VPN Gateways over Network Processors: Implementation and Evaluation

Yi-Neng Lin, Chiuan-Hung Lin, Ying-Dar Lin
Department of Computer and Information Science, National Chiao-Tung University, Hsinchu, Taiwan
{ynlin,chlin,ydlin}@cis.nctu.edu.tw

Yuan-Chen Lai
Department of Information Management, National Taiwan University of Science and Technology, Taipei, Taiwan
laiyc@cs.ntust.edu.tw

Abstract

Networking applications, such as VPN and content filtering, demand extra computing power in order to meet the throughput requirement nowadays. In addition to pure ASIC solutions, network processor architecture is emerging as an alternative to scale up data-plane processing while retaining design flexibility. This article, rather than proposing new algorithms, illustrates the experience in developing IPSec-based VPN gateways over network processors, and investigates the performance issues. The external benchmarks reveal that the system can reach 45Mbps for IPSec using 3DES algorithm, which improves by 350% compared to single XScale core processor and parallels the throughput of a PIII 1GHz processor. Through the internal benchmarks, we analyze the turnaround times of the main functional blocks, and identify the core processor as the performance bottleneck for both packet forwarding and IPSec processing.

I. Introduction

Today's networking applications, such as virtual private network (VPN) [1] and content filtering 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) [2] 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 [3] 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 minor, 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 [4] network processor featuring an XScale 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 [5], 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. [6] presented the results of simple IP

Proceedings of the 11th IEEE Real Time and Embedded Technology and Applications Symposium (RTAS'05)
1080-1812/05 $ 20.00 IEEE

forwarding and Lin et al. [7] implemented DiffServ, both over Intel IXP1200. 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. We first describe the hardware and software architectures of IXP425. Next, we elaborate the details of the design and implementation of VPN over IXP425. Then we present the results and observations from the external and internal benchmarks. Some conclusive remarks of this article are made finally.

II. Hardware and Software Architecture of IXP425

A. Hardware Architecture of IXP425

The hardware block diagram of IXP425 is depicted in Fig. 1. The core of IXP425 is a 533MHz XScale processor handling system initialization and software objects execution. Three buses interconnected by two bridges provide the connectivity among components on IXP425.

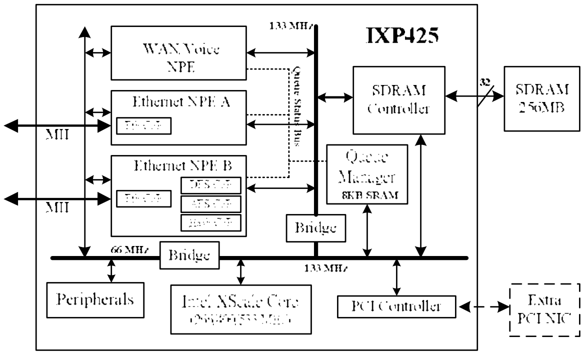

Fig. 1. Hardware architecture of IXP425.

To assist the XScale core in processing networking packets, three 133MHz programmable network processor engines (NPEs) are used to execute in parallel the code image stored in internal memory for providing functions such as MAC, CRC checking/generation, AAL2, AES, DES, SHA-1 and MD5, in cooperate with a number of application-specific coprocessors. The support of hardware multithreading with single cycle context switch overhead further makes NPEs more tolerant to long memory accesses and thus reduces the number of processor stalls. The communication between the XScale core and NPEs is handled by a hardware queue manager using interrupt and message queue mechanisms. The queue manager also contains 8KB SRAM divided into 64 independent queues manipulated as circular buffers for allocating free memory space to incoming packets and for locating packets in the memory. 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.

B. Detailed Packet Flow in IXP425

The processing flow of an ordinary packet is elaborated below referring 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 interrupts the XScale of the reception 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, requested by the XScale, to process a certain data segment in SDRAM, 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.

C. Software Architecture of IXP425

The software architecture shown in Fig. 2 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 favorable especially when an OS migration from a certain H/W platform to another is demanded, that is, the developers need to focus only on the dependent part, namely the development of drivers. When implementing device drivers, 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, such as *OSSL* and *IxOSServices* to implement some OS-related functions such as mutual exclusion.

