A 10 Gbit/s IPSEC Gateway Implementation

CHEE-WEI TAN, MIRKO BENZ and ALEXANDER SCHILL
Institute for System Architecture, Chair for Computer Networks
Dresden University of Technology
01062 Dresden
GERMANY

Abstract: - Internet Security (IPSEC) protocol is part of a design consideration in Virtual Private Networks (VPN). In this paper, we design and implement a 10 Gbit/s gateway router for IPSEC processing using the Intel network processor IXP2850. In particular, using software and hardware partitioning on a complex multi processor system, i.e., selecting appropriate processors to offload computational intensive tasks, we are able to accelerate the IPSEC data path. We also highlight the performance issues with IPSEC protocol implementation using the cryptography engines in IXP2850, and propose efficient data structure for key management in the buffer when large number of security associations are re-keyed at line speed.

Key-Words: - IPSEC hardware support, virtual private networks, network processor, high-speed networks

1 Introduction

By implementing security at the Internet Protocol (IP) level, both applications that are aware and ignorant of security mechanism in a network can ensure secure networking. IP-level security consists of the following three functional areas:

- Authentication and Integrity: Assures that the packet is sent by the source identified in the packet header, and the packet has not been altered by a third party.
- Confidentiality: Enables communicating nodes to share a secret by encrypting and decrypting messages without any third party eavesdropping.
- Key management: Secure exchange of keys for encryption and decryption.

IPSEC is mainly deployed in Virtual Private Networks (VPN), for example, most large enterprises maintain wide-area networks where VPNs are used to ensure secure data transfer over potentially insecure connections such as the Internet, and corporations use tunneling technology for remote users to access the corporate network at maximum security by providing end-to-end security between two hosts. In this paper, we design and implement a 10 Gbit/s gateway for a VPN.

IPSEC uses two functions: Authentication Header (AH) to provide authentication, and the Encapsulating Security Payload (ESP) to encrypt the data portion of the IP packet ([15] pp. 408, 413). In short, AH provides authentication and integrity, prevents address spoofing attacks, guards against replay attack, and both parties must process a secret key. ESP provides confidentiality, with or without authentication. The use of AH and ESP is defined by a Security Association (SA). A SA is a one way relationship between the sender and receiver.

Typical approach for hardware support of secure communication protocols involves offloading the encryption and decryption algorithms on hardware, while the main protocol and I/O handling are done using software, for e.g., the Linux-based FreeS/WAN [3] and BSD-based KAME [11] projects which run on software routers. Software routers are relatively slow in comparison to complete hardware implementation, but it provides full flexibility. On the other hand, recognizing that only a small part of the IPSEC protocol implementation, for eg. IP/IPSEC send and receive data path, and dynamic buffer and key management, has high-speed processing requirement, IPSEC protocol processing can be accelerated by implementing this crucial portion on network processors. The rest of the IPSEC protocol can still be implemented using slow software solutions. In this way, we can distribute tasks with different priorities to different processors. Essentially, a resource allocation approach based on a *processor hierarchy* similar to [14] is adopted in this paper.

There are several key challenges to designing a complete IPSEC protocol on network processors. First, the amount of instructions that can be exe-

cuted in the data path is usually small, ranging between some ten to hundreds of instructions. Hence, to optimize the IP/IPSEC send and receive paths, synchronization points between the software and hardware implementation need to be detected. Second, low level processes such as extraction of misaligned data and buffer access need special tuning to avoid introducing performance bottlenecks in the critical path. Third, protocol related issues may cause bottlenecks even if every IPSEC packet is processed correctly. For example, the Internet Key Exchange (IKE) Protocol ([15] pp. 421–422) is used for negotiation of IPSEC SAs. It is proposed that IPSEC SA should be *re-keyed*, i.e., an automated mechanism by which a new SA should be established before the SA expires, proactively. The elapsed time between the establishment of the new SA and the expiration of the old SA should be adequate to avoid losing any data being transmitted in the old SA. For a large system of interconnected hosts and the gateways, keeping track of authentication keys quickly becomes a formidable task. The task of re-keying and accounting for all possible hosts increases dramatically as we increase the number of tunnels or secure connections in the VPN. As a result, since fast memory (e.g. SRAM) is a limited resource in most network processor applications, system performance may degrade due to large memory consumption. Furthermore, a natural question is, is it possible to guarantee processing requirement for a workload that consists of both IP and IPSEC packets?

