices are described together with recommended approaches.
- **Application Tuning** – Guidelines to further increase the performance of Intel® QuickAssist Technology in the context of a full application.

The intended audience for this document includes software architects, developers and performance engineers.
In this document, for convenience:
Acceleration drivers is used as a generic term for the software that allows the QuickAssist Software Library APIs to access the Intel® QuickAssist Accelerator(s) integrated in the following devices:

- Intel® Communications Chipset 8900 to 8920 Series
- Intel® Communications Chipset 8925 to 8955 Series
- Intel® Atom® processor C2000 product family
- Intel® Atom® processor C3000 product family
- Intel® C620 Series Chipsets
- Intel® Xeon® D-1500 processor
- Intel® Xeon® D-2100 processor

1.1 Terminology

Table 1. Terminology

Term	Description
C-States	C-States are advanced CPU current lowering technologies.
ECDH	Elliptic Curve Diffie-Hellman
IA	Intel® architecture CPU
Intel® SpeedStep® Technology	Advanced means of enabling very high performance while also meeting the power-conservation needs of mobile systems.
LAC	LookAside Crypto
Latency	The time between the submission of an operation via the QuickAssist API and the completion of that operation.
MSI	Message Signaled Interrupts
NUMA	Non Uniform Memory Access
Offload Cost	This refers to the cost, in CPU cycles, of driving the hardware accelerator. This cost includes the cost of submitting an operation via the Intel® QuickAssist API and the cost of processing responses from the hardware.

Term	Description
PCH	Platform Controller Hub
PKE	Public Key Encryption
Throughput	The accelerator throughput usually expressed in terms of either requests per second or bytes per second.

1.2 Where to Find Current Software and Documentation

Associated software and collateral can be found on the open source website: https://01.org/intelquickassist-technology

Table 2 includes a list of related documentation.

1.2.1 Product Documentation

Documentation includes:

- Using Intel® Virtualization Technology (Intel® VT) with Intel® QuickAssist Technology Application Note (this document)
- Additional related documents listed in Table 2.

Table 2. Reference Documents

Document Name	Document Number/ Location
Intel® QuickAssist Technology API Programmer's Guide	330684
Intel® QuickAssist Technology Cryptographic API Reference Manual	330685
Intel® QuickAssist Technology Data Compression API Reference Manual	330686
Intel® QuickAssist Technology for Linux* Release Notes	330683
Intel® QuickAssist Technology Software for Linux* - Programmer's Guide	336210
Intel® QuickAssist Technology for Linux* Getting Started Guide	336212

1.2.2 Documentation Conventions

The following conventions are used in this manual:

- `Courier font`- code examples, command line entries, API names, parameters, filenames, directory paths, and executables
- **Bold text** - graphical user interface entries and buttons

§

2 Intel® QuickAssist Technology Software Overview

This chapter provides a very brief overview of the Intel® QuickAssist Technology software. It is included here to set the context for terminology used in later sections of this document. More details are available in the Programmer's Guide for your platform (refer to Table 1).

The Intel® QuickAssist Technology API supports two acceleration services:

- Cryptographic
- Data Compression.

The acceleration driver interfaces to the hardware via hardware-assisted rings. These rings are used as request and response rings. Rings are grouped into banks (16 rings per bank). Request rings are used by the driver to submit requests to the accelerator and response rings are used to retrieve responses back from the accelerator. The availability of responses can be indicated to the driver using either interrupts or by having software poll the response rings.

At the Intel® QuickAssist Technology API, services are accessed via "instances." A set of rings is assigned to an instance and so any operations performed on a service instance will involve communication over the rings assigned to that instance.

§

3 Software Design Guidelines

This chapter focuses on key design decisions that should be considered, in order to achieve optimal performance, when integrating with the Intel® QuickAssist Technology software. In many cases the best Intel® QuickAssist Technology configuration is dependent on the design of the application stack that is being used and so it is not possible to have a "one configuration fits all" approach. The trade-offs between differing approaches will be discussed in this section to help the designer to make an informed decision.

