國立交通大學

資訊科學研究所

碩士論文

基於網路處理器架構之虛擬私人網路閘道器：實作與探討

VPN Gateways over Network Processors:

Implementation and Evaluation

研 究 生： 林權宏

指導教授： 林盈達 教授

中華民國九十三年六月

基於網路處理器架構之虛擬私人網路閘道器：實作與探討

VPN Gateways over Network Processors: Implementation and Evaluation

研 究 生： **林權宏**　　Student : Chiuan-Hung Lin

指導教授： **林盈達**　　Advisor : Ying-Dar Lin

國 立 交 通 大 學

資 訊 科 學 研 究 所

碩 士 論 文

A Thesis

Submitted to Institute of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2004

HsinChu, Taiwan, Republic of China

中華民國九十三年六月

基於網路處理器架構之虛擬私人網路閘道器：實作與探討

學生 ： 林權宏　　指導教授 ：林盈達

國立交通大學資訊科學系

摘要

今日的網路應用像是虛擬私人網路(VPN)所需要的計算能力已超過了處理器的能力，而網路處理器是有別於特定功能積體電路的另一選擇。網路處理器可程式化其額外的處理器來幫核心處理器卸載不同的處理需求以及能夠快速地依軟體變更作修改。在本論文中，我們利用 IXP425 這個擁有 3 個網路處理引擎以及內嵌硬體處理器的網路處理器來實做出一個使用網際網路安全性協定(IPsec)之虛擬私人網路閘道器，並卸載其上之封包收送和密碼學運算至網路處理器上。我們實作出之原形機在 3DES 之 IPsec 下可達 45Mbps，較純軟體運算之吞吐量多了 350%。另外經由內部測試，我們發現無論封包傳送跟網際網路安全處理的瓶頸都還是在核心處理器上。

關鍵字： 網路處理器，虛擬私人網路，IXP425，瓶頸，效能

VPN Gateways over Network Processors: Implementation and Evaluation

Student: Chiuan-Hung Lin Advisor: Dr. Ying-Dar Lin

Department of Computer and Information Science

National Chiao Tung University

Abstract

Networking applications nowadays such as virtual private network are demanded for more computing power than the original processor, which network processor architecture is an alternative solution different to ASIC. In network processor architecture, additional processors could be programmed as various functions to offload computation from the core processor and adapt for current modification quickly. In this work, we clarify the details of IXP425 network processor, which has three network processor engines with integrated hardware co-processors, and explain the implementation steps to develop an IPsec-based VPN gateway over IXP425 with offloading for packet transferring and cryptographic calculation. Our prototype can reach 45Mhz for IPsec tunnel with 3DES algorithm, which has 350% improvement to only core processor. Through internal benchmarks, we found the performance bottlenecks on both packet forwarding and IPsec processing are still the core processor.

Keyworks: Network Processor, VPN, IXP425, bottleneck, throughput

Contents

List of Figures

Chapter 1. Introduction

Today's networking applications, such as virtual private network (VPN) [1] and content filtering [2] that offer extra security and application-aware processing, have demanded more powerful hardware devices to achieve high performance. The most straight-forward way to tackle this problem is to increase the clock rate of a general purpose processor, though some disadvantages, such as the cost and the technology limit, accompany. Moreover, the low efficiency is also expected since the processor, as its name suggests, is not specifically designed for the processing of networking packets.

Another solution to this problem is to employ the concept of *offloading*, that is, to shift the computing-intensive tasks from the core processor to a number of additional processors. The Application-Specific Integrated Circuit (ASIC) [3] has been a possible candidate to serve as an additional processor. Nonetheless, this workaround might not be preferred in two aspects. First, since the functionalities are fixed once tapped out, it needs to be redesigned for any modifications. Second, the development period is so time-consuming that the time-to-market requirement may not be met.

Network processors [4] are now embraced as an alternative solution to remedy the above-mentioned problems because of its re-programmability, specifically designed instructions for networking purpose and the hardware threads with minimal, if not zero, context switch overhead. In this work, we explored the feasibility of implementing VPN, which is a computation intensive application, over the Intel IXP425 [5] network processor featuring an XScale [6] core, *multiple hardware contexts* and coprocessors, and tried to figure out the performance and possible bottlenecks of the implementation. The VPN mechanism, which is usually based on IPsec [7], comprises several processing stages such as packet reception(Rx) and transmission(Tx), encryption and decryption, authentication and table

lookups, each of which needs a certain amount of processing. We analyzed the detailed packet flow and decided to offload packet transferring and cryptographic calculation to coprocessors. Some efforts have also been done to port the VPN application from ordinary PC to IXP425 in the meantime. We then externally and internally benchmarked the resulting prototype. The former characterized performance figures of the implementation, while the latter carried out the in-depth analysis of the observations which were left unexplained in the external benchmarks such as system bottlenecks. The Xscale is identified to be the bottleneck for IPsec processing.