In this paper, we investigate the underlying cryptography mechanism, and interaction between software and hardware that implement these protocols on an Intel network processor, IXP2850 [8]. The paper is organized as follows: In Section 2, we present our system architecture, and discuss our design of a 10 Gbit/s gateway. In Section 3, we discuss our implementation in detail. In Section 4, we propose a bloom filter approach for key management in the gateway, and we analyze the performance of our implementation in Section 5. We then discuss related work in section 6. Lastly, we conclude the paper in Section 7.

2 System Architecture

In this section, we present our system architecture targeting network processors for IPSEC processing. Our architecture supports IPSEC functionalities such as DES, 3DES, AES, SHA-1 and HMAC,

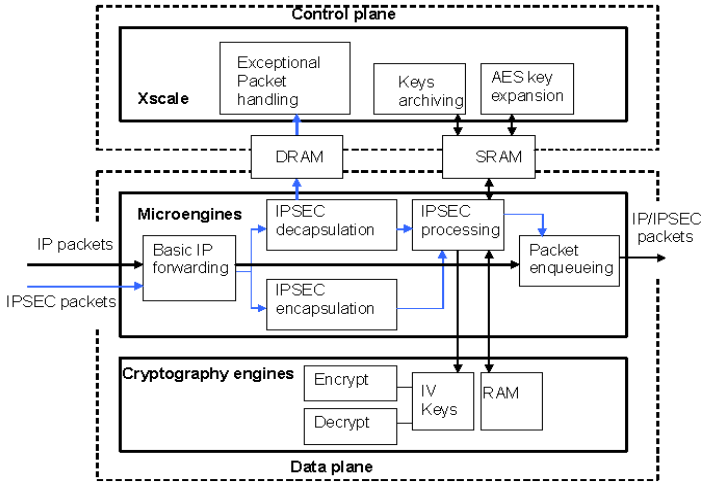

Figure 1: IP/IPSEC system overview.

but we shall concentrate exclusively on AES and HMAC-SHA1 in this paper.

2.1 Overall View

Our IPSEC protocol processing stack is partitioned into a control and data plane as shown in Fig. 1. Control plane software is implemented on an embedded Linux operating system in the XScale processor. Data plane functionalities such as IP forwarding path and cryptography computation are implemented using microengine assembly code. The control plane functionality involves mainly key management, security control of the data plane, and handling of exceptional packets. Computational intensive tasks such as key expansion function in the AES algorithm are offloaded onto the control plane.

2.2 Linux Integration

We adapt the Linux 2.6 kernel to run on the XScale processor. Part of the existing Linux IPSEC protocol are re-used. The SA database and the XFRM policy database are also implemented in the Linux kernel. Further details on these control plane software can be found in [12].

2.3 INTEL IXP Architecture Overview

The Intel IXP2850 is a fully programmable network processing unit (NPU) that implements a high performance parallel processing architecture. It combines a high performance Intel XScale core processor, multiple memory channels, a PCI interface, a multi-purpose network interface, and sixteen 32-bit

independent, multi-threaded microengines on a single chip. In addition, the IXP2850 contains two crypto units [8]. Each crypto unit contains one AES block and two independent 3DES blocks, which are used by the AES and 3DES symmetric key ciphers for bulk encryption and decryption. Each 3DES block has access to three initialization vectors (IV) and three keys. The AES block has access to six IVs and six keys, which are the same physical resources that supply the IVs and keys for the 3DES block. The AES block supports 128-bit IVs and keys of length 128, 192 and 256 bits, while the 3DES block supports 64-bit IVs and 192-bit keys. Each crypto unit also contains two independent SHA-1 blocks for computing 160-bit message digests. A checksum accumulator is used to compute checksums over any data passing through the crypto unit.

The IXP2850 uses hyper task chaining to improve the efficiency of pipeline processing by allowing direct data transfer between microengines using next-neighbor transfer registers instead of convectional memory such as the SRAM and SDRAM.