The guidelines presented here focus on the following performance aspects:

- Maximizing throughput through the accelerator
- Minimizing the offload cost incurred when using the accelerator
- Minimizing latency

Each guideline will highlight its impact on performance. Specific performance numbers are not given in this document since exact performance numbers depend on a variety of factors and tend to be specific to a given workload, software, and platform configuration.

3.1 Polling vs. Interrupts (If Supported)

Note: Not all use cases support interrupt mode, and not all software packages support interrupt mode.

Software can either periodically query the hardware accelerator for responses or it can enable the generation of an interrupt when responses are available. Interrupts or polling mode can be configured per instance via the platform-specific configuration file. Configuration parameter details are available in the Programmer's Guide for your platform (refer to Table 1).

The properties and performance characteristics of each mode are explained below followed by recommendations on selecting a configuration.

3.1.1 Interrupt Mode

When operating in interrupt mode, the accelerator will generate an MSI-X interrupt when a response is placed on a response ring. Each ring bank has a separate MSI-X interrupt which may be steered to a particular CPU core via the CoreAffinity settings in the configuration file.

To reduce the number of interrupts generated, and hence the number of CPU cycles spent processing interrupts, multiple responses can be coalesced together. The presence of the multiple responses can be indicated via a single coalesced interrupt rather than having an interrupt per response. The number of responses that are associated with a coalesced interrupt is determined by an interrupt coalescing timer. When the accelerator places a response in a response ring, it starts an interrupt coalescing timer. While the timer is running, additional responses may be placed in the response ring. When the timer expires, an interrupt is generated to indicate that responses are available. Details on how to configure interrupt coalescing are available in the Programmer's Guide for your platform (refer to Table 1).

Since interrupt coalescing is based on a timer, there is some variability in the number of responses that are associated with an interrupt. The arrival rate of responses is a function of the arrival rate of the associated requests and of the request size. Hence, the timer configuration needed to coalesce X large requests is different from the timer configuration needed to coalesce X small requests. It is recommended that the timer is tuned based on the average expected request size.

The choice of timer configuration impacts throughput, latency, and offload cost:

- Configuring a very short time period maximizes the throughput through the accelerator, minimizing latency, but will increase the offload cost since there will be a higher number of interrupts and hence more CPU cycles spent processing the interrupts. If this interrupt processing becomes a performance bottleneck for the CPU, the overall system throughput will be impacted.
- Configuring a very long timer period leads to reduced offload cost (due to the reduction in the number of interrupts) but increased latency. If the timer period is very long and causes the response rings to fill, the accelerator will stall and throughput will be impacted.

The appropriate coalescing timer configuration will depend on the characteristics of the application. It is recommended that the value chosen is tuned to achieve optimal performance.

When using interrupts with the user space Intel® QuickAssist Technology driver, there is significant overhead in propagating the interrupt to the user space process that the driver is running in. This leads to an increased offload cost. Hence it is recommended that interrupts should not be used with the user space Intel® QuickAssist Technology driver.

3.1.2 Polling Mode

In polled mode, interrupts are fully disabled and the software application must periodically invoke the polling API, provided by the Intel® QuickAssist Technology driver, to check for and process responses from the hardware. Details of the polling API are available in the Programmer's Guide for your platform (refer to Table 1).

The frequency of polling is a key performance parameter that should be fine-tuned based on the application. This parameter has an impact on throughput, latency and on offload cost:

- If the polling frequency is too high, CPU cycles are wasted if there are no responses available when the polling routine is called. This leads to an increased offload cost.
- If the polling frequency is too low, latency is increased and throughput may be impacted if the response rings fill causing the accelerator to stall.

The choice of threading model has an impact on performance when using a polling approach. There are two main threading approaches when polling:

- Creating a polling thread that periodically calls the polling API. This model is often the simplest to implement, allows for maximum throughput, but can lead to increased offload cost due to the overhead associated with context switching to/from the polling thread.
- Invoking the polling API and submitting new requests from within the same thread. This model is characterized by having a "dispatch loop" that alternates between submitting new requests and polling for responses. Additional steps are often included in the loop such as checking for received network packets or transmitting network packets. This

approach often leads to the best performance since the polling rate can be easily tuned to match the submission rate so throughput is maximized and offload cost is minimized.