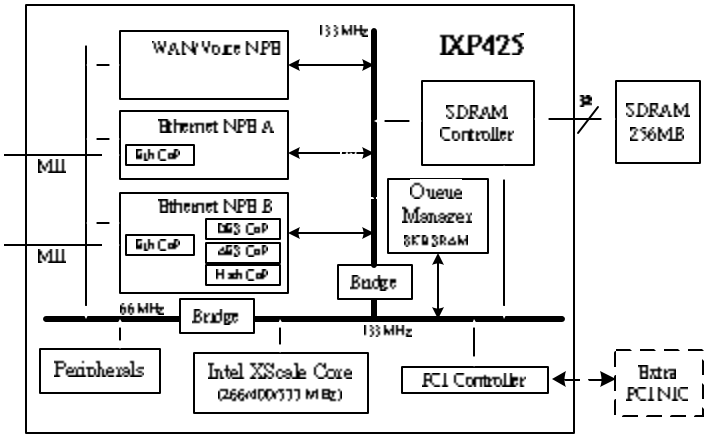
Some related works researching the bottlenecks of network processors can also be found in the literature: Spalink et al. [8] presented the results of simple IP forwarding and Lin et al. [9] implemented DiffServ, both over Intel IXP1200 [10]. Nevertheless, our work differs from theirs in that (1) no coprocessor was involved in their implementations; (2) both the control-plane and part of the data-plane processing were handled in the core processor of IXP425 while the core of IXP1200 took care of the control-plane packets only, and (3) computation intensive VPN application was considered, as compared with simple forwarding and memory intensive classification of these two studies.

This article is organized as follows. Chapter 2 describes the hardware and software architectures of IXP425. Chapter 3 elaborates the details of the design and implementation of VPN over IXP425. Chapter 5 presents the results and observations in the external and internal benchmarks. Some conclusive remarks of this article are made in chapter 5.

Chapter 2. Hardware and Software Architecture of IXP425

2.1 Hardware Architecture of IXP425

The hardware block diagram of IXP425 is depicted in Fig. 1 [11]. The core of IXP425 is a 533MHz XScale processor which furnishes whole system to work from initializing to executing overall software objects. Furthermore, three buses interconnected by two bridges provide the connectivity among components on IXP425.



context switch overhead further makes NPEs more tolerant to long memory accesses and thus reduces the number of processor stalls. The communications between the XScale core and NPEs is taken care of by a hardware queue manager using interrupt and message queue mechanisms. Besides, the queue manager also contains 8KB SRAM divided into 64 independent queues manipulated as circular buffers for allocating free memory space to the incoming packets and for locating those packets. The SDRAM can be expanded up to 256MB for storing tables, policies and OS applications in addition to packets. A PCI interface is available for an additional PCI NIC.

Some peripheral controllers, like USB and UART controllers, are also equipped into IXP425 for better extensibility.

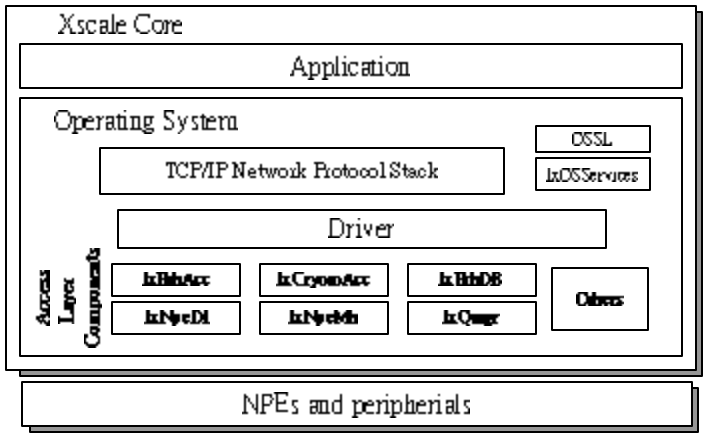
2.2 Detailed Packet Flow in IXP425

The processing flow of an ordinary packet is elaborated below, which also referred to Fig.1. Upon the arrival of a packet at the interface of an NPE, it is partitioned into several 32byte segments and stored at the Receive FIFO of an Ethernet coprocessor which in turn performs MAC-related operations. The NPE then moves those segments into corresponding addresses in SDRAM allocated by the queue manager which then notifies the XScale of the reception by interrupts for further processing. During normal processing procedures such as IP and other higher layer protocol stacks at XScale, chances are that some authentic and cryptographic operations are needed. The XScale core may handle them either by itself or by *offloading* the computation overhead to appropriate coprocessors residing in NPE B. In the latter scenario, the coprocessors are directly invoked by NPE B to process a certain data segment in SDRAM requested by the XScale, where a message queue implemented in the queue manager is exploited to pass the request. The queue manager is informed by NPE B upon the completion of the operations and then interrupts the XScale.

2.3 Software Architecture of IXP425

As shown in Fig. 2, the software architecture [12] is divided into two portions, namely the platform independent (applications and some higher level components such as networking protocol stacks in OS) and dependent parts (mainly device drivers). This design is helpful especially when an OS migration from a H/W platform to IXP425 is demanded, that is, the developers may need to focus only on the dependent part. When it comes to device driver implementation, a set of software libraries collectively referred to as *AccessLibrary* can be used to drive devices such as NPEs, coprocessors, peripherals, etc. The AccessLibrary also provides utilities to implement some OS-related functions such as mutual exclusion.