The software processing flow is described as follows with library functions adopted from the AccessLibrary. During the boot time a function named *IxNpeDl* is called to download the corresponding code image into the instruction cache of each NPE. Then two functions, *IxQmgr* and *IxNpeMh,* are called to initialize the queue manager as well as the message handler responsible for the communications between NPEs and XScale. The Ethernet-related functions, *IxEthAcc* and *IxEthDB*, are used to receive and transmit Ethernet frames, while the *IxCryptoAcc* function is incorporated for possible cryptographic operations during packet processing.

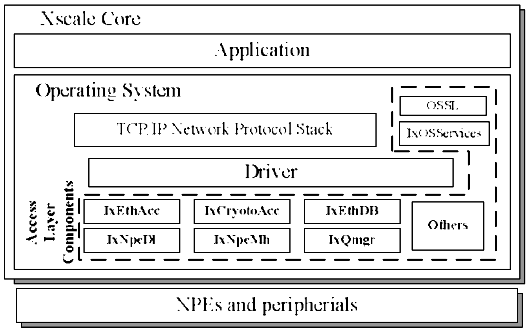

Fig. 2. Software architecture of IXP425.

III. Design and Implementation

In this section, 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.

A. VPN Briefing

Virtual Private Network (VPN) provides secure transmission over un-trusted networks. Normally the IPSec protocol is adopted as the underlying technique due to the popularity of the Internet Protocol. It supports data authentication, integrity and confidentiality, in which two gateways are employed as endpoints constructing a VPN tunnel for secure data transmission. Improving the performance of the gateways is decisive to the VPN throughput.

B. Identifying Offloading Candidates

To resolve the performance issue, we analyze the VPN packet processing flow in order to identify possible candidates to be offloaded to coprocessors. A detailed inbound IPSec packet flow was displayed in Fig. 3. It consists of three main blocks, namely the packet reception, IPSec processing, and packet transmission. Their operations are elaborated below.

Once an Ethernet frame is received by the physical interface, checking for frame check sequence screens out broken frames and the remaining frames are examined in accordance with possible MAC address filtering configurations. Reception is accomplished 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 are performed and payload of this IPSec packet is decrypted or checked for authentication. Finally, a new packet decrypted from the original IP payload is further processed by higher-level protocols, or is transmitted according to the routing table.

Tasks suitable to be offloaded to coprocessors can be identified by two characteristics: whether those tasks are repeated routines or computation intensive ones. As mentioned earlier this section, we know that IPSec processing, especially the cryptographic operation, is computation intensive. Hence, we decide to pick the cryptographic processing as an offloading candidate. Another candidate comprises the packet transfer, CRC checking/generation, MAC filtering, and packet movement between NPE and memory, since the procedures are precisely the same for every packet. From the hardware block diagram in Section II, it is obvious that the IXP425 has the hardware components for the identified candidates.

C. Implementation

We adopt the NetBSD [8], a secure, highly portable and open-source OS derived from 4.4BSD, as our operating system. Clean design between platform dependent and independent parts and the driver support for various networking interfaces make it a good implementation target for new hardware platform. Following relates three major components in prototyping a security gateway over IXP425.

Operating System Porting. The most efficient way to porting an OS to a new platform is refer to the port of another similar platform and then implement drivers for the target platform based on that port [9]. To port NetBSD over IXP425, therefore, we adopt the “EvbARM” port in NetBSD. It supports various evaluation boards that equip with XScale or other ARM-based core processors, so that only system-level modifications have to be done to enable normal operations of IXP425. Example modifications include the CPU identification, setup of board-specific memory map, and system initialization procedures.

