Performance Enhancement for IPsec Processing on Multi-Core Systems

Sandeep Malik
Freescale Semiconductor India Pvt. Ltd
IDC Noida, India

Ravi Malhotra
Freescale Semiconductor India Pvt. Ltd
IDC Noida, India

Abstract — Most of the existing crypto drivers cannot operate effectively over the multi-core systems, resulting in high synchronization overheads. Moreover, the crypto drivers under utilize the full capacity of the hardware device underneath, resulting in low IPsec throughput.

This paper discusses a design methodology that virtualizes the access to the common crypto accelerators across the multiple cores and leverages the existing feature of Linux to design a more balanced system. The new design can utilize the hardware capabilities in a better and efficient way to improve the overall crypto performance on multi-core systems.

I. INTRODUCTION

With the increased usage of Internet and availability of high network bandwidth, more and more IP traffic flows across various networks. The task of securing the traffic is computation intensive, as the traffic needs to encrypted and decrypted to ensure its security. The security related processing can be offloaded using the security co-processors also termed as crypto engines.

The usage of these crypto engines has been prevalent in the industry from quite some time for offloading security related processing. The motivation to write this paper came when we studied the different implementations being used in the security drivers for these crypto engines.

These crypto engines can be divided among two categories namely Flow-Through accelerators and Look-Aside accelerators. The Flow-Through accelerators perform the cryptographic operations on data as it is flowing from one end to other end. These types of accelerators are generally built in form of ASIC or cores with the packet classification as part of hardware with some configurability. The Look-Aside accelerators are the devices wherein cryptographic operations are supported by the presence of software driven entity which performs the packet classification as a pre-requisite for security processing.

In this paper we will talk about the Look-Aside accelerators and their driver implementation.

When we analyzed the implementation of various open source drivers for the crypto engines, we found issues related to the multi-core platforms in most of the designs.

II. PROBLEMS WITH EXISTING SECURITY DRIVERS

Before discussing the issues, let’s take a look at how IPsec processing is done in Linux. In Linux, the security processing is done in two parts, pre-processing and post-processing. In the first part called pre-processing, the incoming packets are matched against the existing Security Policy Database (SPD). In case a matching policy exists, next step is to find what kind of security transformation needs to be applied on the incoming packet. All this information is stored in a Security Association (SA) and these associations together form an entity named Security Association Database (SAD).

This paper explains the issues found in the existing crypto engine drivers and the approaches that can address these issues and help you design drivers that can deliver maximum throughput in the multi-core environment.

To validate the design approaches, Freescale’s P2020[1] platform which comes with a crypto engine named SEC v3.1 is used. This platform has two PowerPC cores running at 1200MHz with System Bus running at 600MHz and DDR operating at 400MHz. This paper also explains the incremental performance enhancements that were observed when these design approaches were tested on the P2020 platform from Freescale.

Figure 1. Linux IPsec overview with Crypto Engine.
This information is now passed to the underlying crypto engine hardware. This information passing is done with the help of another abstraction layer called Crypto API, which provides an abstraction over various underlying crypto engines.

Figure 1 illustrates various blocks involved in any security related transformation for IPsec processing in Linux.

The crypto engine starts processing the packet as soon as it receives the request in form of a buffer along with security algorithm, keys etc. to be used to apply a security transformation. Once the crypto engine finishes the processing it sends notification back to the core either via an interrupt or by setting some bit which is being polled by the core.

The core receives the packet after crypto engine finishes the processing on the passed buffer. The processed packet again undergoes same steps of SPD/SA lookup and if require re-submission to crypto engine to perform another set of security operations. Once this processing is finished the packet is finally transmitted on the network.

Following are the issues that are found in the existing security drivers:

- Resource sharing and synchronization overheads
- Pre-processing and post-processing at different priorities
- Allocating and de-allocating buffers at job submission and retrieval

A. Resource Sharing and Synchronization Overheads

Most of the crypto engines provide different resources like job queues, interrupts etc. that are shared among different cores. Since these resources are being shared among cores some synchronization mechanism is required for accessing these resources, resulting in synchronization overheads. Apart from that, pre-processing and post-processing gets distributed in different cores, resulting in cache thrashing, which in-turn affect the IPsec performance.

Since the ideas referred in this paper were verified on P2020, paper will outline the techniques used to improve the performance by virtualization of the access to the shared resources available in P2020. But all the techniques used to address the problems in P2020 platform are generic and can be applied to other platforms on need basis.

Let’s look at the crypto engine used in P2020. The P2020 crypto engine had only one interrupt enabled, which interrupts the core when the job given to the crypto engine finishes. Now in case of SMP Linux, this interrupt can go to any of the cores based on the factors, such as system load or affinity of the interrupt. Assuming a fairly loaded system, we found that the interrupt dynamically goes to Core0 and Core1. This makes necessary for the driver to use spinlocks to provide proper synchronization between the cores while submitting the buffer to crypto engine and while retrieving the processed buffer from the crypto engine. Apart from that, since the interrupt can go to any of the core, one core ended up doing pre-processing and post-processing for its own packets as well as post-processing for the packets submitted by other core. This eventually leads to cache-thrashing as the whole packet context needs to be transferred to the other core.

Figure 2 shows how cache thrashing and spinlocks hamper the IPsec performance.