3 Prototype Implementation

In this section, we describe our software implementation on the Intel IXP2850 network processor which consists of software partitioning for the microengines and XScale processor. We adapt the 10 Gbit/s IP forwarding reference application (10gb_ethernet_ingress) for our IPSEC implementation where four microengine threads are statically allocated to each port in a ten 1 Gbit/s ports setting.

3.1 Software partitioning on the Microengines

IPSEC functionality such as AES or HMAC can be sub-divided into smaller functional blocks, which after can be assigned to different microengine contexts for parallel execution. In our implementation, five microengines are used for packet forwarding that include Ethernet decapsulation and basic IP header processing. Due to the highly variable payload size of IPSEC packets and demanding security computation, one microengine, Security microengine (*SecME*), is dedicated to process the payload of IPSEC packets. Hence, *SecME* acts as an interface between IP forwarding and access to the security cores.

In *SecME*, four threads are statically assigned to IPSEC input processing and the other four to IPSEC output processing. Communication between the forwarding engines and *SecME* is done through inter-microengine signaling and the memory interfaces. When an IPSEC packet is received, the forwarding engines first inspect if the policy for the packet is allowed. If the policy is allowed, and if the packet has reached the destination of the tunnel, the security function is processed and removed by *SecME*. For an outgoing packet, the forwarding engine determines the corresponding security function (algorithm and mode) from the policy, that is subsequently applied by *SecME*. Finally, the forwarding engines encapsulate an outer header with the new source and destination address to be the address of the tunnel entry and the tunnel exit nodes respectively. Forwarding is based only on the destination address of the outer packet. Our implementation of the AES algorithm supports only a fixed non-linear 16×16 substitution table (S-box) which can be found in [6], and runs using 128-bit, 192-bit and 256-bit keys. To accelerate extraction and storage of different sized keys at the data plane level, we use an efficient Bloom filter data structure described in section 4.

A software partition of the AES algorithm, in particular the key expansion algorithm that expands the decryption key runs on the XScale processor as a Linux daemon module. The expanded decryption key is subsequently used by the AES core that runs on the data path. For HMAC functionality, a double inner and outer SHA-1 hashing plus state preparation is used rather than the HMAC assembly instruction provided by INTEL as we could not get it to work. HMAC is run to authenticate the IPSEC packet before decryption in *SecME*. Also, the policy management and some other computation intensive portions of the IPSEC algorithm initialization run on the XScale core.

4 Use of a Bloom Filter for efficient key storage

In a VPN network, it can be expected that there will be huge numbers of secure connections that will traverse and be processed at the edge router which is located at both endpoints of VPN tunnels. In order to support the large numbers of different secured connections, both fast processing and large memory requirements are needed at the edge routers. Given the limited number of microengine threads and onboard memory, we use a well-known data structure known as a *Bloom Filter*. A Bloom filter is a method for probabilistically representing a set $A =$

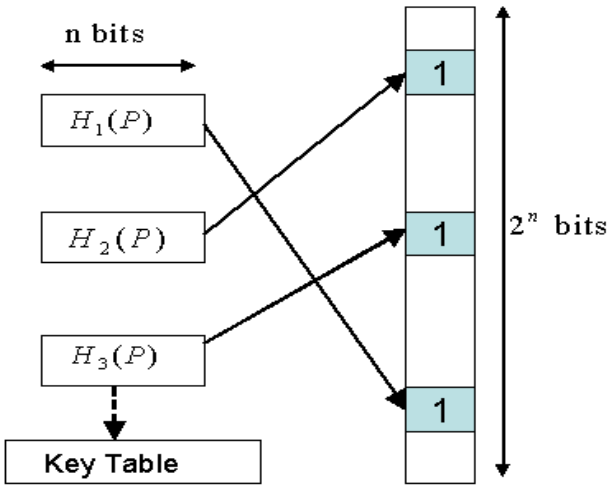

Figure 2: Three hash functions computed to index into a 2^n bit vector. If the query succeeds, $H_3(P)$ is masked for indexing into a hash list.

$\{a_1, a_2, ..., a_n\}$ of n elements (also called keys) to support membership queries. This compact representation is the payoff for allowing a small rate of false positives in membership queries, i.e., queries might incorrectly recognize an element as member of the set. It was invented by Burton Bloom in 1970 [2] and was proposed for use in free text searching [13]. Basically, the set of words that appear in a text is succinctly represented using a Bloom filter.

