IPsec Protocol Acceleration using Network Processors

Ralf Lehmann
Institute for System Architecture, Chair for Computer Networks
Technische Universität Dresden
01062 Dresden, Germany
email: lehmann@rn.inf.tu-dresden.de
Mirko Benz and Stephan Groß and Maik Hampel
Institute for System Architecture, Chair for Computer Networks
Technische Universität Dresden
01062 Dresden, Germany
email: {benz, gross, hampel}@rn.inf.tu-dresden.de

ABSTRACT
Gigabit networks can not be fully utilized by today's end systems. The processing requirements of next generation network protocols with enabled encryption features will aggravate this situation. To overcome these shortcomings this paper presents an approach and prototype implementation for hardware support of IP/IPsec protocols. We outline a hardware/software partitioning, where the data path of the IP/IPsec protocol is accelerated by a network processor. In contrast to other approaches, we reuse existing software protocol stack implementations.

KEY WORDS
IPSec hardware support, Virtual Private Networks, Network Processor, high speed networks.

1 Introduction and Motivation

With the growth of the Internet and the involved increase of the available network bandwidth as well as the increase of the data volume, the processing of network protocols requires an increasing amount of overall computing of the used processors. Thus even high end systems cannot fully utilize today's high speed networks.

Due to the increasing industrial globalization and general security requirements, the local networks of cooperating institutes are more and more connected via secure IP tunnels, commonly called Virtual Private Networks. Due to the usage of secure but CPU-intensive encryption and decryption algorithms, the usable network bandwidth for interconnection is usually very small when using relatively inexpensive software solutions.

There have been quite a number of approaches for hardware support of secure communication protocols. Typically, they focus on the hardware offload of the encryption and decryption algorithms. Thus, the main protocol handling and the I/O handling have to be handled in relatively slow software solutions without getting high bandwidth usage [7] or extremely expensive hardware solutions. Furthermore these hardware crypto engines are inflexible by supporting new cryptographic algorithms.

In this paper we present an approach for IPsec hardware acceleration based on network processors. The main emphasis is to drastically improve throughput and latency. Further essential requirements include on wire compatibility to other software and hardware based VPN implementations as well as to enable existing applications to take advantage of the protocol acceleration in a transparent manner. By combining high performance and the flexibility of a software programmable architecture network processors are an ideal suit for line speed operation.

In the following chapter we compare several IPsec software implementations based on Linux. Afterwards, we describe our hardware/software partitioning of the IPsec protocol stack, which allows taking advantage of existing software implementations. Next, we present our prototype architecture, which is based on the INTEL IXP2400 network processor [6]. Here we give a brief overview of the processor architecture and the design flow. Next we discuss the mapping of the IPsec data path onto the network processor and the synchronization with the software stack as well as the operating system integration. Finally we conclude our results and address future work.

2 Linux IPSec Implementations

2.1 FreeS/Wan

The most common IPv4 IPsec implementation for systems using a Linux kernel version of 2.2 or 2.4 is FreeS/Wan [3]. This software offers the possibility of host to host connections as well as host to network and network to network interconnections. With special patches, X.509 authentication and AES cipher are supported. Although of this versatility the FreeS/Wan code is not included into the main Linux kernel due to former U.S. export limitations for strong cryptographic software and today's doubts on the quality of the source code.

2.2 Lightweight IPsec Tunnel implementation for Linux 2.4

The IPsec Tunnel implementation for Linux 2.4 Kernels [13] developed by Tobias Ringström is not a complete IPsec implementation. It's main goals are to provide an easy to use and RFC compliant tunnel implementation with simple manual keying and normal routing as well as no requirement of kernel source code patching. Due to the usage of the CryptoAPI ciphers and digests, new cryptographic algorithms are easy to use.