B. Pre-processing and Post-processing at Different Priorities

The IPsec processing consists of two parts pre-processing and post-processing. Almost all the crypto drivers implement the post-processing part using the tasklet context whereas pre-processing is done in different context generally NAPI[2] which runs under the netifrx soft_irq and has higher priority than tasklet. Fig.3 shows the diagram of a dual core system having a Look-Aside crypto accelerator for offloading the IPsec processing.
Because of this difference in priorities, the queue of processed buffers starts building. In this case if the traffic is pumped at a very high rate the buffer queue builds up and eventually the packets drop occurs.

C. Allocating and De-allocating Buffers at Job Submission and Retrieval

Most of the drivers need to pass along with the buffer, certain information called job descriptor required by the crypto engine to do the desired security processing. This job descriptor is allocated before submitting a packet and once the crypto engine finishes the processing on the submitted buffer this job descriptor needs to be freed. This eventually leads to one allocation and free operation for every packet which impacts the overall performance of the system.

III. SOLUTIONS FOR THESE PROBLEMS

This section will cover the design techniques that can be used to address these issues and design the driver to deliver maximum throughput in a multi-core environment.

This paper also shares the incremental performance enhancement that was observed when these ideas were implemented and tested on P2020 platform from Freescale.

A. Resource Sharing and Syncronization Overheads

While doing the performance analysis using oprofile[3], it was found that lots of CPU cycles are used by the synchronization activities that need to be performed for accessing the resources shared among the cores.

First experiment was to split the IPsec pre-processing and post-processing jobs into two different cores so that one core does the pre-processing and the other core does the post-processing. So the affinity for the crypto engine interrupt is set to one core and the affinity for all rx interrupts for the Ethernet controller is set to other core as shown in Figure 4. The rationale behind this experiment was that by doing this packet processing load will be equally balanced between the cores although the cache-thrashing will still be there in the system.

By doing this there was performance enhancement of around 13-14% over the case where in no affinity was set for crypto engine and Ethernet controller interrupts. Figure 5 summarizes the performance improvement seen for various packet sizes when the affinities were set.

But as this solution cannot remove the spinlocks in a generic way, we also need to virtualize the access to the crypto engine. To virtualize the access to the crypto engine, the two interrupts of the crypto engine were enabled and the job queues (channels) were partitioned such that each core has its own dedicated channel for submitting a job and receiving back the processed job. Now, as each core has its own queue for job submission and the interrupts of the processed job are also sent to the same core, the issue of cache-thrashing is resolved. The complete processing for a packet from the time of reception till the time the IPsec processed packet leaves the system is done in the same core, as shown in Figure 6.
Figure 6. Virtualizing access by partitioning the job queues.

Figure 7 summarizes the performance improvement seen for various packet sizes when the access to the job queues of the crypto engine was virtualized.

B. Pre-processing and Post-processing at different priorities

As described in the problem section that because of the execution of pre-processing and post-processing in different contexts, the queues of processed buffers build up when the traffic is pumped at a very high rate. There are multiple ways to avoid this in Linux. We will discuss few of these in this section.

One of the solutions to avoid this issue is to replace the existing tasklet implementation with the high tasklet so that post-processing can be done frequently and the processed buffer queues do not get a chance to build up.

However, this solution has following issues associated with it:

- The high tasklet is generally not recommended for these types of jobs.

- This solution does not help design a balanced system. This is because, in this solution, the post-processing is done more frequently than pre-processing.

- This solution does not deal with the interrupts raised by the crypto engines in case the device is configured in interrupt mode operation, or the resources consumed in case the device is configured in polling mode.

To deal with these issues we used NAPI for the post-processing part. When the post-processing is done in the NAPI context, processed buffers are not queued and the system works as shown in Figure 8. In addition, this results in an overall improvement in the system performance.

Benchmarking results shows that there is around 10-15% performance enhancement in the IPsec system throughput for smaller packet size. Moreover, usage of NAPI helps in ironing out the issues of queue building in the system and helps in reducing the overheads of the interrupts. So now the system does both pre-processing as well as post-processing in the NAPI context.

Figure 9 summarizes how performance enhances for various packet sizes when the NAPI is used for post-processing part in the system.
C. Allocation and Freeing Buffers at Job submission and retrieval

At every job submission and retrieval, memory allocation and freeing routines, `kmalloc()` and `kfree()`, are called. On profiling the system with the `oprofile` tool, we found out that lots of CPU cycles are used by the calls to these routines.

To resolve this issue, we created a recycle queue for allocating and de-allocating buffers, when required. Now, when there is a request for the new buffer, first the recycle queue is checked. If the buffer is available in the recycle queue, it is assigned from the recycle queue. Otherwise, the new buffer is requested using `kmalloc()`.

When the crypto engine finishes the security processing and the processed buffer is being freed, the freed buffer is sent back to the recycle queue. Due to this, the future requests for buffer allocation can be served from the recycle queue instead of a call to `kmalloc()`.

However, on implementing this solution, the IPsec performance did not improve as expected. With further profiling, we found out that since recycle queue is shared between cores and spinlocks are used for synchronization. Because of this, the recycle queue is not able to improve the performance of system.

To resolve the issue, we implemented separate recycle queue for each core, so that synchronization is not required, as shown in Figure 10.

This results in performance enhancement of 2-3 KPPS. Figure 11 summarizes the comparison between the non-NAPI and NAPI with recycle queue per core systems.

IV. CONCLUSION

This paper describes various techniques to improve the IPsec system performance in a multi-core system, and evaluates them on a dual core platform P2020 from the Freescale QoriQ P2 platform series. By using the design approaches explained in this paper, IPSec system throughput for the smaller packet sizes can be improved by around 25-30%. And the CPU utilization for smaller packets size can be reduced to around 10-15%.

REFERENCES