The software processing flow is described as follows. During the boot time the IxNpeDl is called to download the corresponding code image into the instruction cache of each NPE. Then IxQmgr and IxNpeMh are called to initialize the queue manager as well as the message handler responsible for the communication between NPEs and XScale. The IxEthAcc and IxEthDB are used to receive Ethernet frames, while IxCryptoAcc is incorporated for possible cryptographic operations during packet processing.

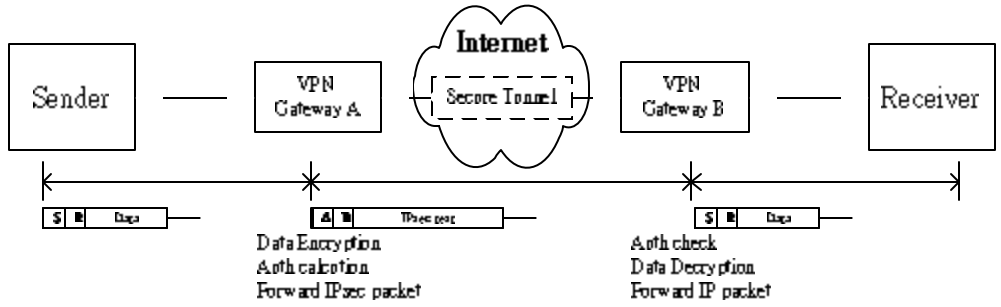


Chapter 3. Design and Implementation

In this chapter, we first introduce basic operations in a VPN environment and then analyze its packet processing flow in order to identify possible bottlenecks as offloading candidates. Finally, we describe how to implement a VPN gateway over IXP425.

3.1 VPN Briefing

Virtual Private Network (VPN) provides secure transmission over un-trusted networks. Oftentimes the IPSec protocol is adopted as the underlying technique thanks to the popularity of the Internet Protocol. As Fig. 3 shows, it supports data authentication, integrity and confidentiality, in which two VPN gateways are employed to serve as nodes that construct tunnels for secure data transmission without user awareness. Since the bandwidth between networks depends mainly on the processing capability of gateways if the bandwidth of internet is enough, improving the performance of them will be critical to the whole VPN throughput.



To resolve this performance problem, we analyze the VPN packet processing flow in order to identify possible candidates that can be offloaded to coprocessors. A detailed incoming IPsec packet flow was shown in Fig.4. It consists of three main blocks, namely the packet reception, IPsec processing and packet transmission, whose operations are elaborated below.

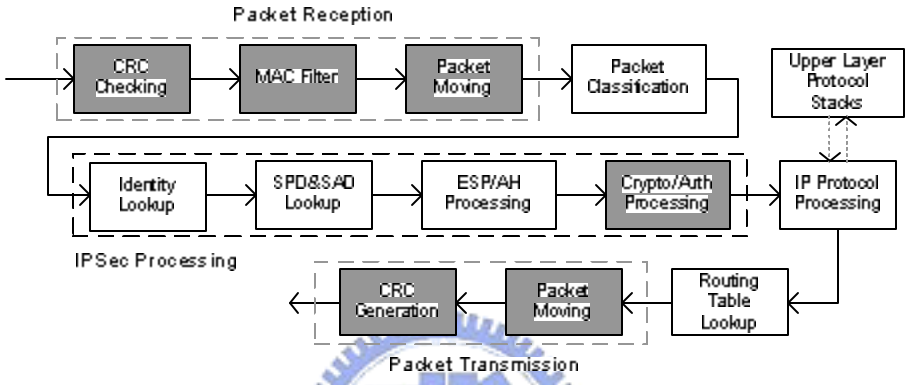


Fig. 4. Processing flow of an IPsec packet. Shaded blocks are stages that need to be offloaded.

Once an Ethernet frame is received by the physical interface, checking for frame check sequence sieves out broken frames and then remaining frames are filtered in accordance with possible destination MAC address configurations. Reception step ends after the frame is moved into memory, followed by a classification recognizing it as an IPSec packet. At this time, some table lookups for processing rules and cryptographic parameters proceed and payload of this IPsec packet is decrypted or checked for authentication according to its parameters. Afterwards, a new packet resulting from the original IP payload may proceed further processing of higher-level protocols, or be transmitted as a normal packet according to the routing table.

Tasks suitable to be offloaded to coprocessors can be identified by two characteristics: whether they are repeated routines or computation intensive ones. As mentioned earlier this chapter, we know that IPsec processing is computation intensive, especially the cryptographic

operations. Hence, we decide to pick the cryptographic processing as an offloading candidate. The packet transfer, CRC checking/generation, MAC filtering and packet movement between NPE and memory, is chosen as another candidate since the procedures are exact the same for every packet. Some IPsec processing except data encryption/decryption is not adopted as offloading candidates since the packet processing steps and IPsec databases are different in according to design of OS. From the hardware block diagram in chapter two, it is clear that IXP425 meets well the requirements of the identified candidates.