2.3 Upcoming Linux 2.6 Kernel

With the Linux developer kernel version 2.5.45, a new IPsec implementation is integrated into the Linux kernel [8]. It is based on USAGI, an IPv6 and IPsec project for Linux, KAME, an IPv6 and IPsec implementation for several BSD-derivatives and the Widely Integrated Distributed Environment (WIDE) project. For cryptographic calculations like authentication and encryption, the new version of the wellknown CryptoAPI is used, which is also integrated in the upcoming Linux 2.6 kernel version.

3 IPsec Acceleration Approach

Today's transport protocols like IPsec are far too complex to be completely implemented in custom hardware. Furthermore, this approach would limit the flexibility and maintainability of the solution. On the other hand, only a relatively small part of the protocol implementation has high speed processing requirements. The remaining parts consists of lower priority tasks like exception handling, buffer registration or connection management. Thus, only a small part of the implementation has to be accelerated by special hardware. The rest can still be performed by a modified software implementation, because relatively time expensive but very rare administration tasks are tolerable.

In the past only the cryptographic algorithms were accelerated using special hardware, so the processing bottleneck was shifted to the software protocol processing and the communication between the host processor and the crypto engine. By implementing the whole IP/IPsec receive and send path onto network processors, the protocol processing can be accelerated as well.

4 Data Path Partitioning

For an optimized implementation of the data path onto the network processor, the send as well as the receive path has to be analyzed. Thus, synchronization points and data between the hardware and the software implementation have to be detected.

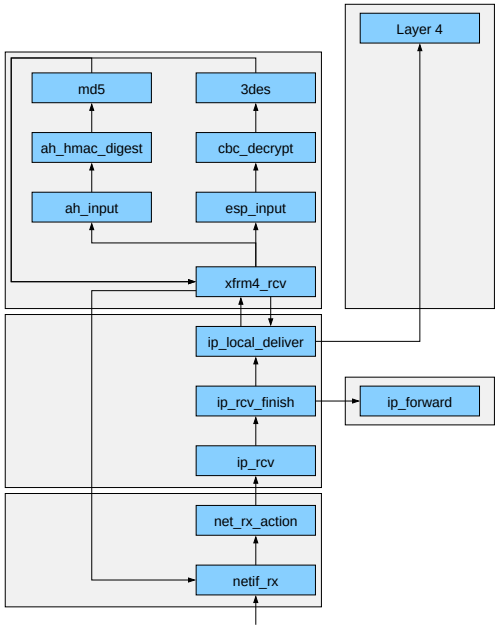

Figure 1. Linux 2.6 IPsec Receive Path

4.1 Receive Path

The schematic overview of the upcoming Linux 2.6 IPsec receive path is illustrated in Figure 1. The received network packet is handled by the device handler of the network card and put into the ethernet layer. After that – if an IP transport protocol is detected – the extracted IP packet is handled by the IP layer; at first by ip_rcv and ip_rcv_finish. Unencrypted network packets, which have to be forwarded will be commited to ip_forward. Local deliverable and encrypted IP packets will be next handled by ip_local_deliver. If the packet is coded, decoding is done by xfrm4_rcv, otherwise it is sent to the next protocol layer handler.

At first, the authentication header of the encrypted network packet is tested and – if it succeeded – the packet is decrypted. After finishing decoding, if the network packet is to be delivered locally, xfrm4_rcv sends the data back to ip_local_deliver, which will put it into the layer 4 protocol handler. If the decrypted packet has to be forwarded, the data will be sent by xfrm4_rcv to the beginning of the network receive path, to netif_rx and the – now plain – network packet will be handled by the network stack.

4.2 Send Path

Figure 2 shows a schematic overview of the Linux 2.6 IPsec send path.