Driver Development. A number of drivers for devices

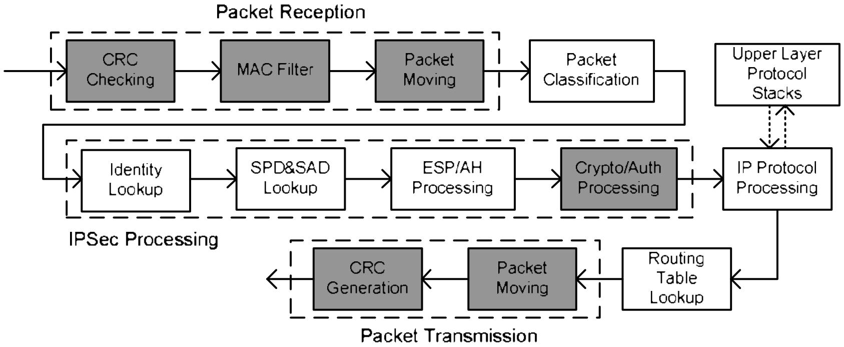

Fig. 3. Processing flow of an inbound IPSec packet. Shaded blocks are candidates to be offloaded.

such as UART, NPEs and coprocessors need to be implemented for communication between the operating system and those devices. This effort can be alleviated with the help of the AccessLibrary introduced in Section II. Besides drivers, we have to modify two OS dependent modules, namely OSSL and IxOSServices, in AccessLibrary to ensure proper operations of the OS-related services.

Offloading the Cryptographic Operations. The last modification to kernel concerns the offloading of in-kernel IPSec cryptographic computations from XScale to coprocessor. Ordinary method requires that the kernel performs and subsequently *waits* on the encryption/decryption operations carried out by the coprocessor. However, NetBSD provides another option named *FAST_IPSec* that makes use of the Open Crypto Framework (OCF) for offloading. In OCF, the cryptographic operations can be handled by a *registered* function. The FAST_IPSec prevails over the original offloading technique in that the XScale would not suspend during cryptographic operations. We exploit this technique by pre-registering the crypto driver, which drives the crypto coprocessor using functions in AccessLibrary, to the OCF.

IV. System Benchmark and Bottleneck Analysis

In this section, we investigate the benefits from offloading by externally benchmarking the implementation using various offloading schemes. A number of internal tests are also conducted in order to observe what cannot be obtained in the external benchmarks.

A. System Benchmark Setup

To have a better understanding of the improvement from the network processor architecture as well as the offloading mechanisms, we design and benchmark systems of different offloading schemes, and compare their performance results. Four offloading schemes are adopted: (1) offload both crypto operations and packet Rx/Tx to the corresponding coprocessors; (2) offload crypto operations only; (3) offload Tx/Rx only, and (4) no offloading. Figure 4 diagrams the corresponding data paths for the four schemes.

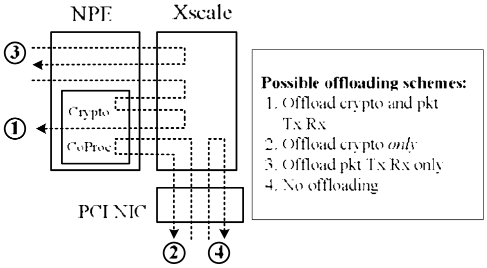
NPE
Xscale
③
①
Crypto
CoProc
PCI NIC
②
④
Possible offloading schemes:
1. Offload crypto and pkt Tx/Rx
2. Offload crypto *only*
3. Offload pkt Tx/Rx only
4. No offloading

Fig. 4. Data paths of the four offloading schemes.

As for the external benchmark environments for packet forwarding and IPSec, we use *SmartBits* to generate the input traffic and to 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 as well as other internal behaviors such as CPU and memory utilizations.

B. Scalability Test

Scalability tests aim to derive the maximum throughput of the prototypes of different offloading schemes. Another gateway implementation using Pentium III 1GHz processor and 256MB SDRAM is also included for comparison between IXP425 and x86-based systems.

Packet Forwarding. Figure 5 shows the performance results of 1-to-1 packet forwarding under the condition of zero packet loss. From the figure we can see that throughput of the IXP425 offloaded by two NPEs parallels the one of Pentium III 1GHz. Both of them can support wired speed for packet lengths larger than 512 bytes. Besides, a performance improvement of up to 60% contributed by NPEs can also be gained. We also observed that the maximum throughput occurs when the packet length is 1024 bytes, rather than other larger lengths. This is because the longer processing time of larger packets counteracts the benefit from their reduced header processing overhead.