3.3 Implementation

We adopt NetBSD [13], a secure, highly portable and open-source OS derived from 4.4BSD, as our target operating system. Clean design between platform dependent and independent parts, makes it a good implementation target for new platform. Following relates three major parts in want of modification to operate on IXP425.

3.3.1 Operating System Porting

The most efficient way to porting an OS to a new platform [14] is to refer to the port of another similar platform and then implement drivers for devices of the target platform. To port NetBSD over IXP425, we adopt the "EvbARM" port in NetBSD, which supports for evaluation boards based on XScale and other ARM-based core processors so that only some system-level modifications, for example the CPU identification, setup of board-specific memory map, and system initialization procedures, have to be done to enable normal operations on IXP425.

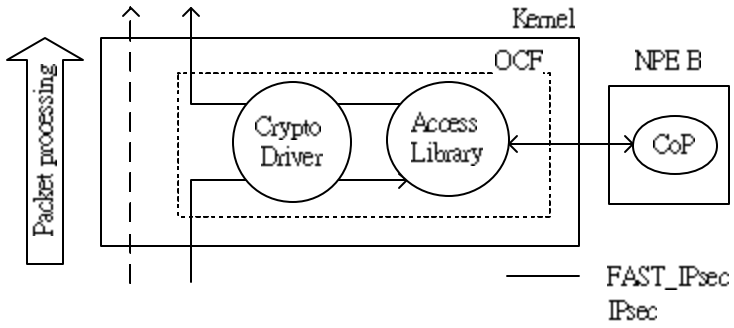
3.3.2 Driver Development

A number of drivers for devices such as UART, NPEs and coprocessors need to be

implemented for communications between the operating system and those devices. This effort is alleviated with the help of the AccessLibrary introduced in chapter two. Besides drivers, we also have to modify those two OS dependent modules, namely OSSL and IxOSServices, to ensure proper operations of the OS-related services.

3.3.3 Offloading the Cryptographic Operations

The last modification to kernel regards the offloading of original in-kernel IPsec cryptographic computations from XScale to NPE. In addition to the ordinary method requiring the kernel (and therefore the core processor) to perform whole encryption/decryption operations, NetBSD provides another option named *FAST_IPsec* which makes use of the Open Crypto Framework, OCF, for offloading. In OCF, the cryptographic operations can be handled by a *registered* function. The most significant difference between the original IPsec and Fast_IPsec is that the flow in later would not suspend on data cryptographic process and engage the work with other thread running as background process. We exploit this design by pre-registering the crypto driver, which drives the crypto coprocessor through the AccessLibrary functions, in the OCF so that all cryptographic overhead is shifted from the core to the coprocessor in NPE B. Figure 5 diagramed the concept of FAST_IPsec with OCF support.



Chapter4. System Benchmark and Bottleneck Analysis

In this chapter, we investigate the benefits from offloading by externally benchmarking the implementation using various offloading schemes. A problem regarding the unbalanced load between NPE A and NPE B is also addressed and analyzed. Finally, a number of internal tests are conducted in order to identify possible bottlenecks in the system.

4.1 System Benchmark Setup

4.1.1 Offloading Schemes Design

To have a better understanding of the improvement from the network processor architecture as well as the offloading mechanism on packet Rx/Tx and IPsec processing, we design and benchmark systems of different offloading schemes, and compare their performance results. Figure 6 shows the four offloading schemes, we can disable one or both of them to imitate other possible offloading schemes. The throughput results are listed and explained in next section.

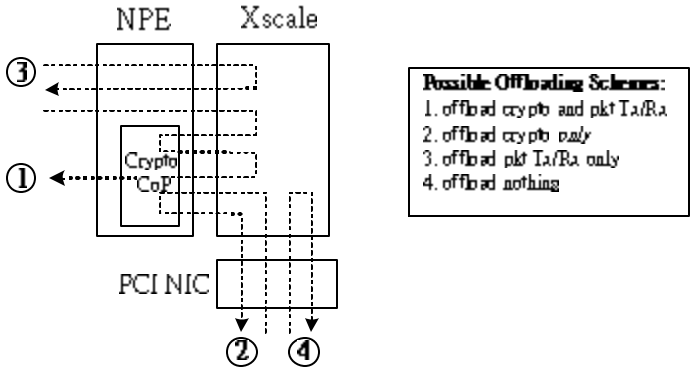
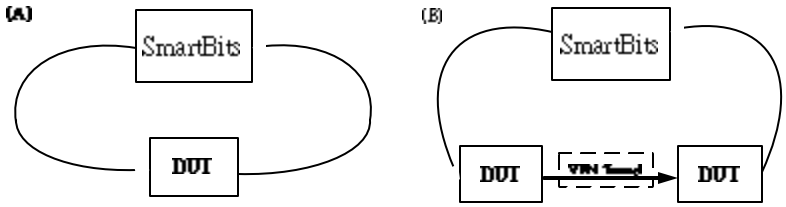


Fig.7 illustrates the external benchmark environments, for packet forwarding and IPsec. We use SmartBits, which is a networking traffic generator and a performance analyzer, to generate the input traffic and collect and analyze the performance results. For internal tests, some system utilities, such as *vmstat*, *top* and *GProf*, are employed to obtain the system state and other internal behaviors such as CPU utilization and memory usage.