3.1.3 Epoll Mode

This mode effectively replaces user space Interrupt Mode, which has been deprecated.

The mode can only be used in user space. The following must be considered if opting to use this mode (such as, over the standard polling mode in user space).

Because epoll mode has two parts, of which the kernel space part utilizes the legacy interrupt mode, if there is a delay in the kernel interrupt (such as, by changing the coalescing fields), there will be a corresponding increase in latency in the delivery of the event to user space.

The thread waiting for an event in epoll mode does not consume CPU time, but the latency could have an impact on the performance. For higher packet load where the wait time for the next packet is insignificant, polling mode is recommended.

You are limited to one instance (and one process) per bank in epoll mode.

3.1.4 Recommendations

Polling mode tends to be preferred in cases where traffic is steady (such as packet processing applications) and can result in a minimal offload cost. Epoll mode is preferred for cases where traffic is bursty, as the application can sleep until there is a response to process.

Considerations when using polling mode:

- Fine-tuning the polling interval is critical to achieving optimal performance.
- The preference is for invoking the polling API and submitting new requests from within the same thread rather than having a separate polling thread.

Considerations when using epoll mode:

- CPU usage will be at 0% in idle state in epoll mode versus a non-zero value in standard poll mode. However, with a high load state, standard poll mode should out-perform epoll mode.
- You are limited to one instance (and one process) per bank in epoll mode.

3.2 Use of Data Plane (DP) API vs. Traditional API

The cryptographic and compression services provide two flavors of API, known as the traditional API and the Data Plane API. The traditional API provides a full set of functionality including thread safety that allows many application threads to submit operations to the same service instance. The Data Plane API is aimed at reducing offload cost by providing a “bare bones” API, with a set of constraints, which may suit many applications. Refer to the Intel® QuickAssist Technology Cryptographic API Reference Manual for more details on the differences between the DP and traditional APIs for the crypto service.

From a throughput and latency perspective, there is no difference in performance between the Data Plane API and the traditional API.

From an offload cost perspective, the Data Plane API uses significantly fewer CPU cycles per request compared to the traditional API. For example, the cryptographic Data Plane API has an offload cost that is lower than the cryptographic traditional API.

Note: One constraint with using the Data Plane API is that interrupt mode is supported only if one bank is served by only one thread.

3.2.1 Batch Submission of Requests Using the Data Plane API

The Data Plane API provides the capability to submit batches of requests to the accelerator. The use of the batch mode of operation leads to a reduction in the offload cost compared to submitting the requests one at a time to the accelerator. This is due to CPU cycle savings arising from fewer writes to the hardware ring registers in PCIe* memory space.

Using the Data Plane API, batches of requests can be submitted to the accelerator using either the `cpaCySymDpEnqueueOp()` or `cpaCySymDpEnqueueOpBatch()` API calls for the symmetric cryptographic data plane API and using either the `cpaDcDpEnqueueOp()` or

`cpaDcDpEnqueueOpBatch()` API calls for the compression data plane API. In all cases, requests are only submitted to the accelerator when the `performOpNow` parameter is set to `CPA_TRUE`.

It is recommended to use the batch submission mode of operation where possible to reduce offload cost.

3.3 Synchronous (sync) vs. Asynchronous (async)

The Intel® QuickAssist Technology traditional API supports both synchronous and asynchronous modes of operation. The Intel® QuickAssist Technology Data Plane API only supports the asynchronous mode of operation.

With synchronous mode, the traditional Intel® QuickAssist Technology API will block and not return to the calling code until the acceleration operation has completed.

With asynchronous mode, the traditional or Data Plane Intel® QuickAssist Technology API will return to the calling code once the request has been submitted to the accelerator. When the accelerator has completed the operation, the completion is signaled via the invocation of a callback function.