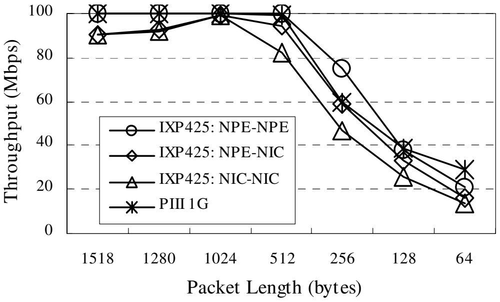

Fig. 5. Throughput of packet forwarding when different numbers of NPEs are used.

IPSec Processing. Figure 6 depicts the throughput of DES for different packet lengths. Some observations can be made. First, offloading IPSec processing to coprocessors in NPE B improves the performance by 350%; in some cases IXP425 even outperforms the Pentium III 1GHz. Second, the maximum throughput occurs when the packet length is 1450 bytes, instead of 1518 bytes. This is because 1450 bytes is the largest length for a packet not to be fragmented when being encapsulated into an IPSec one. Third, the throughput of 3DES on IXP425, as shown in Fig. 7, is similar to the one of DES whereas the computation requirement of the former is almost triple of the later. The reason is that it is the XScale, not the coprocessors, that becomes the bottleneck.

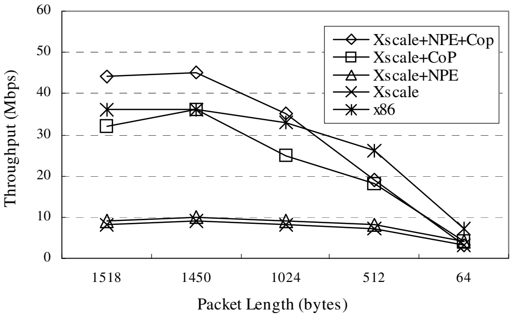

Fig. 6. IPSec Throughput: the DES case.

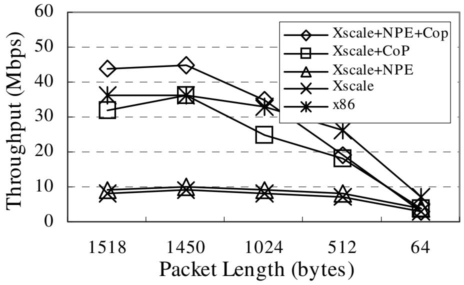

Fig. 7. IPSec Throughput: the 3DES case.

C. Bottleneck Analysis

Bottleneck of Packet Rx/Tx. To proceed the bottleneck analysis, we considered four main functional units likely to affect system performance: bus, memory system, NPE and XScale. It is obvious that neither the bus nor the memory is a bottleneck because wired speed can be achieved for some larger packet lengths. The NPE is not a bottleneck either, since, as observed by the *netstat* utility, all packets are received and stored at the memory. The bottleneck can therefore be identified as the XScale since the packet processing is carried out mostly by it. Figure 8 shows that the utilization of the XScale linearly advances as the traffic load increases.

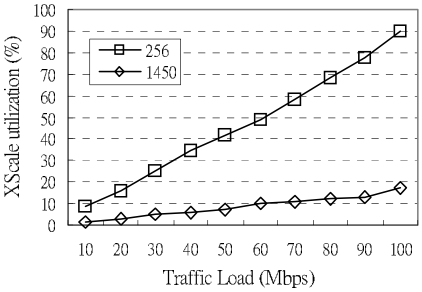

Fig. 8. Input traffic load vs. XScale utilization for two packet lengths (bytes).

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. Figure 9 shows that when traffic load is 50Mbps exceeding the maximum system throughput of 46Mbps, the utilization of XScale approaches 100% and the success ratio of IPSec packets significantly drops to 22%. This is because the processor is so busy that incoming packets are dropped due to limited buffer space.