Rather than extracting the IV, encryption and decryption keys from the packets needed for cryptography computation, we use a Bloom Filter to store these information in a compact fashion on the local memories of IXP2850. In summary, a Bloom Filter executes a series of hash functions to determine a query as shown in Fig. 2. A bit vector is stored in the scratchpad memory to verify if the packet received by the IXP2850 belongs to a new secure connection. A bloom filter is associated with each input port. The input of the bloom filter, P, is a concatenated string of the encryption key and the IV. For every first packet of a new secure session, a new session will, of course, fail the query, and will thus be inserted into the Bloom filter. The relevant states are then extracted from the encrypted IP packet. Using the states, the decryption key is computed using the key expansion algorithm on the XScale processor. This information is stored in a hash list, and accessed at a later stage by the cryptography functions in the microengines and the XScale processor. Subsequent packets that belong to this particular session will make a query in the Bloom filter, and retrieve them from the hash list. For our purpose, we use a Bloom filter configuration of five secured connections per bit in the Bloom filter, and three independent hash operations (one on-board hash hardware and two universal hash software), thus corresponding to a false positive rate of 0.092. A false positive collision is treated as a "cache miss", and therefore the decryption key must be re-computed for that particular collision. However, it is rather rare for a false positive collision to persist for a substantial long time because the connections are expected to be continuously re-keyed at high rates for enhanced security. We mask 12-bit of the last hash function as an index to a hash list whose entries contain the address of the SDRAM memory that stores the IV vector, encryption and decryption keys that are stored upon the arrival of the first packet in the session.

In the next section, we investigate the overhead saved in using a Bloom Filter as compared to state extraction, and report its feasibility for high input streams with many connections per port. In addition, the XScale will periodically map the data stored in the SDRAM onto slower memory that may be accessed at a later time for management and authentication purpose. For example, the Internet Key Exchange can use this information for transmitting pairs of keys for AH and ESP. Lastly, a nice feature of the Bloom filter is that two independent sets can be combined together by performing a bit-wise OR operation over the two corresponding filters. Hence, to achieve greater savings in space, this property can be exploited for a multi-port setting. For the ten 1 Gbit/s ports setting in our implementation, we merge the bloom filters for every two ports.

5 Performance Analysis

In this section, we analyze the performance of IPSEC protocol processing in terms of computational requirements and scalability using test vectors available from public domain [5], [6] and [7]. The purpose is to empirically determine the combined capacity of the forwarding threads and a security thread using a typical IPSEC workload. We conduct all our experiments using the cycle accurate Intel IXP2850 Developer Workbench 3.5. Microengines and the XScale core run at 1400 MHz and 700 MHz respectively. SRAM and SDRAM memories are configured at 256MB and 192MB respectively.

Table 1: Processing requirement for (a) a single microengine thread in *SecME* of IPSEC header extraction, insertion, states (16 Byte IV and 128 Bit key) extraction and checksum computation, and (b) the execution time of various cryptography functions in microengine cycles of the crypto engines for a 108 Byte secure payload.

State processing	Cycles
Header extraction	538
Header insertion	494
Key and IV extraction	345
Checksum computation	68

Cryptography function	Cycles
AES-CBC	1008
3DES	829
SHA-1 hash	450
HMAC using SHA-1 hash	1162

Table 2: Processing requirement for (a) decryption of [7] Case 5 packets and (b) authentication of [5] packets with 100Byte HMAC keys

ESP Decryption	Cycles
IPSEC header extraction	533
Key extraction and load	181
IV extraction and load	169
Bloom filter lookup	156
AES-CBC decryption	1278
Checksum computation	72
IP Packet re-alignment	378

HMAC authentication	Cycles
IPSEC header extraction	533
Key hashing (to 20 Bytes)	752
XOR with Ipad	43
XOR with Opad	43
HMAC authentication	1548
Checksum computation	72
IP Packet re-alignment	349

5.1 Computational Requirements

We measure the performance of our implementation for a transport mode ESP decryption using test vectors found in Request for Comments (RFC) 3602 ([7] Case 5 test vector), and, for AH authentication using HMAC SHA-1, we use the sample data found in ([5] Sample 3). Since our design is based on assembling functional blocks, the number of cycles allocated to each functional block depends on the following factors:

- Minimum packet size and the worst-case packet arrival rate for a given line rate, e.g., OC-192 min packets arrive 40 ns apart
- The frequency of the microengine

thus the number of cycles corresponding to the size of the IPSEC payload is used as a performance metric. Table 2(a) shows the breakdown of processing cycles at each stage in decrypting a packet that is protected by a 16 Byte key and IV with a total encrypted payload of 80 Bytes. The overhead of key expansion is not shown in the table as it is performed in the XScale processor. However, the processing budget for the key expansion algorithm in the XScale translates into the order of thousands of cycles, including memory references. As shown in the table, performing a query using a bloom filter and hashing into a list has significantly lower overhead than the combined state preparation (key and IV extraction plus key expansion).

Table 2(b) shows the processing cycles required for a packet that is protected by a 100 Byte key and a 20 Byte digest. The number of cycles required for authenticating a 100-Byte packet is comparable to that of decrypting an 80-Byte payload in AES. For the above experiments, a single thread in *SecME* is sufficient to compute the two functionalities. In summary, cycle budget allocated for these two separate functionalities in AH and ESP is a function of the payload size. However, it is seldom that keys greater than 100 bytes are used for authentication since they would be hashed to 20-Byte keys in SHA-1. Hence, the main design consideration is that, for larger payloads, multiple threads in *SecME* need to be utilized in parallel to decrypt the same packet.

5.2 Scalability

We configure different input streams to test the scalability of our implementation. First, we simulate one secure connection (80-Byte payload AES encrypted with a 100 Byte MAC digest) that is mixed with normal 64-Byte IP packets for each stream per port. A single thread in *SecME* is configured to handle the secure connection. We configure the above stream on two separate 1 Gbit/s input ports and measure the achievable output rate. Table 3(a) shows the

achievable average output rate as compared to the number of secured packets for every 16 packets (a mixture of secured IPSEC and normal IP packets) received. The IPSEC packet is distributed randomly among every 16 packets received. As shown, the combined processing of AES decryption and MAC authentication on a single thread can sustain a mixed traffic with approximately half IP and IPSEC packets for every 16 received packets. However, as the ratio of secured packets increase beyond half, the processing power of a single thread is not sufficient and in turn affects the throughput.

In the second experiment, we configure two different secure connections with the same payload and digest size but with different keys and IV per connection. The purpose is to simulate a re-key session. The interim between the re-keying of the session, i.e., the time between the departure of the last packet of the pre-re-keyed session and the arrival of the first packet of the re-keyed session, is simulated by a variable number of 64-Byte normal IP packets. Table 3(b) shows the achievable average output rate for the period between the time when the first packet of the re-keyed session is received and when the processing of the third packet of the re-keyed session has finished. The achievable rates in Table 3(b) are lower as the interim decreases because the forwarding threads are blocked at the mutex lock before the new state processing is completed when a session is re-keyed. The above experiments only illustrate the AES-CBC and HMAC authentication execution capacity of a single microengine thread. However, we have utilized only eleven out of the total sixteen microengines, hence it is possible to scale up to 10 Gbit/s IPSEC decryption and authentication from ten 1Gbit/s ports by statically extending *SecME*'s functionalities to the remaining microengines.

6 Related Work

While significant amount of work on IPSEC protocol implementation has focused on implementing a fully working protocol stack in the context of software routers such as the Linux-based FreeS/WAN [3] and the BSD-based KAME project [11], this work is most closely related to [14], [12] and [1] where the idea is to design a framework that exploits the underlying hardware capabilities of network processors. First, designing a framework that supports data plane partitioning based on a characterization of the protocol workload is not new. Early work in [4] has shown that a simultaneous multithreaded processor model is best suited for accelerating protocol based on workload that represents IP forwarding, encryption and authentication. Second, it factors in the interoperability between hardware and software. Other related work on accelerating data paths for cryptography processing include [10] where high performance AES 128-bit encryption engine is implemented on field programmable bit array.