From a performance perspective, the accelerator requires multiple inflight requests per acceleration engine to achieve maximum throughput. With synchronous mode of operation, multiple threads must be used to ensure that multiple requests are inflight. The use of multiple threads introduces an overhead of context switching between the threads which leads to an increase in offload cost.

Hence, the use of asynchronous mode is the recommended approach for optimal performance.

3.4 Buffer Lists

The symmetric cryptographic and compression Intel® QuickAssist Technology APIs use buffer lists for passing data to/from the hardware accelerator. The number and size of elements in a buffer list has an impact on throughput; performance degrades as the number of elements in a buffer list increases. To minimize this degradation in throughput performance, it is recommended to keep the number of buffers in a buffer list to a minimum. Using a single buffer in a buffer list leads to optimal performance. See also 4.1.4Payload Alignment for additional considerations.

Note: Specific performance numbers are not given in this document since exact performance numbers depend on a variety of factors and tend to be specific to a given workload, software and platform configuration.

When using the Data Plane API, it is possible to pass a flat buffer to the API instead of a buffer list. This is the most efficient usage of system resources (mainly PCIe* bandwidth) and can lead to lower latencies compared to using buffer lists.

In summary, the recommendations for using buffer lists are:

- If using the Data Plane API, use a flat buffer instead of a buffer list.
- If using a buffer list, a single buffer per buffer list leads to highest throughput performance.
- If using a buffer list, keep the number of buffers in the list to a minimum.

3.5 Maximum Number of Concurrent Requests

The depth of the hardware rings used by the Intel® QuickAssist Technology driver for submitting requests to, and retrieving responses from, the accelerator hardware can be controlled via the configuration file using the `CyXNumConcurrentSymRequests`, `CyXNumConcurrentAsymRequests` and `DcXNumConcurrentRequests` parameters. These settings can have an impact on performance:

- As the maximum number of concurrent requests is increased in the configuration file, the memory requirements required to support this also increases.
- If the number of concurrent requests is set too low, there may not be enough outstanding requests to keep the accelerator busy and so throughput will degrade. The minimum number of concurrent requests required to keep the accelerator busy is a function of the size of the requests and of the rate at which responses are processed via either polling or interrupts (refer to 3.1Polling vs. Interrupts (If Supported) for more details).
- If the number of concurrent requests is set too high, the maximum latency will increase.

It is recommended that the maximum number of concurrent requests is tuned to achieve the correct balance between memory usage, throughput and latency for a given application. As a guide the maximum number configured should reflect the peak request rate that the accelerator must handle.

3.6 Symmetric Crypto Partial Operations

The symmetric cryptographic Intel® QuickAssist Technology API supports partial operations. This allows a single payload to be processed in multiple fragments with each fragment corresponding to a partial operation. The Intel® QuickAssist Technology API implementation will maintain sufficient state between each partial operation to allow a subsequent partial operation for the same session to continue from where the previous operation finished.

From a performance perspective, the cost of maintaining the state and the serialization between the partial requests in a session has a negative impact on offload cost and throughput. To maximize performance when using partial operations, multiple symmetric cryptographic sessions must be used to ensure that sufficient requests are provided to the hardware to keep it busy.

For optimal performance, it is recommended to avoid the use of partial requests if possible.

There are some situations where the use of partials cannot be avoided since the use of partials and the need to maintain state is inherent in the higher level protocol (such as, the use of the RC4 cipher with an SSL/TLS protocol stack).

3.7 Reusing Session in QAT Environment

The session is the entry point to perform symmetric cryptography with the QAT device. Every session has assigned algorithm, state, instance, but also allocated memory space.

If you are limited with the number of instances and want to run several different algorithms or change keys for another session, de-initialize the session and create a new one. However, such an approach impacts performance because it involves buffer disposal, deinitialization of the instance, and so on.

Instead, the session can be reused with updating only a direction (encryption / decryption), key or symmetric algorithm to be used. This method will not dispose buffers and can reduce the CPU cycles significantly.