In transport mode for host to host connections the data from upper layers are received by ip_queue_xmit. If the data have to be encrypted, the encoding is done by esp_output and the 3des encryption functions and thereafter the data will be authenticated by ah_output and the md5 hash func-

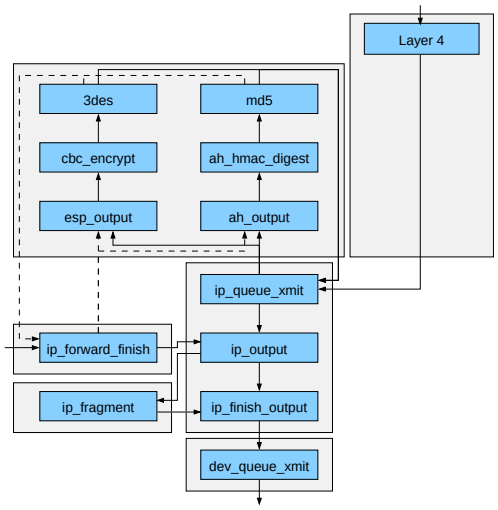

Figure 2. Linux 2.6 IPSec Send Path

tions. Afterwards, the encrypted or in case of an unencrypted data connection the plain data packet is sent from ip_queue_xmit via ip_output, in case of necessary fragmentation ip_fragment, to ip_finish_output. Next the IP packet is put to dev_queue_xmit and sent to the network.

In case of network routing, the tunnel mode will be used. The packets will be processed by ip_forward_finish. Same as in ip_queue_xmit, if the network packet has to be encrypted, the data will now be encoded similar to the transport mode. After that, the IP packet is put to ip_finished_output and sent to the network.

4.3 Data Path Synchronization

Since only the described IP and IPsec send and receive data path will be implemented on the network processor, several data interfaces are necessary for the accelerated implementation.

4.3.1 Receive Path

- Non-IP packets have to be put into netif_rx from the Linux network stack.
- In case of IP forwarding the network packets have to be sent to the forward handler of the Linux network stack.
- After finishing IP and IPsec handling, the extracted data have to be delivered to the layer 4 handler of the Linux kernel.

4.3.2 Send Path

- If the application data have to be sent via the IP protocol, the layer 4 implementation has to put the data into the IP queue of the hardware accelerator.
- Forwarded packets must be sent to the IP forwarder part of the hardware IP send path.
- Since the network processor handles the network device as well, non IP packets have to be put to the device queue of the hardware network stack.

4.3.3 Synchronization data

To enable full IP/IPsec protocol processing in the network processor, the IP stack implementation needs several data for encryption and routing desicions.

The IPsec Security Associations (SA), stored in the SA Database (SAD), specify, if and how the IP packets have to be encrypted and authenticated. A Security Association consists of the source and the destination IP address, the IPsec protocol (AH or ESP), the Security Parameter Index (SPI), the used algorithms and the secret encryption keys.

The Security Policies (SP), stored in the SP Database (SPD), defines the usage of the Security Associations. Each SP contains the IP adresses of the communicating partners, the used mode (tunnel or transport mode), the IP protocol to be encrypted and the port of the protocol, the IPsec protocol (ESP or AH) and the connection direction (sending or receiving).

These data are stored in the Linux kernel xfrm_state structures, so the network processor protocol stack has to access these data.

The final implemented flow of the synchronization data is illustrated in figure 3.

5 IXP Data Path Implementation

5.1 Data Path Measurement

5.1.1 Measurement environment

The measurement of the IPsec data path processing was done between two hosts with following configuration: Host A – Mobile P3 (1,13GHz), 512 MB-SDRAM, 100MBit LAN Intel Etherexpress Pro 100 and Host B – Mobile P4 (1,8GHz), 512 MB-DDRAM, 100MBit LAN Sis 900.

Both hosts used kernel version 2.5.66 and Debian Linux 3.0 Operating System. The kernel itself was configured to support IPsec protocol version 4, Crypto-API and OProfile [11].

The IP security connection between host A and B was configured to work in transport mode with Authentication Header and Encapsulated Security Payload. The algorithms used were HMAC-MD5 for Authentication Header and 3DES in CBC-Mode for Encapsulated Payload. The keys were set manually with preshared keys.

The network load was generated using netperf [5]. Netperf was configured to produce heavy TCP network

traffic in a way that it was possible to separate the send and receive path in Kernel sources when measuring with OProfile. The data stream was set up to send large packets from Host A to Host B and receive only small ones.