Another research topic related to this paper is that of efficiently encrypting and decrypting large numbers of secure connections. Accelerating the data path of IPSEC processing means more than processing every IPSEC packet correctly. Our work has implications on the kind of workload that VPNs might generate which is related to resource allocation at the gateway. Hence, there is a need to analyze how to adapt IPSEC protocol stack without creating performance bottlenecks. We have highlighted a situation where key management is constrained by the number of sessions. Though it can be argued that this problem can be solved using faster memory, for example, Content Addressable Memory (CAM) has been conventionally used in the industry for caching packet headers or to store an optimized database in IP co-processors such as ASICs and FPGAs [9], we note that CAM cannot scale to a large size due to technological constraints, and is costly to implement. On the other hand, we present a novel approach to solving this problem of storage capacity using a Bloom filter.

7 Conclusion

We have designed and implemented the data path for an IPSEC gateway using the Intel IXP 2850 network processor that is capable of processing 10 Gbit/s of data. We use a Bloom filter to reduce the overhead of key management that might arise from the key exchange protocol when packets are processed at high speed. We present a performance analysis of our implementation and show that a single thread per IXP2850 microengine can process IPSEC packet AES-CBC decryption and HMAC-SHA-1 authentication at a rate of 2 Gbit/s if the workload consists of about half IPSEC and IP packets per 16 packets received.

Our future work focuses on optimizing the IPSEC implementation to fully exploit the network processor's potential. Furthermore, we plan to integrate it with the 2.6 Linux IPSEC framework.

Table 3: Achievable output rate in Mbit/s as (a) the number of secured packets per 16 packets differ for two single secure connections, and (b) a secured session being re-keyed

No. of secured packet	output rate
1	1999.8
2	1993.3
4	1981.2
8	1973.3
10	1928.2
12	1907.0
16	1823.9

No. of packets in interim	output rate
20	1999.8
16	1985.2
14	1951.2
8	1920.7
4	1914.2
2	1871.0
0	1783.9

References:

[1] M. Benz and R. Lehmann, "TCP acceleration on network processors", IASTED International Conference on Communications, Internet and Information Technology (CIIT), St. Thomas, Virgin Islands, November 2002

[2] B. Bloom, "Space/time tradeoffs in hash coding with allowable errors", CACM, 13(7), pp. 422–426, July 1970

[3] Colubris Networks, "FreeS/WAN hardware acceleration patch", http://sources.colubris.com/en/projects/FreeSWAN/

[4] P. Crowley and J-L. Baer, "A modeling framework for network processor systems", Network Processor Workshop in conjunction with Eighth International Symposium on High Performance Computer Architecture (HPCA-8), Cambridge, MA, February 2002

[5] Information Technology Lab, National Institute of Standards and Technology, "The Keyed-Hash Message Authentication Code (HMAC)", Federal Information Processing Standards Publication, March 2002

[6] Information Technology Lab, National Institute of Standards and Technology, "Announcing the Advanced Encryption Standard (AES)", Federal Information Processing Standards Publication, November 2001

[7] S. Frankel, R. Glenn, and S. Kelly, "The AES-CBC cipher algorithm and its use with IPsec", RFC 3602, September 2003

[8] Intel Corporation, "Intel IXP2850 network processor hardware reference manual", First release, Order number: 278738-001, February, 2003

[9] IDT Corporation, "Taking packet-processing to the next level—Achieving next-generation classification performance using multiple databases and IP co-processors", White paper, August 2002

[10] K. Jaervinen, M. Tommiska, and J. Skyttae, "A fully pipelined memoryless 17.8 Gbps AES-128 encryptor", ACM/SIGDA International Symposium on Field Programmable Gate Arrays, pp. 207–215, Monterey, February 2003

[11] Kame Project, available at http://www.kame.net

[12] R. Lehmann, M. Benz, S. Gross and M. Hampel, "IPSEC protocol acceleration using network processor", IASTED International Conference on Communications, Internet and Information Technology (CIIT), Scottsdale, November 2003

[13] M.V. Ramakrishna, "Practical performance of Bloom filters and parallel free-text searching", Communications of the ACM, 32 (10). 1237-1239, October 1989

[14] T. Spalink, S. Karlin, L. Peterson and Y. Gottlieb, "Building a robust software-based router using network processors", 18th ACM Symposium on Operating Systems Principles (SOSP'01), pp. 216–229, Chateau Lake Louise, Banff, Alberta, Canada, October 2001

[15] W. Stallings, "Cryptography and network security—Principles and practice", Prentice Hall, 2nd edition, 1999