3.8 Backpressure Mechanism

On FreeBSD, there is a possibility to save CPU cycles by getting information about a ring with few CPU cycles. If the ring is full, the user can push back and does not have to execute request submission. Since request submission evaluates the possibility to submit to a ring at the end of the call stack, many CPU cycles are used up with packet preparation.

3.9 Load Balancing Within a QAT Endpoint

When using the Intel® QuickAssist Technology API with the Intel® Communications Chipset 8900 to 8920 Series or Intel® Atom® processor C2000 product family, optimal throughput performance is achieved when operations are load balanced across the engines within a QAT endpoint.

For example, for a top-SKU Intel® Communications Chipset 8900 to 8920 Series device, which has four cryptographic engines and two compression engines, a minimum of four cryptographic service instances and two compression service instances are required to maximize performance.

Note: Intel® Communications Chipset 8925 to 8955 Series and later products have multiple cryptographic and compression engines, but the hardware can load balance and provide full performance using only one service instance for cryptographic operations and one service instance for compression operations.

It is also recommended to assign each service instance to a separate CPU core to balance the load across the CPU and to ensure that there are sufficient CPU cycles to drive the accelerators at maximum performance.

When using interrupts, the core affinity settings within the configuration file should be used to steer the interrupts for a service instance to the appropriate core.

Detailed guidelines on load balancing and how to ensure maximum use of the available hardware capacity are available in the Programmer's Guide for your platform (refer to Table 1).

3.10 Best Known Method (BKM) for Avoiding PCI Performance Bottlenecks

For optimal performance, ensure the following:

- All data buffers should be aligned on a 64-byte boundary.
- Transfer sizes that are multiples of 64 bytes are optimal.
- Small data transfers (less than 64 bytes) should be avoided. If a small data transfer is needed, consider embedding this within a larger buffer so that the transfer size is a multiple of 64 bytes. Offsets can then be used to identify the region of interest within the larger buffer.
- Each buffer entry within a Scatter-Gather-List (SGL) should be a multiple of 64bytes and should be aligned on a 64-byte boundary.
- Ensure that enough PCIe* lanes are connected between the acceleration device and the root port and ensure that these lanes are training to the expected width and speed.
- Modify your application so it retries in an optimal manner. This is important for small data packets. The backoff timer in the sample code may be used as an example of such "back off" implementation.

§

4 Application Tuning

This chapter describes techniques you may employ to optimize your application.

4.1 Platform-Level Optimizations

This section describes platform-level optimizations required to achieve the best performance.

4.1.1 BIOS Configuration

In some cases, maximum performance may only be achieved with the following BIOS configuration settings:

CPU Power and Performance:

- Intel® SpeedStep® technology is disabled
- All C-states are disabled
- Max CPU Performance is selected

4.1.2 Core Selection

Using physical cores as opposed to hyper threads may result in higher performance.

4.1.3 Memory Configuration

Ensure that memory is not a bottleneck. For instance, ensure that all CPU nodes have enough local memory and can take advantage of available memory channels.