All measurements were done on host A with OProfile. For measurement the hardware processor events CPU_CLK_UNHALTED and DATA_MEM_REFS have been mainly used to get informations about the CPU utilization and memory accesses. In all measurements the OProfile daemon was configured to count samples every 1000 events. With higher granularity the system got frozen.

5.1.2 Processor load

Kernel Function	Time (%)	Memory (%)
md5	5,4	3
ah_output	0,8	0,7
esp_output	0,3	0,2
cbc_encrypt	2,6	2,4
3des_encrypt	76	78
others	8,9	11,2
sum	94	96
ip_queue_xmit	0,3	0,1
ip_output	0,2	0,1
ip_finish_output	≈0	≈0
others	4,4	3,8
sum	5	4
dev_queue_xmit	0,5	0,5
sum	0,5	0,5

Table 1. IPsec send path measurement results

Kernel Function	Time (%)	Memory (%)
md5	8	3,8
ah_input	0,8	0,6
esp_input	0,2	0,2
cbc_decrypt	3,6	8,8
3des_decrypt	74	74
others	7,7	9
xfrm4_rcv	0,2	0,2
sum	94,5	96,6
ip_local_deliver	0,1	0,1
ip_rcv_finish	≈0	≈0
ip_rcv	0,2	0,1
others	4,7	3,1
sum	5	3,3
netif_rx	0,5	0,1
sum	0,5	0,1

Table 2. IPsec receive path measurement results

Tables 1 and 2 show the measured values of the send respectively receive path. The time values describe the percentage of time the functions need for packet processing, relative to the overall processing time for the IP/IPsec protocol stack.

The memory value illustrates the amount of the portion of memory accesses done by the referenced function in relation to the overall memory references in the protocol stack.

As indicated, these measurements are based on the usage of the CPU_CLK_UNHALTED event.

Both tables show also, that the time spent at the network layer is both at send and receive path less than 1% of CPU time. So, very little computation to the packet has been done by that layer.

The measured values show as expected that the main CPU time is spent at the cryptographic functions. These functions directly limit the IPSEC throughput at all. In particular the functions needed for Authentication Header, in this case md5 and hmac, do not need that much time to process. The most time consuming function is 3DES.

The time spent for processing data packets is much more smaller than one percent.

5.1.3 Memory access

These measurements are based on the usage of DATA_MEM_REFS event, which reacts on all memory references, cachable and non cachable.

At the network layer the memory is refenced only a few times, supposeable due to the zerocopy mechanism in Linux network architecture. Memory transactions are heavily done in the CryptoAPI functions. In fact there are over 95% of all memory accesses. Therefrom belong over 75% to the 3DES implementation. The hash algorithms HMAC and MD5 are relatively leightweight with about 3% each.

5.2 IXP Architecture Overview

The INTEL IXP2400 is a fully programmable network processor unit (NPU) that implements a high-performance parallel processing architecture. It combines a high-performance Intel® XScale™ core, multiple memory units, a PCI interface, a multi-purpose network interface and eight 32-bit independent multi-threaded microengines on a single chip.

A major problem for the design of network processors is the gap between CPU and memory performance. Hence, the architecture allows having multiple pending memory transactions. While waiting for the operation to complete another thread may perform computing tasks. A new feature with the second generation of INTEL NPUs is hyper task chaining. This allows improving the efficiency of pipeline processing by having direct access to next neighbor transfer registers. As a consequence, synchronization

time is greatly reduced compared with SRAM based mechanisms.