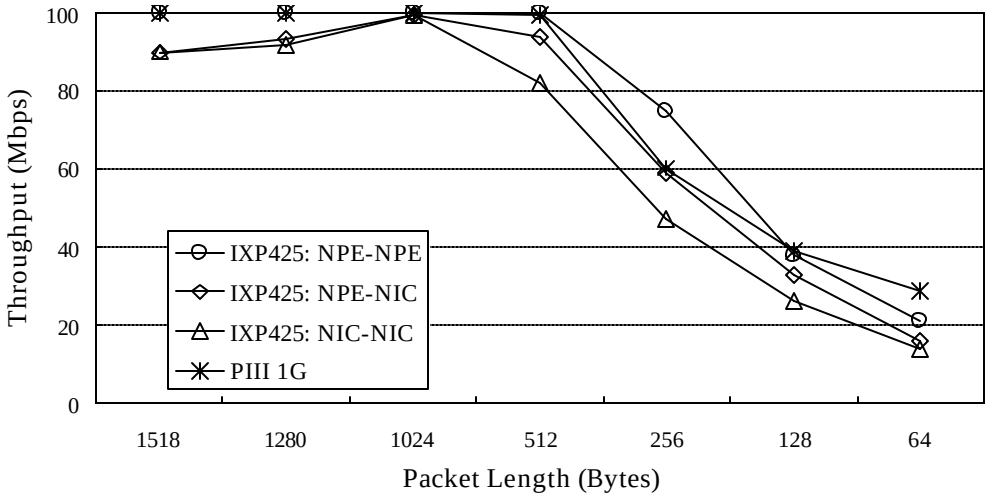


Fig. 8. Throughputs of packet forwarding when different numbers of NPEs are used for offloading.

4.2.2 IPsec Processing

Figure 9 and 10 depict the throughputs of DES and 3DES of different packet lengths. Some observations can be made. First, offloading IPsec processing to the coprocessors in NPE B improves the performance of IXP425 by 350%; in some cases IXP425 even outperforms the Pentium III 1GHz as can be seen in Fig. 9. Second, the maximum of throughputs occurs when the packet length is 1450 bytes, instead of 1510 bytes. This is because 1450 bytes is the largest length for a packet not to be fragmented after encapsulated into IPsec one. Third, the throughput of DES is similar to 3DES, though the computation requirement of the former is almost triple of the later. This is because it is the XScale, not the coprocessors, that becomes the bottleneck, as is to be explained in the later simulations.

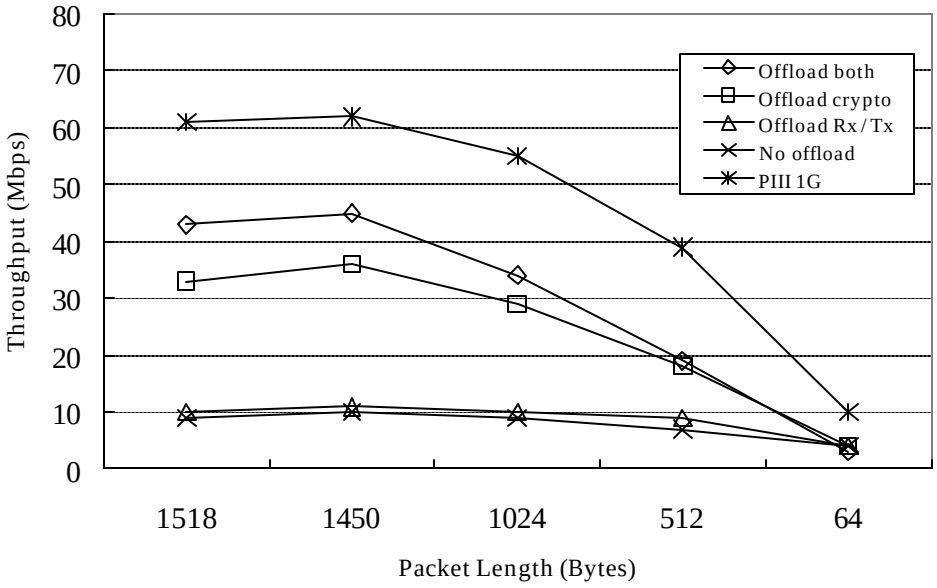


Fig. 9. IPsec Throughput : the DES case.