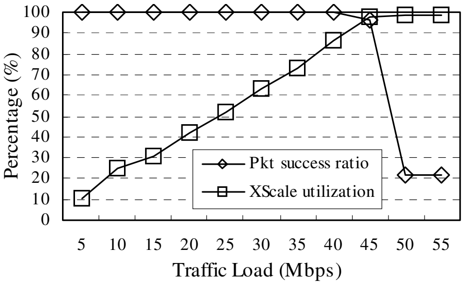

Fig 9. IPSec packet success ratio vs. XScale utilization.

The XScale bottleneck can be further confirmed with the turnaround times of the DES and 3DES requests, respectively, as shown in Fig. 10. The turnaround time means the duration from the time a request of cryptographic operations is issued by XScale to 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 the same. However, this contradicts the results in Fig. 10 in which the turnaround times of DES and 3DES are different, justifying that the XScale, rather than the crypto coprocessor, is the bottleneck when performing DES and 3DES. The throughputs of DES and 3DES are the same because they are bound by XScale.

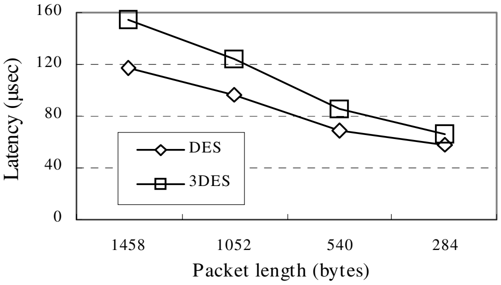

Fig. 10. Turnaround time of a cryptographic request for a packet. Packet size varies.

We can also estimate the maximum throughput of the crypto coprocessor as the processing times of encryption and decryption are proportional to the data length. The estimated performances can be computed by △s/△t, where △s and △t represent the differences of two packet lengths and two latencies, respectively. Therefore, the crypto coprocessor is estimated to scale approximately to $\frac{1458-1052}{117-97} \cong 20.3(bytes/\text{u}\sec) = 162.4(Mb/\sec)$ for DES, and to 101Mbps for 3DES likewise.

D. Turnaround Time Analysis of Functional Blocks

Figure 11 depicts the turnaround time analysis of the functional blocks when processing DES and 3DES packets. Functional blocks considered consist of the IP processing, IPSec preprocessing including identity and SAD/SPD lookups, and IPSec encryption. Three kinds of testbed configurations are conducted for testing DES and 3DES: IXP425 with the cryptographic operations offloaded to the coprocessor; IXP425 without offloading, namely XScale only; and PIII processor.

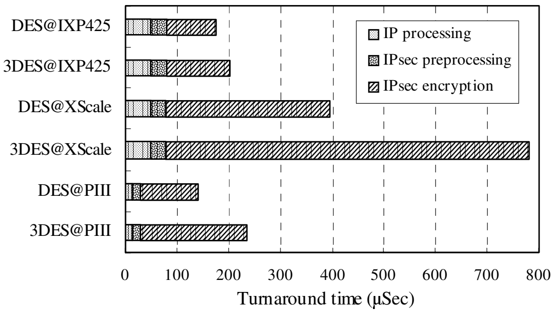

Fig. 11. Turnaround time of functional blocks.

From the figure we can see that cryptographic calculation accounts for a major portion, from 80% to 90%, in the packet processing time before offloading. After offloading to the coprocessor, the time for cryptographic calculation is reduced from 700 us to 100 us. Notably both the IXP425 and single XScale configurations have the same IP processing and IP preprocessing periods because those tasks are executed only by XScale.

V. 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 packet Rx/Tx and IPSec processing are improved by *60%* and *350%*, 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 offloaded to NPEs or to 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 call the related functions in the AccessLibrary directly for cryptographic operations, instead of going through the Open Crypto Framework.

References

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

[2] M. John and S. Smith, "Application-Specific Integrated Circuits," Addison-Wesley Publishing Company, ISBN 0-201-50022-1, June 1997.

[3] P. C. Lekkas, "Network Processors: Architectures, Protocols and Platforms (Telecom Engineering)," McGraw-Hill Professional, ISBN 0071409866, July 2003.

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

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

[6] 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.

[7] 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.

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

[9] Lawrence Kesteloot, "Porting BSD UNIX to a New Platform," January 1995.