5.3 Data Path Implementation

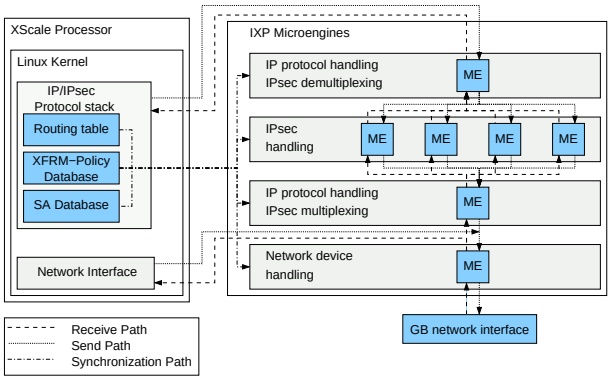

Figure 3. IPSec Protocol processing implementation

Following the measurement results, the IXP receive data path implementation is done as illustrated in figure 3 using pipeling and parallelizing techniques.

The first microengine (ME) represents the network layer and acts as a networking device. It receives the Ethernet packages and handles the Ethernet packet information. Non-IP packets are directly sent to the unchanged Linux software network stack – in this case the IXP implementation acts like a non-intelligent network interface card. In case of a received IP packet it will be forwarded to the next microengine, which is representing the IP layer. This second microengine handles the IP header information, does checksum computing and acts as a multiplexer to distribute the packets to a set of microengines, which are implementing the IPsec layer.

These microengines handle all the IPsec decryption and authentication. Since currently only 3DES, MD5 and SHA-1 are supported, each of the IPsec microengines is running identical code. With adding more cryptographic algorithms, a mechanism for dynamic changing the microengines program code for load balancing has to be implemented.

At the end of the network stack pipeline a demultiplexing microengine is situated, which serializes the IP stream and delivers it to higher protocol layers.

The send path implementation – illustrated in figure 3 too – is working similar but in reverse order.

This implementation of the IP/IPsec protocol stack does not mandatory shorten latency time but it is raising drastically the data-throughput.

To improve the latency time, the crypto engines have to be optimized, because these algorithms take the largest portion of overall computing time as shown in section 5.1.

5.4 Crypto Engine Optimization

The crypto core of the IPsec protocol uses both authentication algorithms and symmetric block ciphers. In our test scenario we used the cryptographic hash functions MD5 and SHA-1 for the authentication part and 3DES to conceal the transmitted packages.

First tests have shown the good performance of MD5 and SHA-1 without any sophisticated optimizations. We therefore concentrated on the optimization of the 3DES. Using the 3DES algorithm a plaintext is encrypted by concatenating three DES functions using three different keys. Decryption works just the other way round. As each DES function relies on the output of the previous one, there is obvioulsy not much space for optimizations. Let us therefore have a closer look to the inner of a DES function. It can be divided into two main parts: the generation of round keys from the encryption key and the actual encryption part. The later one takes a plaintext and after a permutation divides it into a left and right part. Now sixteen rounds of a so called Feistel cipher [9] are processed. In each of these rounds the right part of the input is first encrypted using the function f and then combined with the left part by a bitwise xor operation. The function f is based on the two principles of diffusion and confusion by using an expansive permutation function and specific S-boxes respectively. After all, the plaintext should be mixed up completely.

There are several related works on the optimization of DES. E.g., for hardware implementations Broscius and Smith [2] have presented a composite optimization approach by decomposing a single round into several parallel computations by generating the necessary round keys in advance, computing each round in a pipelined parallel structure[1], and performing the system I/O in parallel with the encryption computation.

For software implementations you gain the best performance by realizing the S-box calculations via table lookups [12]. As the Intel IXP 2400 lies somewhere in between software and hardware, we are currently working on combining both mentioned approaches to maximize encryption throughput.

5.5 Implementation Results

In result to the implementation, a significant speedup of IPsec processing was not visible. The reason is the required use of large S-boxes for the crypto engines. Unfortunately the IXP2400 microengines have only a limited amount of local memory, comparable with cache memory of common desktop processors. Thus there were a lot of time consuming memory accesses.

Hence, for efficient use of the described acceleration approach upcoming network processors like the IXP2800 or even the IXP2850 have to be used. Nevertheless, this

[1] Unfortunately, this works only for operation modes which do not require ciphertext feedback like CBC does. Therefore, this approach is not applicable in our case.

implementation demonstrates the possibilities of this acceleration approach.