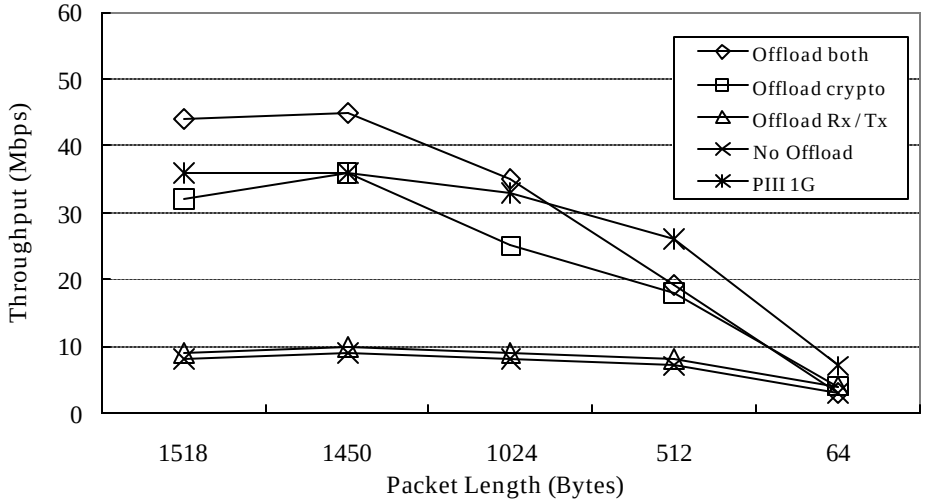


Fig. 10. IPsec throughput : the 3DES case.

4.3 Unbalanced Load between NPE A and NPE B

Unlike NPE A, NPE B doesn't only provides packet reception and transmission but also IPsec processing. We wonder if the different services provided on NPE A and B forms

performance of them different. To measure this problem, we compare the throughput of IPsec processing between settings with different port to receive packet, but only NPE B can form IPsec offloading. As shown in Fig.11, the influence doesn't matter.

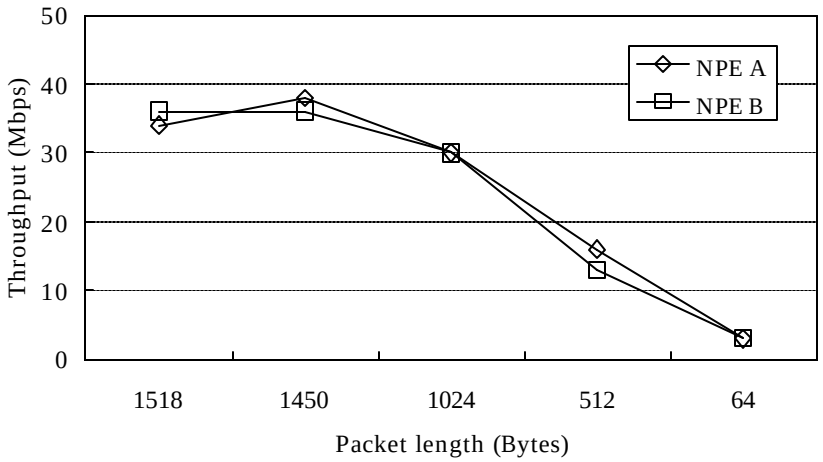


Fig. 11. IPsec throughput with different Rx/Tx interfaces.

4.4 Bottleneck Analysis

Another question to be answered after system benchmark is: what is the bottleneck of the system? Some components in IXP425 could become a bottleneck, namely the XScale, bus and memory. We try to identify it in this section.

4.4.1 Bottleneck of Packet Rx/Tx

The most important factor in performance of Rx/Tx is the turnaround time for packet processing. Figure.12 shows the CPU utilization increases with a direct ratio to the number of packet. Though the processing load for each shorter packet is a little smaller, the number of packet applies that the throughput under small packets are very low. Hence, busyness of XScale leads the latency for packets to increase when packets come too frequently. Then the

packet queue in system is full and remaining packets are all dropped.

Bandwidth of bus or memory would not be the bottlenecks because the throughput on bigger packet length can achieve the wire speed. Furthermore, the NPE isn't either since when we can obtain the packets are all received into system by "netstat" utility. We could say the bottleneck is XScale since most packet processing proceeds on it. Anything with benefit to decrease packet processing time would has improvement to maximum throughput, such as better packet processing method in kernel and higher-speed core processor or memory.

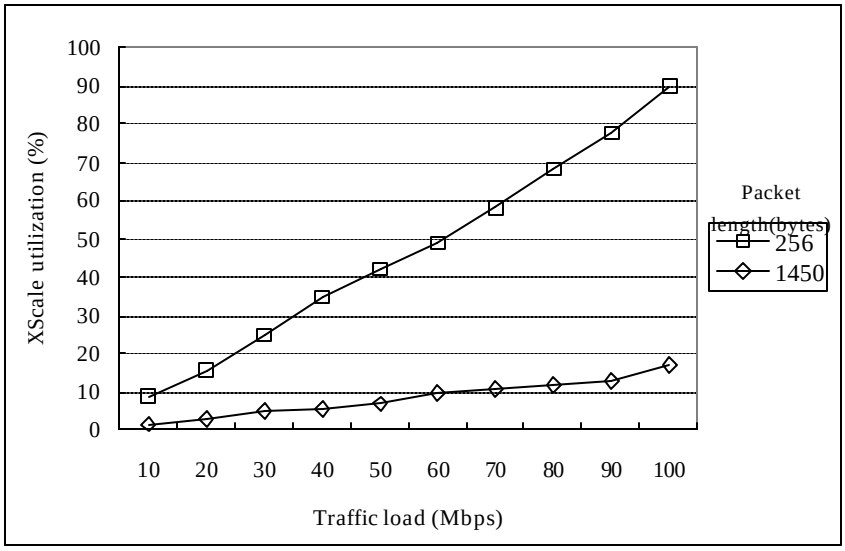


Fig. 12. XScale utilization with varying traffic load and packet length.