4.1.4 Payload Alignment

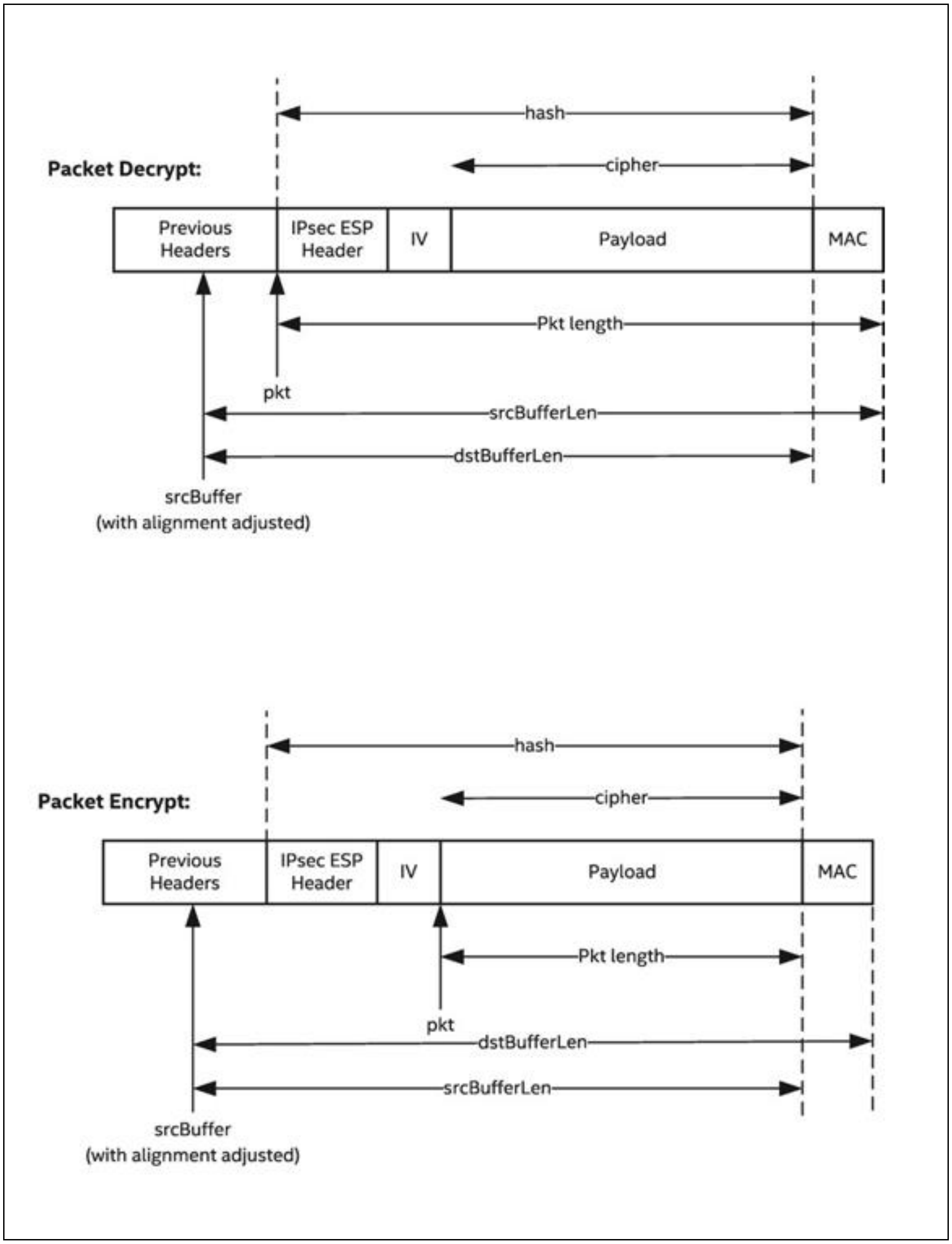
For optimal performance, data pointers should be at least 8-byte aligned. In some cases, this is a requirement. Refer to the API for details.

For optimal performance, all data passed to the Intel® QuickAssist Technology engines should be aligned to 64B. The *Intel® QuickAssist Technology Cryptographic API Reference* and the *Intel® QuickAssist Technology Data Compression API Reference* manuals (refer to Table 2) document the memory alignment requirements of each data structure submitted for acceleration.

Note: The driver, firmware, and hardware handle unaligned *payload* memory without any functional issue but performance will be impacted.

Note: It is common that packet payloads will not be aligned on a 64B boundary in memory, as the alignment usually depends upon which packet headers are present. In general, the mitigation for handling this is to adjust the buffer pointer, length and cipher offsets passed to hardware to make the pointer aligned. This works on the assumption that there is a point in the packet, before the payload, that is 64B aligned. See the diagram below for an illustration of adjusted alignment in the context of encrypt/decrypt of an IPsec packet.

Figure 1. Packet Decrypt and Encrypt



4.1.5 NUMA Awareness

For a dual processor system, memory allocated for data submitted to the acceleration device should be allocated on the same node as the attached acceleration device. This is to prevent having to fetch data for processing on memory of the remote node.