6 Related work

Much attention has been focused on the design of cryptographic accelerators for IP security speedup of software VPN implementations.

An approach for FreeS/WAN acceleration is described in [14]. It suggests to split the different functionality of KLIPS into three parts and use generic engines for acceleration of these function units, the tunnel processing engine, protocol processing engine and the crypto processing engine.

A software patch for using hardware crypto engines with FreeS/WAN is specified in [10]. It describes the usage of an asynchronous crypto API to improve the speedup of the hardware crypto engine.

There is a number of ASIC based approaches for IPsec acceleration, like the Hifn products as specified in [4], but only with a limited flexibility when implementing new cryptographic algorithms and changed or enhanced protocols.

Comparable to our IP/IPsec acceleration approach, a TCP acceleration based on network processors is presented in [1].

7 Conclusions and future work

Supporting high performance networks and advanced services at the same time presents a challenge for today's protocol implementation architectures. Even high end systems can not fully utilize high speed networks using IP security protocols. Therefore, we presented a hardware supported acceleration approach for the IP/IPsec protocol.

Designing protocol engines is a complex task. As a consequence we decided to take advantage of an existing software IP/IPsec stack and extracted the data path which was accelerated by specific hardware. For data path partitioning systematic methods were successfully used to reduce partitioning and implementation time.

Future work includes other cipher algorithms than 3DES in our crypto engine. We are specifically working on an implementation of the new Advanced Encryption Standard (AES), the designated successor of DES.

References

[1] Mirko Benz and Ralf Lehmann. TCP Acceleration based on Network Processors. In *IASTED International Conference on Communications, Internet, and Information Technology (CIIT)*, St. Thomas, US Virgin Islands, 2002.

[2] Albert G. Broscius and Jonathan M. Smith. Exploiting Parallelism in Hardware Implementation of the DES. In *Crypto '91*, number 576 in LNCS, pages 367–376, Berlin, 1992. Springer Verlag.

[3] *Linux FreeS/WAN – http://www.freeswan.org*. Internet WWW document.

[4] *Hifn – Intelligent Secure Networking – http://www.hifn.com*. Internet WWW document.

[5] HP Information Networks Division. *Netperf: A Network Performance Benchmark – http://www.cup.hp.com/netperf/NetperfPage.html*. Internet WWW document.

[6] Intel Corp. *Intel Network Processors – http://www.intel.com/design/network/products/-npfamily/*. Internet WWW document.

[7] Ralf Lehmann and Mirko Benz. Analysis of TCP/IP Protocol Processing in Gigabit Networks. In *Proceedings of the 2002 WSEAS International Conference on Information Security, Hardware/Software Codesign, E-Commerce and Computer Networks*, Rio de Janeiro, Brazil, October 2002.

[8] *The Linux Kernel Archives – http://www.kernel.org*. Internet WWW document.

[9] Alfred J. Menezes, Paul C. van Oorschot, and Scott A. Vanstone. *Handbook of Applied Cryptography*. CRC Press, Boca Raton, 1997.

[10] Colubris Networks. *FreeS/WAN Hardware acceleration patch – http://sources.colubris.com/en/projects/FreeSWAN/*. Internet Document.

[11] *OProfile – http://oprofile.sourceforge.net*. Internet WWW document.

[12] Andreas Pfitzmann and Ralf Assmann. More Efficient Software Implementations of (Generalized) DES. *Computers & Security*, 12(5):477–500, 1993.

[13] Tobias Ringström. *An IPsec tunnel implementation for Linux 2.4 – http://ringstrom.mine.nu/ipsec_tunnel/*. Internet WWW document.

[14] Bart Trojanowski. *FreeS/WAN – KLIPS Hardware Acceleration Notes – http://jukie.net/∼bart/linux-ipsec/freeswan-hardware-acceleration-draft-4.txt*. Internet Document.