4.4.2 Bottleneck of IPsec Processing

The bottleneck in the IPsec processing is known to be the XScale before offloading is applied, since the cryptographic calculation demands much computing power. However, the XScale is again found to be the bottleneck even after offloaded by the crypto coprocessors. This is proven by Fig. 13. From the figure we can see that as the utilization of the XScale approaches 100%, the success ratio of IPsec packets drops significantly; the processor is so busy that incoming packets are dropped due to limited buffer space.

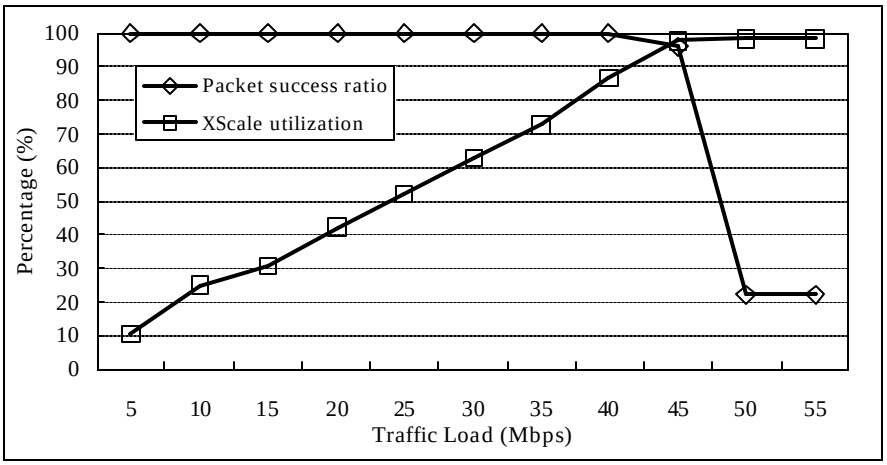


Fig. 13. IPsec packet success ratio versus XScale utilization.

The XScale bottleneck can be further confirmed with the turnaround times of DES and 3DES requests, respectively, as shown in Fig.14. The turnaround time means the duration from the time a request of cryptographic operations is issued by XScale the queue manager, to the time the XScale is notified of the completion. As mentioned previously, the throughputs of DES and 3DES are similar, indicating that their turnaround times should also be same. However, this contradicts the results in Fig.14, in which the turnaround times of DES and 3DES are different.

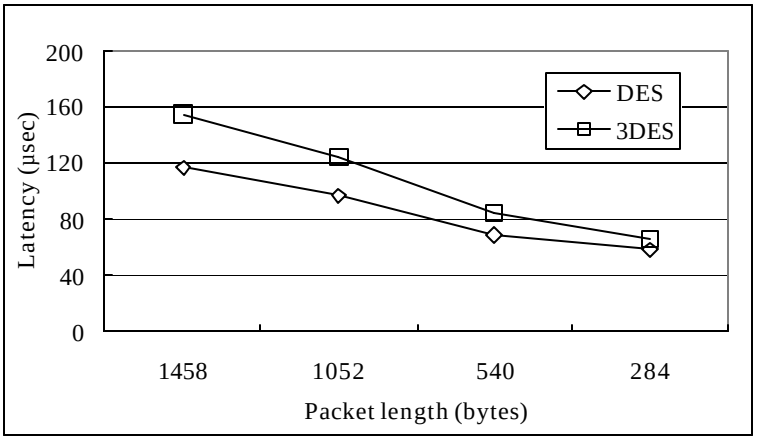


Fig. 14. Turnaround time of a cryptographic request for a packet. Packet size may vary.

We can also estimate the maximum throughput of the crypto coprocessor since the processing times of encryption, decryption and data movement are proportional to the data length. The approximate estimated performances can be computed by △s/△t, where △s and △t represent the differences of two packet lengths and latencies, respectively. That is, the crypto coprocessor is estimated to scale to about $\frac{1458-1052}{117-97} \cong 20.3(bytes/\text{u sec}) = 153.6(Mb/\text{sec})$ for DES and 101Mbps for 3DES.

4.5 Turnaround Time Analysis of Functional Blocks

Figure 15 depicts the turnaround time analysis of the functional blocks in processing DES and 3DES packets. Functional blocks considered in the analysis include the IP preprocessing, IPsec preprocessing, and IPsec encryption. Three different configurations are conducted: IXP425 with the cryptographic operations offloaded to the coprocessor; IXP425 without offloading; and PIII processor. From the figure we can see that the the longest period is cryptographic calculation. Here one important thing is the time of cryptographic calculation in IXP425 could be ignored since NPE handles that but not XScale. Same turnaround time on XScale for DES and 3DES makes our prototypes has the same throughput on DES and 3DES since the bottleneck is XScale but not NPE or co-processor.