4.2 Intel® QuickAssist Technology Optimization

This section references parameters that can be modified in the configuration file or build system to help maximize throughput and minimize latency or reduce memory footprint. Refer to Table 2, Programmer's Guide, for your platform for detailed descriptions of the configuration file and its parameters.

4.2.1 Disable Services Not Used

The compression service, when enabled, impacts the throughput performance of crypto services at larger packet sizes and vice versa. This is due to partitioning of internal resources between the two services when both are enabled. It is recommended to disable unused services.

Note: With the Intel® Communications Chipsets 8900 to 8920 series, to use Wireless Firmware you must enable the dc service, even though it is not used.

4.2.2 Using Embedded SRAM (if supported)

Note: This section is relevant only for Intel® Communications Chipset 8900 to 8920 Series and Intel® Atom® processor C2000 product family software.

Embedded SRAM can be used to reduce the PCIe* transactions that occur during dynamic compression sessions. This will have a small positive impact on performance, but also frees up PCIe* bandwidth for other services.

Embedded SRAM can be set using the configuration file parameter:

```
dcTotalSRAMAvailable = 524288
```

The default value is 0, the maximum value is 1024 x 512 (524288).

4.2.3 Disable Parameter Checking

Parameter checking results in more Intel® architecture cycles consumed by the driver. By default, parameter checking is enabled. This is controlled by `ICP_PARAM_CHECK`, which can be set as an environment variable or it can be controlled with the configure script option, if available.

4.2.4 Adjusting the Polling Interval

This section describes how to get an indication of whether your application is polling at the right frequency. As described in Section 3.1.2Polling Mode the rate of polling will impact latency, offload cost and throughput. Also, Section 3.1.2Polling Mode describes two ways of polling:

- Polling via a separate thread.
- Polling within the same context as the submit thread.

With option 1, there is limited control over the poll interval, unless a real time operating system is employed. With option 2, the user can control the interval to poll based on the number of submissions made.

Whichever method is employed, the user should start with a low frequency of polling, and this will ensure maximum throughput is achieved. Gradually increase the polling interval until the throughput starts to drop. The polling interval just before throughput drops should be the optimal for throughput and offload cost.

This method is only applicable where the submit rate is relatively stable and the average packet size does not vary. To allow for variances, a larger ring size is recommended, but this in turn will add to the maximum latency.

4.2.5 Reducing Asymmetric Service Memory Usage

This section only applies to the Intel® Atom® processor C2000 product family and Intel® Communications Chipset 8900 to 8920 Series.

This section describes how to reduce the memory footprint required by using the asymmetric crypto API.

The asymmetric cryptographic service requires a far larger memory pool compared to symmetric cryptography and compression. The more logical instances that are defined in the configuration file, the more memory is consumed by the driver. This memory usage may be unnecessary if only the symmetric part of the cryptographic service is used. Alternatively, the memory requirement may be reduced depending on the user’s requirements on the asymmetric service.

Option 1: Asymmetric Not Required

The minimum value for `CyXNumConcurrentAsymRequests` in the configuration file is 64, where `x` = the instance number.

Large values of the above configuration file parameter increase memory requirements and more instances will also cause increases. To minimize memory, the user should:

- Set `max_mr = 1` (at compile time)
- Set `LAC_PKE_MAX_CHAIN_LENGTH = 1`
- Minimize the number of logical crypto instances

Option 2: Reduce Prime Miller Rabin Rounds

By default, the driver uses 50 rounds of Miller Rabin to test primality. If the user does not require this amount of prime testing, the following environment variable can be set at build time to reduce this:

Set `max_mr = NUM_ROUNDS`

where `NUM_ROUNDS` is between 1 and 50.

Option 3: No Prime or ECDH Services Required

At build time:

- Set `max_mr = 1`
- In `<ICP_ROOT>/quickassist/lookaside/access_layer/src/common/crypto/asym/include/lac_pke_utils.h, set LAC_PKE_MAX_CHAIN_LENGTH = 1`

§