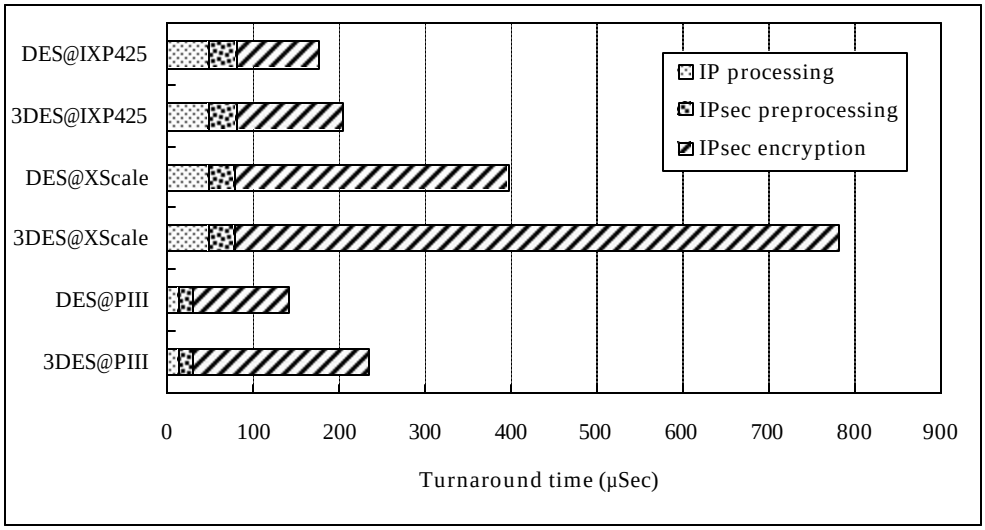


Fig. 15. Turnaround time of functional blocks.

Chapter 5. Conclusions and Future Works

In this work, we elaborate the implementation of a VPN gateway over the IXP425 network processor, where a number of coprocessors are provided for offloading computation intensive tasks from the Xscale core. We introduce the hardware and software architectures of the platform, analyze the VPN, i.e. IPsec, processing flow, and then identify the packet Rx/Tx as well as encryption/decryption as the ones to be offloaded to coprocessors. We realize the offloading design by implementing a number of drivers in NetBSD, and finally externally and internally benchmark the system in order to find possible performance bottlenecks.

The benchmark results show that the throughputs of IPsec processing and packet Rx/Tx have improvements of 350% and 25%, respectively, after offloading. However, the Xscale is again found to be the bottleneck for both packet Rx/Tx and IPsec processing.

Two issues are to be investigated in the future. First, more tasks may be considered to be offloaded to NPEs or coprocessors. An example of this is the IPsec database lookup, which determines the policy to be applied to a certain IPsec packet. Second, the performance may be further improved if we may call the related functions directly in the AccessLibrary for cryptographic operations, instead of going through the Open Crypto Framework.

References

[1] T. Braun, M. Günter, M. Kasumi and I. Khalil, “Virtual Private NetworkArchitecture.” Technical Report IAM-99-001, CATI, April 1999.

[2] About Antivirus Software, Content filtering guide picks, http://antivirus.about.com/cs/contentfiltering.

[3] Michael John and Sebastian Smith, “Application-Specific Integrated Circuits,” Addison-Wesley Publishing Company, ISBN 0-201-50022-1, June 1997.

[4] Panos C. Lekkas, “Network Processors: Architectures, Protocols and Platforms (Telecom Engineering),” McGraw-Hill Professional, ISBN 0071409866, July 2003.

[5] Intel IXP425 Network Processor, http://developer.intel.com/design/network/products/npfamily/ixp425.htm.

[6] Intel XScale Microarchitecture, http://www.intel.com/design/intelXScale/benefits.htm?iid=ipp_a2z+i_620_xsmicro&.

[7] R. Atkinson, Security architecture for the Internet protocol, RFC1825, IETF Network Working Group, August 1995.

[8] T. Spalink, S. Karlin, L. Peterson, and Y. Gottlieb, “Building a Robust Software-Based Router Using Network Processors,” Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP), 2001.

[9] Ying-Dar Lin, Yi-Neng Lin, Shun-Chin Yang, Yu-Sheng Lin, "DiffServ Edge Routers over Network Processors: Implementation and Evaluation," IEEE Network, Special Issue on Network Processors, July 2003.

[10] Intel® IXP12XX Product Line of Network Processors, http://www.intel.com/design/network/products/npfamily/ixp1200.htm.

[11] Intel® IXP4XX Product Line and IXC1100 Control Plane Processors, Developer’s

Manual, Intel® Document Number: 252480, February 2003.

[12] Intel® IXP400 Software, Programmer's Guide, Intel® Document Number: 252539, December 2003.

[13] The NetBSD Project, http://www.netbsd.org/.

[14] Lawrence Kesteloot, "Porting BSD UNIX to a New Platform," http://www.teamten.com/lawrence/291.paper/291.paper.html, January 1995.